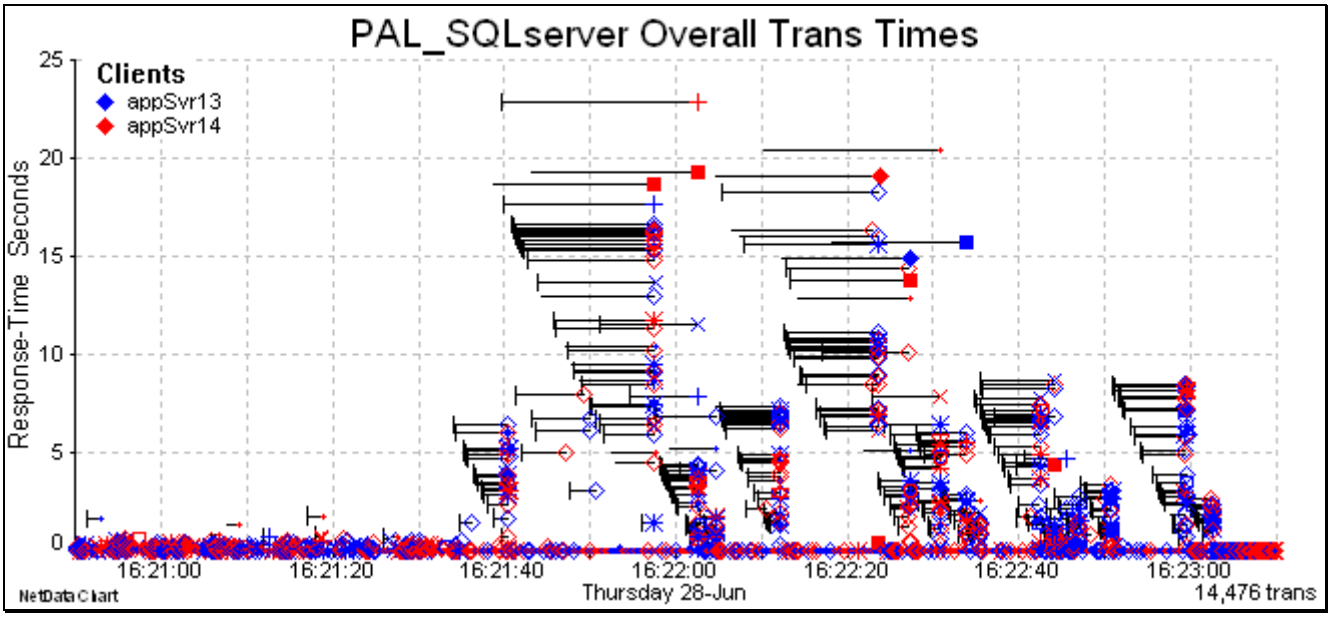
Measure IT Pty Ltd

NetData

visualising IT performance



User Guide

DECODERS

Commercial-in-Confidence

This document contains commercially-sensitive and proprietary technical information. Measure IT requires that it not be released in any form to any personnel other than NetData licensees and those addressed explicitly and directly by Measure IT.

The analytical techniques and charting methods described in this document are protected by copyright.

Contact Measure IT: Bob@netdata-pro.com

CONTENTS

1 Introduction 1

2 Web Services 2

2.1 Distinguishing Transactions Encrypted in SSL 2
2.2 Suppressing Transaction Characterisation in SSL-TLS Traffic 2
2.3 Summarising Digital Certificate Chains in SSL Handshakes 3
2.4 Digital Certificate Validity 4
2.5 Displays of SSL Renegotiation (2022) 5
2.6 Secure Session Renegotiation in SSL (2015) 6
2.7 Recognising 100-Continue Round-Trips in HTTPS Transactions 7
2.8 TLS Transactions Initiated by Either Client or Server 10
2.9 TLS Client Messages without Response 10
2.10 Displaying HTTP Chunk Contents 11
2.11 Quick UDP Internet Connection (QUIC) Versions Q043, Q044 12
2.11.1 One Extra Round-Trip Time (1-RTT) 12
2.11.2 Zero Extra Round-Trip Times (0-RTT) 14
2.11.3 Tracking Sequence Gaps 15
2.11.4 Transaction Characterisation 15
2.11.5 Measuring Round-Trip Times 17
2.12 Google QUIC Q050 20
2.13 Extracting Files Downloaded by HTTP 21
2.14 Alteryx Data Analytics and Workflow 22
2.15 HTTP Pipelining 25
2.16 WebMethods and Java RMI 31
2.17 AJPv1.3 (Apache JServ Protocol) and Engaged Threads 32
2.18 Serialised Java Objects 32
2.19 Modelling Web Transactions with a Polling Mechanism 34
2.20 User-Agent Types 38
2.21 Extracting Fields in HTTP Query Strings 40
2.22 Tables in JSON Messages 40
2.23 Tables in XML Documents 40
2.23.1 Viewing Tables in XML Messages 40
2.24 Tables in HTML Web Documents 41
2.25 Presence of Particular Fields in HTTP Query Strings 42
2.26 Extended HTTP Response Signatures 43
2.27 Symantec (ex MessageLabs) Proxy Server 43
2.28 Web NTLM Proxy Authentication 43
2.29 Port 1080 VPN Detection 44
2.30 OpenUI Protocol (2012) 44

3 Database Management 45

3.1 SQL (Structured Query Language) 45
3.2 Oracle SQLnet and TNS 47
3.2.1 Dialect Hints 48

3.2.2 Composite Commands 49
3.2.3 Response Messages 51
3.2.4 Terminating SQLnet Query Group Transactions 54
3.2.5 Oracle SQLnet Missing Column Definitions 55
3.2.6 Different Client Dialects 56
3.2.7 Oracle SQLnet Version 314 56
3.2.8 Oracle SQLnet Piggyback Functions 57
3.2.9 Severely Truncated Oracle SQLnet Traffic 57
3.2.10 Encrypted Oracle SQLnet Transactions 58
3.3 Oracle Hyperion SQR Production Reporting 59
3.4 Oracle RAC Cache Fusion 59
3.5 Microsoft SQL Server TDS 61
3.5.1 Progressive Responses 64
3.5.2 Session Multiplex Protocol 65
3.5.3 Error Messages 66
3.5.4 Cancelled Transactions 67
3.5.5 Distributed Transactions 68
3.5.6 In-Procedure Status Blocks 73
3.5.7 SQL Server SSL Login 75
3.5.8 SQL Server Transact-SQL System Stored Procedures 76
3.5.9 Describing Transact-SQL Statements 77
3.5.10 Tabular Data Stream Data Formatting 78
3.6 Microsoft SQL Monitor, Port 1434 78
3.7 Microsoft SQL Server Database Mirroring 79
3.8 IBM DB2 DRDA and DDM 80
3.8.1 Transaction Signatures 81
3.8.2 Data-Row Format Specifications 82
3.8.3 Error Indications 85
3.8.4 Data Blocks and Minimising Round Trips 87
3.8.5 DB2 Queries Re-opened or Data Manipulation Commands Re-executed 91
3.8.6 Unique IDs for Sections in DB2 Databases and Packages 92
3.8.7 DB2 Log Shipping for High Availability Disaster Recovery 94
3.8.8 DB2 Group Transactions 96
3.8.9 DB2 Sync Point Control 97
3.9 IBM Business Monitor and DB2 Monitor 98
3.10 Java Objects and XML in DB2 Transactions 98
3.11 Distributed Transactions 99
3.12 Recognising Redundant Queries Dependant on Current Time 102
3.13 MySQL 103
3.14 Redis Inter-Node Traffic 104
3.15 SQL Like-Clause Performance Weaknesses 105
3.16 Progress Software Database and Application Servers 106
3.17 Teradata Unity and Database Traffic 106

4 File Service 107

4.1 Server Message Block Version 2 (SMB2) 107
4.2 Signed SMB Messages 107
4.3 SMB2_Transform_Header and SMB2_Compression_Transform_Header 108
4.4 Forming User Transactions 110
4.5 FileNet Traffic Decoding 111
4.6 Tivoli Storage Productivity Centre Device Server 111

4.7 FileMaker Pro Host Contact 111
4.8 QVOD Player Search for SMV File Servers 111
4.9 Google Logging Service (glog) 111
4.10 Sun RPC/UDP Call-Reply Matching 115
4.11 Working With Huge File-Server Transaction Trees 116
4.12 Novell NCP File Service 119
4.13 IBM GPFS (General Parallel File System) 119
4.14 EMC Documentum Content Server 119

5 File Transfer 120

5.1 Missing Files Reported by SMB Commands 120
5.2 Aspera FASP 120
5.3 DST Systems AWD (Automated Work Distributor) 120
5.4 AOL Instant Messaging and OSCAR File Transfer 121

6 VoIP and Media Transfer 122

6.1 Multiple RTP Streams in a Single Connection 122
6.2 T.38 Facsimile over IP (FoIP) 123
6.3 ShoreTel ShoreWare Call Management 126
6.4 Q.931/LAPD Call Signalling 127
6.5 Data Streaming for Teams and Skype for Business 129
6.6 Unknown Traffic with UDP Port 3478 (STUN and Teams) 129
6.7 Microsoft Teams Codec Packet Formats 129
6.8 Displaying RTP (VoIP) Transit Times 130
6.9 Miralix Phone Monitor 131
6.10 VoIP/SCCP (Skinny) Decoder 131
6.11 VoIP SIP Decoder 134
6.12 Avaya RTA Interface 134
6.13 Universal Alcatel Signalling Protocol 135
6.14 RTMP Chunk Stream Decoder 135
6.15 MGCP Decoder 135

7 Peer to Peer 136

7.1 Skype Traffic Detection (2012) 136
7.2 BitTorrent Local Peer Discovery (LPD) 137
7.3 Short Message Peer-to-Peer Protocol 137
7.4 BitTorrent 137
7.5 Lotus Sametime Connect Peer-To-Peer File Transfer 137
7.6 QVOD Peer-to-Peer Video Player and Streaming Media Server 138

8 Mail Service 140

8.1 Experian QAS (QuickAddress) 140

9 Messaging Systems 141

9.1 WhatsApp Traffic 141
9.2 Solace Message Format 146
9.2.1 Assured Delivery and Flow Control 147
9.2.2 Messages In Flight 148
9.3 Java Message Service Decoder (2012) 149
9.4 SwiftMQ JMS for VMware VDM Connection Servers 150

9.5 IBM Java Message Service of a Service Integration Bus 154
9.6 IBM WebSphere MQ 154
9.6.1 Slow Power System Backplane Transfers....155
9.7 IBM MQ Series Telemetry Transport (MQTT) 156
9.8 Apache ActiveMQ....157
9.8.1 Charting Propagation Displays....157
9.9 Microsoft Message Queuing 159
9.10 Computer Associates Message Queuing (CAM) 162

10 Trading Systems 163

10.1 Financial Information eXchange (FIX)....163
10.1.1 Session Protocol and Message Sequence Number Tracking....164
10.2 BATS/Cboe (Chicago Board Options Exchange) Traffic....165
10.3 Australian Energy Market Transaction Decoding (2012) 167

11 System Management....168

11.1 SAS Intelligence Platform....168
11.2 Media Gateway Control Protocol (MGCP)....173
11.3 Topology Services of IBM Reliable Scalable Cluster Technology 174
11.4 IBM Maestro Tivoli Workload Scheduler (TWS) 174
11.5 IBM Tivoli Director Agent....174
11.6 IBM Sterling Connect:Direct 175
11.7 IBM Download Director 175
11.8 Cisco Gateway Load Balancing Protocol (GLBP)....175
11.9 Spector CNE Decoder 176
11.10 Linux Audit Logging....177
11.11 Ganglia Monitoring System 179
11.12 Compuware Distributed License Management (DLM)....179
11.13 Cisco Meraki Cloud Dashboard 180
11.14 Cisco EnergyWise 180
11.15 Microsft Office Groove LAN Device Presence Protocol (DPP)....180
11.16 Remote Management Control Protocol (RMCP) 180
11.17 Argent Infrastructure Monitoring....180
11.18 Snare Event-Log Collection 180
11.19 Nagios NSClient++ Windows Monitor....180
11.20 BMC Control-M Workflow Manager 181
11.21 Microsoft High Performance Computing (HPC) 182
11.22 Microsoft Operations Manager Health Service....183
11.23 Microsoft Cluster Service and Multicast Request Reply 183
11.24 Microsoft Remote Desktop on UDP with Extended Transport....185
11.25 Flexera (ex Globetrotter) Software FlexLM Licence Manager....188
11.26 Citrix ICA Session Reliability....188
11.27 Netgear Switch Discovery Protocol 188
11.28 IBM and SAS Grid Computing....189
11.29 Simple Network Management Protocol (SNMP)....190
11.30 Online Certificate Status Protocol....191
11.31 BlueCoat Authentication and Authorization Agent (BCAAA)....193
11.32 Citrix NetScaler Metric Exchange Protocol....193
11.33 Citrix NetScaler High-Availability Services....194
11.34 TACACS+....194
11.35 Novell Service Location Protocol (SLP)....194

12 Naming Systems 195
12.1 Link-Local Multicast Name Resolution (LLMNR) 195
12.2 Multicast DNS 195
12.3 NetBIOS Name Resolution 195
12.4 JBOSS HA Pooled Invoker (Port 4446) and RMI Naming Service (1099) 196
13 Security 197
13.1 LDAP Logical Expressions 197
13.2 Internet Protocol Security (IPsec) 198
13.3 Matching IPsec Data Streams in Multiplexed Connections 201
13.3.1 Investigating Throughput Issues with IPsec Traffic 203
13.4 Datagram Transport Layer Security (DTLS) 204
13.5 Gemalto SafeNet ProtectV Manager 205
13.6 IBM Tivoli Directory Server (ITDS) 206
14 Backup Systems 207
14.1 IBM ProtecTIER System Storage Deduplication and Replication 207
14.2 Acronis 209
14.3 CommVault 209
14.4 Symantec NetBackup Utility and Client Daemons 209
14.5 NetApp snapMirror and snapVault 209
15 Print Services 210
15.1 Common Unix Printer System (CUPS) Browse Protocol 210
15.2 Canon BubbleJet Network Protocol 210
15.3 NTware UniFlow Printing Management 210
15.4 HP Printer Job Language 210
15.5 Equitrac Printing Management and Cost Accounting 212
15.6 RPC Access to Spooling Services in Print Servers 213
16 Remote Control 215
16.1 TeamViewer 215
16.2 Secure Shell 215
16.3 Remote Desktop Protocol: UDP Transport Extension Protocol v1 216
16.4 Remote Desktop on UDP with Extended Transport Version 2 217
17 Virtual Private Networks and Tunnelling 220
17.1 Teredo Tunnelling IPv6 over UDP 220
17.2 ZeroTier One Connectivity Traffic 223
17.3 WireGuard VPN Tunnels 223
17.4 OpenVPN 223
17.5 GPRS Tunnelling Protocol 223
17.6 Point-to-Point Protocol over Ethernet 223
17.7 General Routing Encapsulation 223
17.8 Unscrambling the PPP Omelette 224
18 WiFi 230
18.1 WiFi Management Traffic 231

18.2 Radiotap WiFi Decoding....233
18.3 Control And Provisioning of Wireless Access Points (CAPWAP) 233

19 Routing Protocols 234

19.1 BGP Decoding....234
19.2 EIGRP Decoding....236
19.3 IS-IS Decoding....238
19.4 OSPF Decoding....239
19.5 ES-IS Decoding....241
19.6 IGMP and Protocol Independent Multicast (PIM)....242
19.7 Shortest Path Bridging (SPB) and TRILL....243
19.8 MPLS (MultiProtocol Label Switching)....245
19.9 Pseudo-Wire (PW) Emulation in MPLS Networks....245
19.10 Label Distribution Protocol (LDP) for MPLS Networks....245

20 OSI Protocols....247

20.1 LLC OSI Network Layer....247
20.2 Cisco HDLC and Frame Relay Frame Formats....248
20.3 Serial Line Address Resolution Protocol (SLARP) and Inverse ARP....249

21 Industrial Protocols....250

21.1 Pilz SafetyNET p Real-Time Frame Network (RTFN), IEC 61158-series Type 22 - Fieldbus....250
21.2 Modbus-TCP....250
21.3 Encrypted Solace Telemetry Traffic....252
21.4 Message Queuing Telemetry Transport (MQTT)....253
21.5 Internet Relay Chat....253

22 Packet Capture....254

22.1 Supported Wireshark Frame Formats....254
22.2 Wireshark PCap New Generation Capture Files....255
22.3 Millisecond TCPdump Capture-File Format....257
22.4 F5 Ethernet Trailers....257
22.5 CheckPoint Firewall-Monitor....260
22.6 Citrix NetScaler Nstrace Capture Files....264
22.7 Bluetooth Smart (Low Energy)....266
22.8 Juniper Ethernet Encapsulated Packets....271
22.1 National Instruments Observer....271
22.1 Ethernet Interspersing Express Traffic (IET)....272
22.2 ProfiShark Metadata with High-Resolution Timestamps....272
22.3 NetMon 3.3 Capture Files....273
22.4 NetMon 3.4....273
22.5 Linux ‘Cooked-mode’ Captures....275
22.6 HP-UX Tracing Tool Nettl....275
22.7 TCPdump Null File Format for IP-Only Frames....276
22.8 Citrix NetScaler Nstrace Files....276

23 Miscellany....278

23.1 Characterising Transactions of Unknown Types....278
23.2 Characterising Incomplete Transactions....279

23.3 Network Time Protocol 281
23.4 iPerf Decoding 283
23.5 VMware Small Messages Sent by Clients 284
23.6 Nessus Port Scanner 284
23.7 Windows Update Delivery Optimization (WUDO) 284
23.8 Mapping of Airline Traffic over Internet Protocol (MATIP) 285
23.9 SITA Baggage Source Messages (BSM) 286
23.10 SITA Airport Messaging and Sabre Data Source 286
23.11 Visualising Automated Bag Drop Performance 289
23.12 Riverbed Out-of-Path Optimisation Traffic 293
23.13 Inter-Juniper Accelerator Traffic 293
23.14 BlackBerry Enterprise Server (BES) 293
23.15 Micros Systems 9700 HMS Point-of-Sale Equipment 293
23.16 Bentley Systems ProjectWise 293
23.17 Eclipse Test & Performance Tools Platform (TPTP) 294
23.18 AMF in Adobe Flex for VMware Management Consoles 295
23.19 Data-Compression Benefits with Connect:Direct Traffic 299
23.20 TCP Port 4505 301
23.21 ZK Software Biometric Attendance Management 301
23.22 Verint Quality Monitoring 302
23.23 Memcache 302
23.24 Lotus Notes 302
23.25 Decoding RADIUS with PEAP 303
23.26 Autonomy Qfiniti Agent Monitoring Engine (AME) 303
23.27 IPFX (ex Performance Solutions) CTI 303
23.28 Artsoft 303
23.29 IBM Content Manager OnDemand 303
23.30 IBM IMS Connect 303
23.31 Dropbox LAN Sync Decoder 303
23.32 SDI Decoder 303
23.33 BMC BEM to IBM Tivoli Netcool/Omnibus 304
23.34 Unknown Protocols Using Ports 4892 and 12000 304
23.35 SAP Gateway (RFC) Decoder 305
23.36 Siebel Customer Relationship Management (CRM) 306
23.37 Sapia Open Source Ubik 308
23.38 Blocks Extensible Exchange Protocol (BEEP) 309
23.39 Android Debug Bridge and Runtime Logging 310
23.40 Protocol for Operations Discovery (pfod) 310
23.41 Bloomberg Professional Services 311

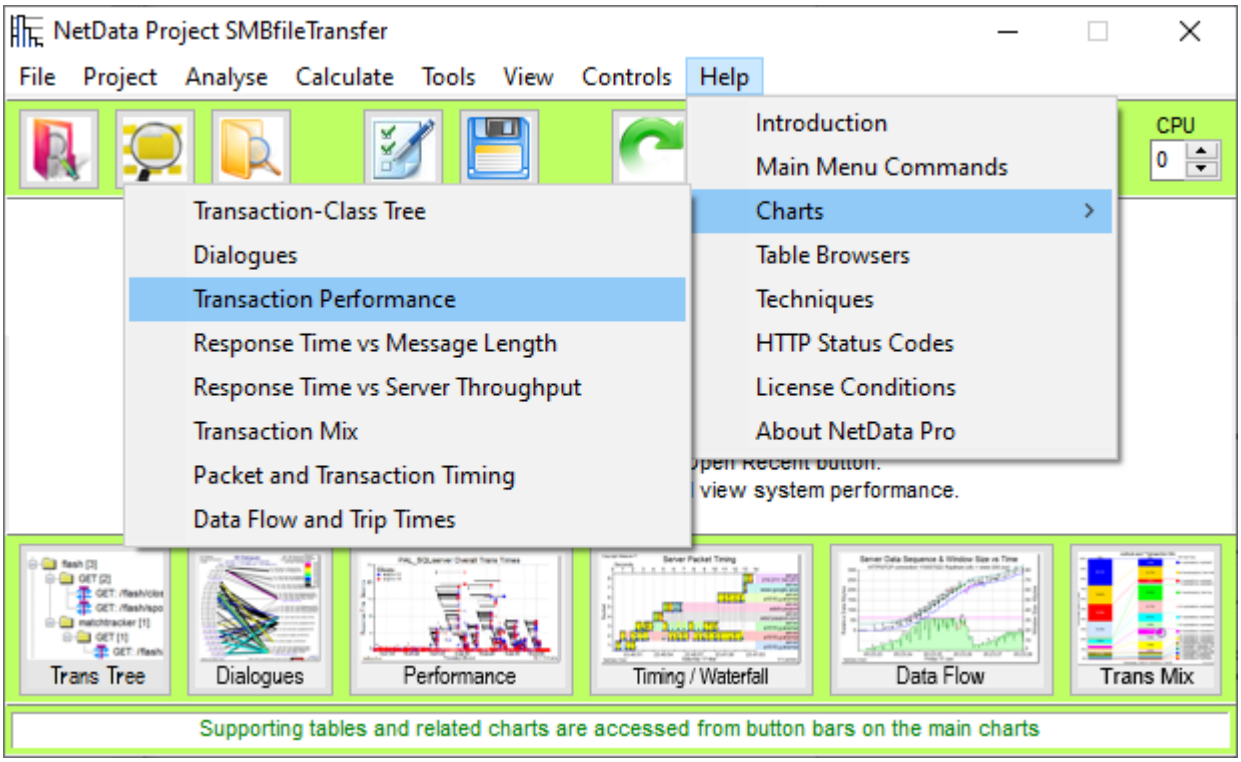
1 Introduction

This user guide to decoders is one of four sources of help and advice on using NetData:

- NetData Charting Guide

  A brief outline of all a user needs to install NetData and get started, generating charts, controlling overlays and zooming into the interesting details.

- Online Help
  - Pop-up descriptions of individual controls
  - Context-sensitive help for major charts and windows requested with '?' button
  - Discussion of major topics listed in Help menu



- Decoders User Guide
- Advanced User Guide to Analytical Techniques

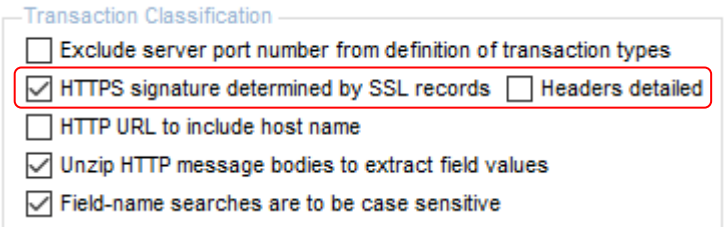
Because many of the following articles first appeared in NetData release notes and user guides when NetData functions were added or revised, their illustrative screenshots may be out of date. In most cases it is only the formatting controls that have been reorganised and updated to accommodate additional functions, but the capabilities have not changed.

2 Web Services

2.1 Distinguishing Transactions Encrypted in SSL

NetData forms signatures of SSL transactions from the lengths and types of the SSL records in request and response messages. Small variations in record lengths are masked by ‘generalised’ lengths such as ‘[4**]’ for all application-data lengths between 400 and 499 bytes, and ‘[*]’ for all lengths over 999 bytes. This scheme avoids characterising huge numbers of transaction types while distinguishing many different types that may have quite different response times. It is very useful to see even with encrypted transactions whether poor performance is confined to particular types of transactions.

An option in the Decoding page of controls further refines SSL transaction signatures. A ‘Headers detailed’ checkbox stops NetData from generalising the length of the first application-data record in a message. That first record probably conveys an HTTP request or response header, and the extra detail in the signatures may help distinguish transaction types that would otherwise look the same. This option may be more useful when transactions are generated by a load generator such as LoadRunner, because all the clients use the same software (i.e. the same browser) and the same types of requests are more likely to have the same header length.



2.2 Suppressing Transaction Characterisation in SSL-TLS Traffic

NetData can identify transactions in SSL-TLS traffic only by noting reversals in the direction of data flow, on the assumption that the transactions are request-response pairs and no transactions run concurrently unless they are pipelined (responses must be in the same order as requests). There are many protocols for which these assumptions don’t apply, and, if the traffic is encrypted with SSL, NetData’s transaction characterisation is likely to be invalid.

Normally the invalid transactions don’t present an analysis problem provided they are ignored or not displayed on charts. However, for cleaner charts, and to save some analysis time, the characterisation of SSL data transactions can be suppressed by assigning to the relevant services the new application type ‘SSLmpxS’, regarded as a dialect of SSL. For this dialect, NetData characterises only handshake transactions and transactions of the internal class that measure the delay from the appearance of a Change Cipher record and the first application-data record. These internal

transactions identify significant delays in an application’s use of a secure session, and sessions that are never used.

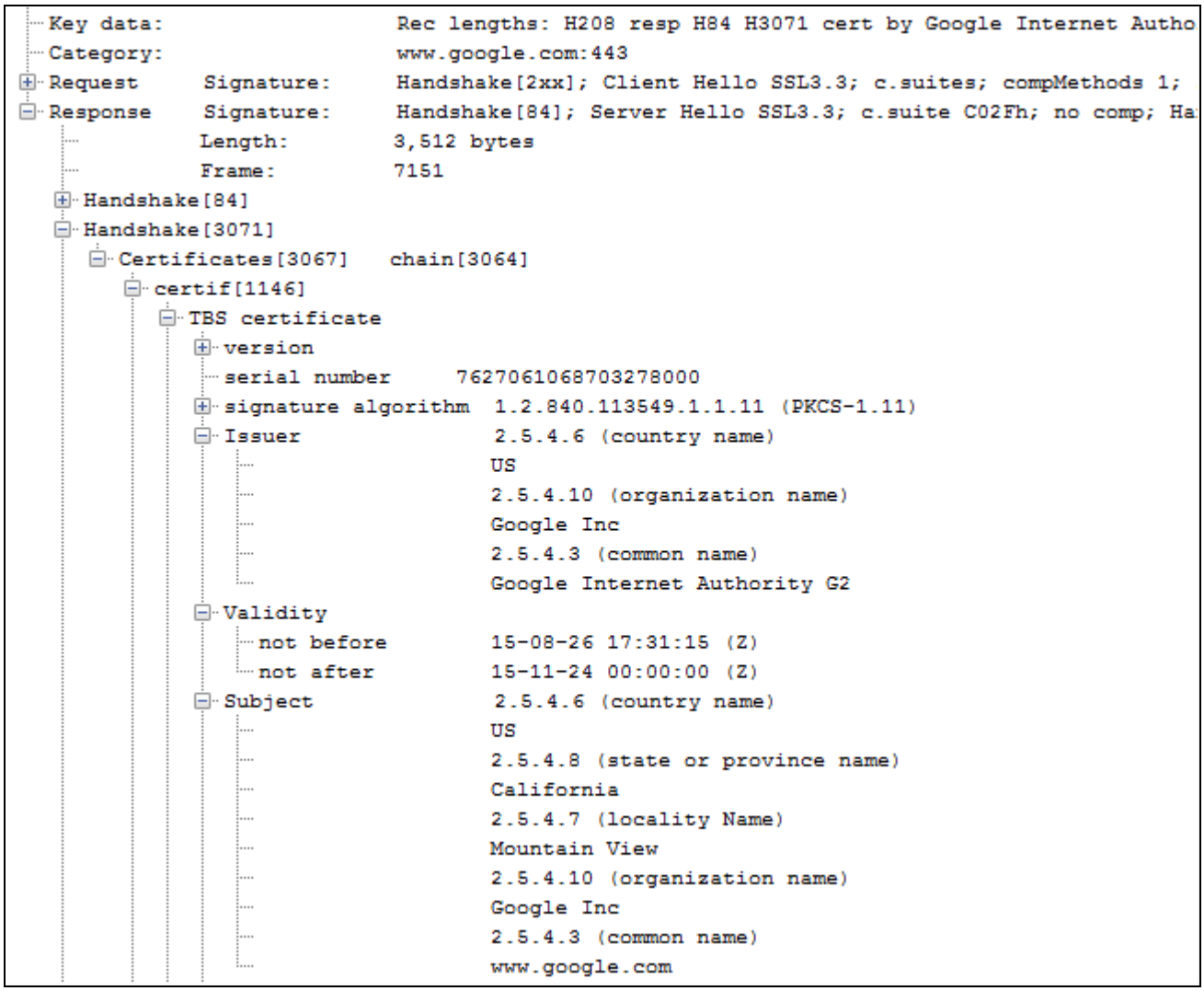
2.3 Summarising Digital Certificate Chains in SSL Handshakes

During analysis NetData summarises the content of all SSL-TLS transactions by listing the types and lengths of the SSL records in the messages, and this information appears in the Data column of the transaction table and in chart pop-ups. A full description of an un-encrypted handshake transaction presents the details of all the records including details of any digital certificates. It is often useful to see who issues the certificates and check the range of dates for which the certificates are valid.

During analysis NetData extracts from digital certificates the names of the issuer and subject, and the range of valid dates. This information is inserted in the list of record types and lengths, as in

```
Rec lengths: H208 resp H84 H3071 cert by Google Internet Authority
G2 for www.google.com (15-08-26 to 15-11-24); cert by GeoTrust
Global CA for Google Internet Authority G2 (13-04-05 to 16-12-31);
cert by Equifax for GeoTrust Global CA (02-05-21 to 18-08-21) H333
H4
```

The panel below displays part of the first certificate in the corresponding chain of three certificates:



When a server consistently uses an invalid certificate the new facility makes it easier to see how clients handle the problem, such as occasionally reverting to an older SSL version and re-using old session IDs.

2.4 Digital Certificate Validity

When decoding SSL handshakes during analysis NetData summarises chains of digital certificates by extracting from each certificate the issuer's organisation name, the subject's common name and the two dates defining a period of validity. This information appears in the data column of the transaction table, as in this summary of a transaction's SSL record types (H for Handshake) and lengths:

```
Rec lengths: H213 resp H511 H3275
    cert by Google Internet Authority G2 for *.google-analytics.com
          (17-01-18 to 17-04-12);
    cert by GeoTrust Global CA for Google Internet Authority G2
          (15-04-01 to 17-12-31);
    cert by Equifax for GeoTrust Global CA (02-05-21 to 18-08-21)
  H300 H4
```

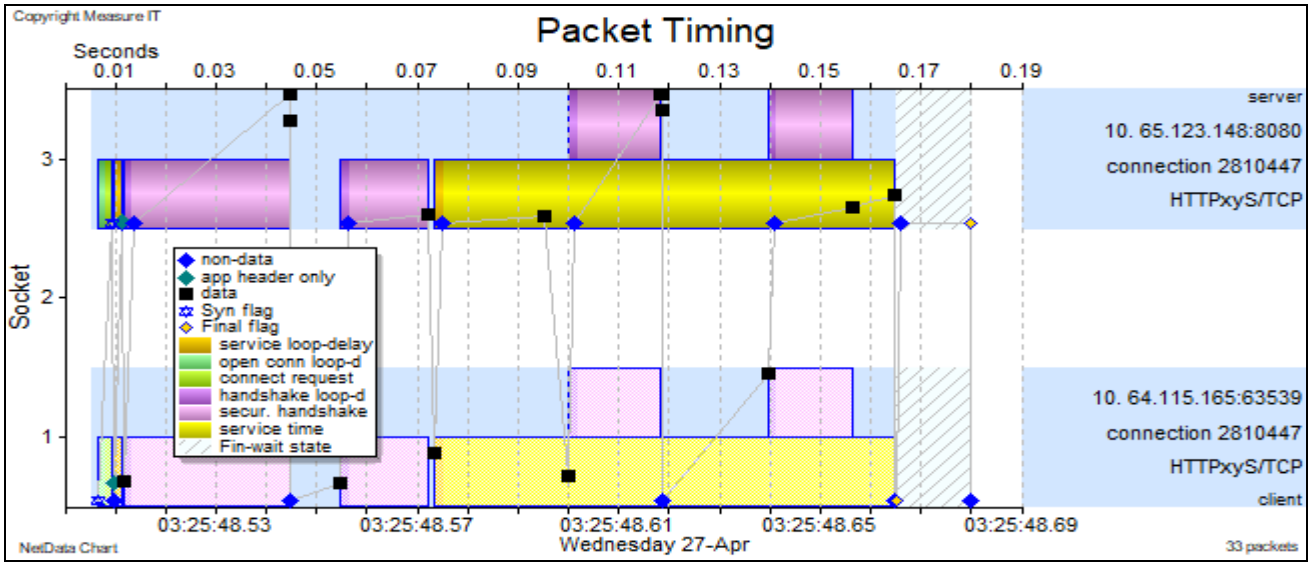
NetData compares each period of validity with the packet's timestamp, and, if a certificate is out-of-date, records a network event categorised as an application error, with a description such as:

```
Digital certificate may be out of date:
    29/01/14 13:04:28 to 29/01/17 13:14:28
    by NSW-DEC-ISS-CA1 for portalsrvs.det.nsw.edu.au
    at 30/01/17 21:06:48
```

2.5 Displays of SSL Renegotiation (2022)

Some Internet connections require SSL (Secure Sockets Layer) renegotiation, a process that allows the details of an SSL or TLS (Transport Layer Security) handshake to be changed after a connection and secure session has been made with the server. Often the second handshake requires the client to send its digital certificates to the server and the second pair of handshake round-trips begin only after the client has sent its first application request to the server; the handshake must be completed before the server can send its application response.

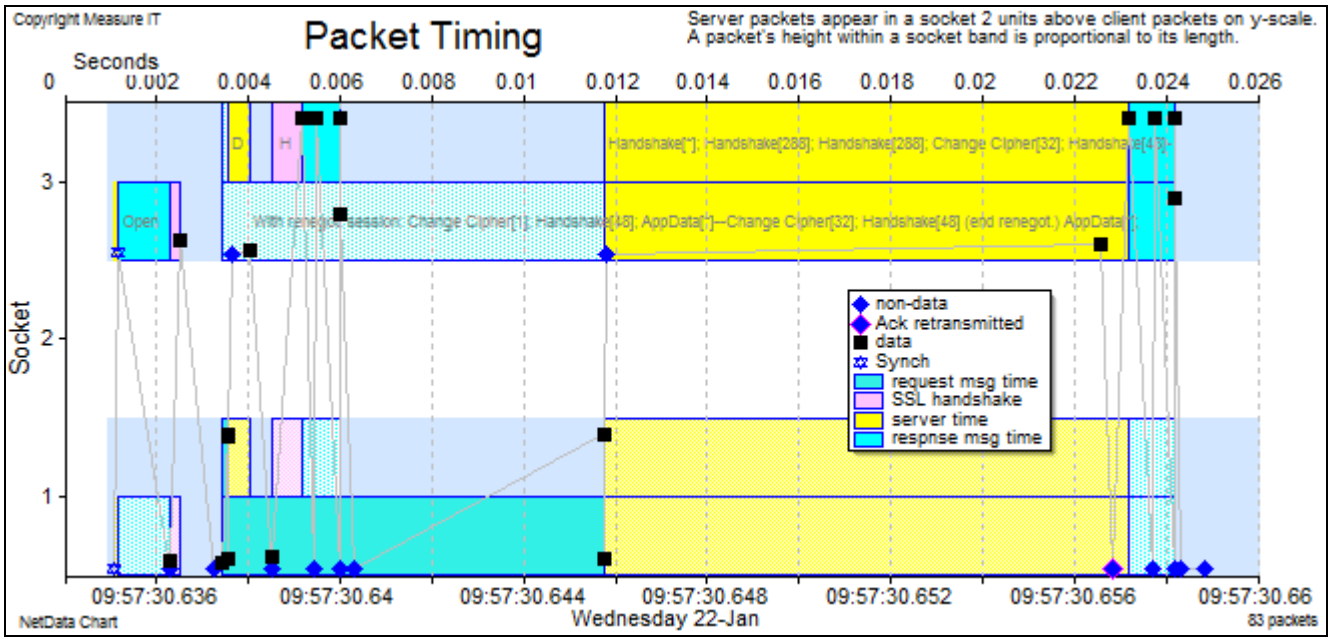
The timing chart now clearly identifies all the handshake round-trips (with a purple colour) and separates them from the data round-trip (in yellow) that spans the time from the application request to its response, as in the chart below that illustrates an HTTP proxy connection that executed only one application transaction but had to wait for four handshake trips:

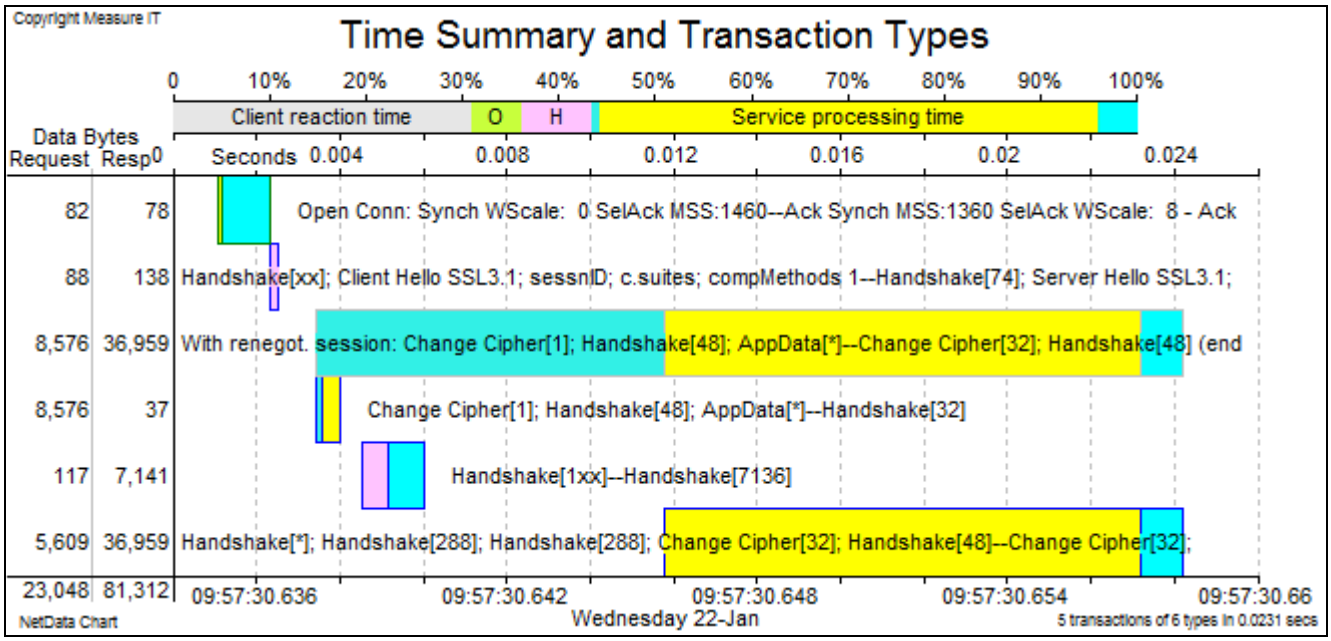


Renegotiation creates a man-in-the-middle security hole and has been disabled in TLS 1.3.

2.6 Secure Session Renegotiation in SSL (2015)

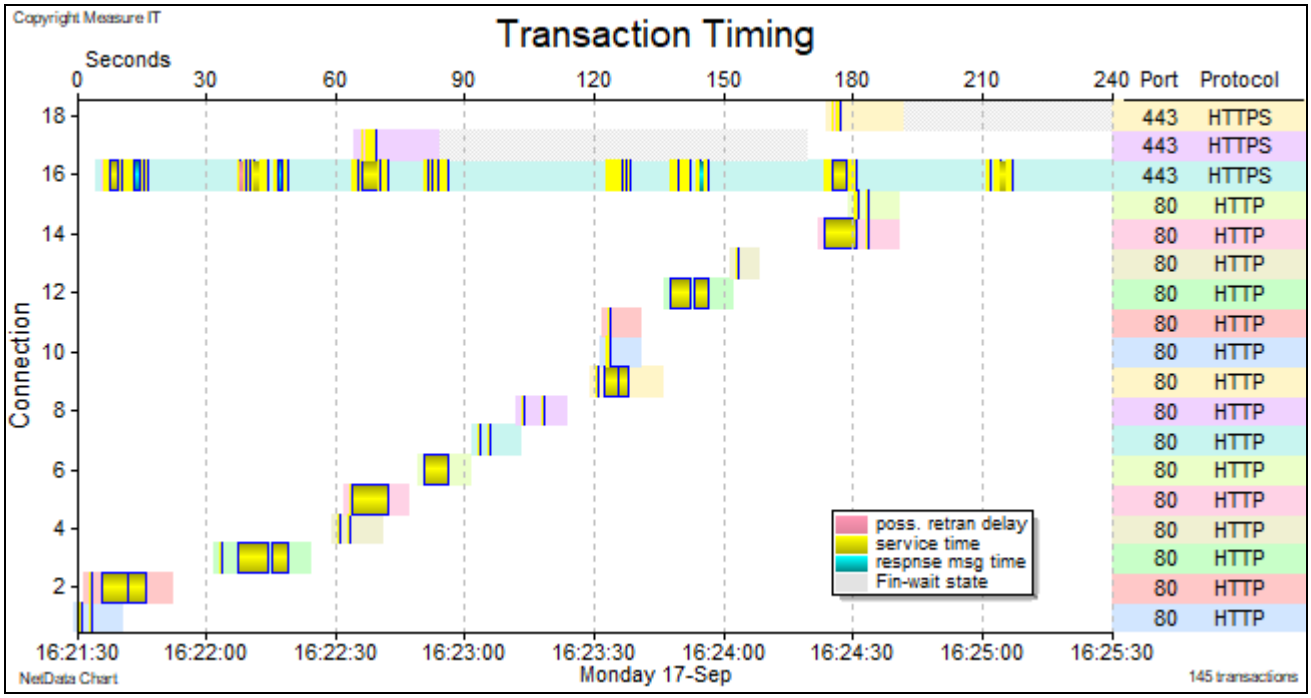
The following two charts show two views of the same HTTPS transaction which is typical of those involving a session renegotiation. The connection began with a quick handshake to re-use the credentials of an existing session. After the client sent the application request (in an AppData record) the server initiated a renegotiation with mutual authentication, by exchanging their chains of certificates. At the end of these handshakes the server sent its application response to conclude the transaction. Besides characterising all the individual round-trips, NetData also records the time over the three round-trips, from the application request to the application response, in a *group* transaction whose description begins with the phrase 'With renegot. session". On the charts the group transaction appears to run concurrently with the three individual round-trips.



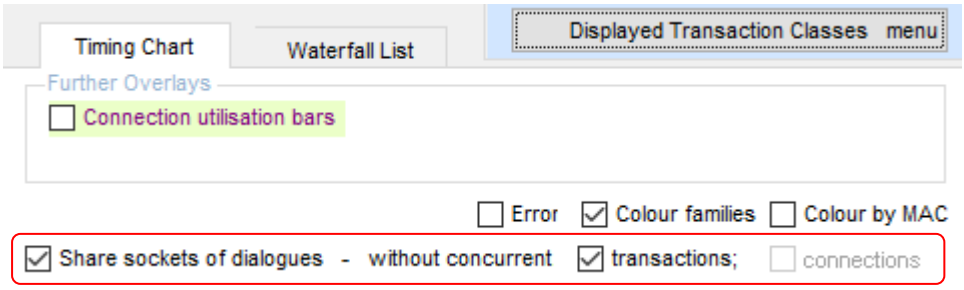


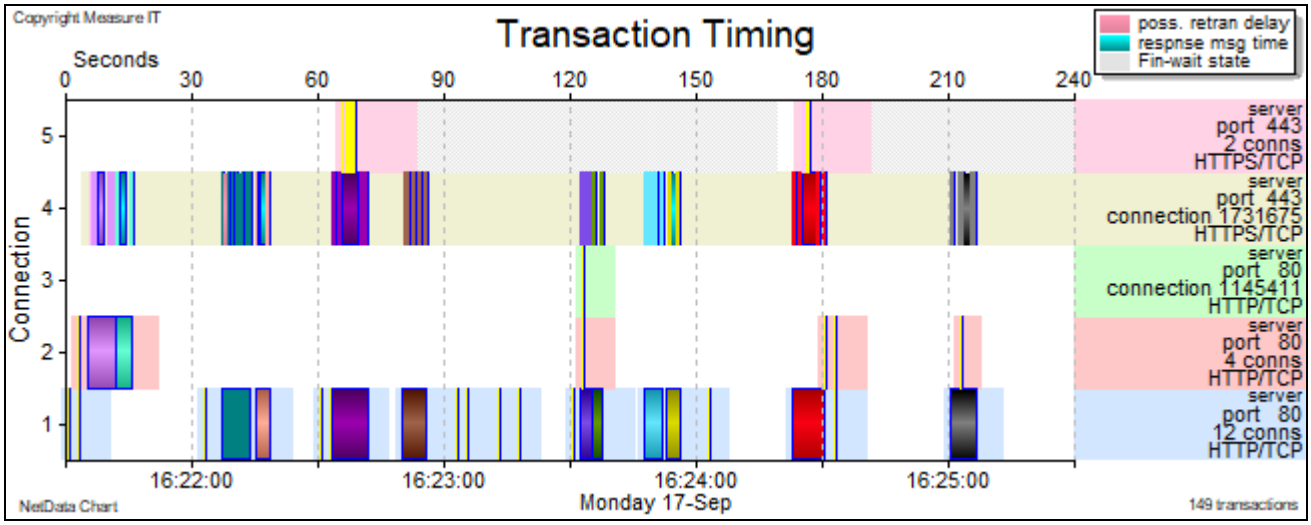
2.7 *Recognising* 100-Continue *Round-Trips in HTTPS Transactions*

Several customising controls of the timing chart – namely, to share connection bands on a transaction timing chart and display partial transactions on a waterfall chart – help to find and illustrate an often-severe performance degradation in HTTP transactions: the addition of unnecessary network round-trips to check the acceptability of a Post request's headers before sending a Post message body. In the following case study most of the front-end HTTP transactions use different TCP connections, whereas the backend HTTPS transactions use a single persistent connection:

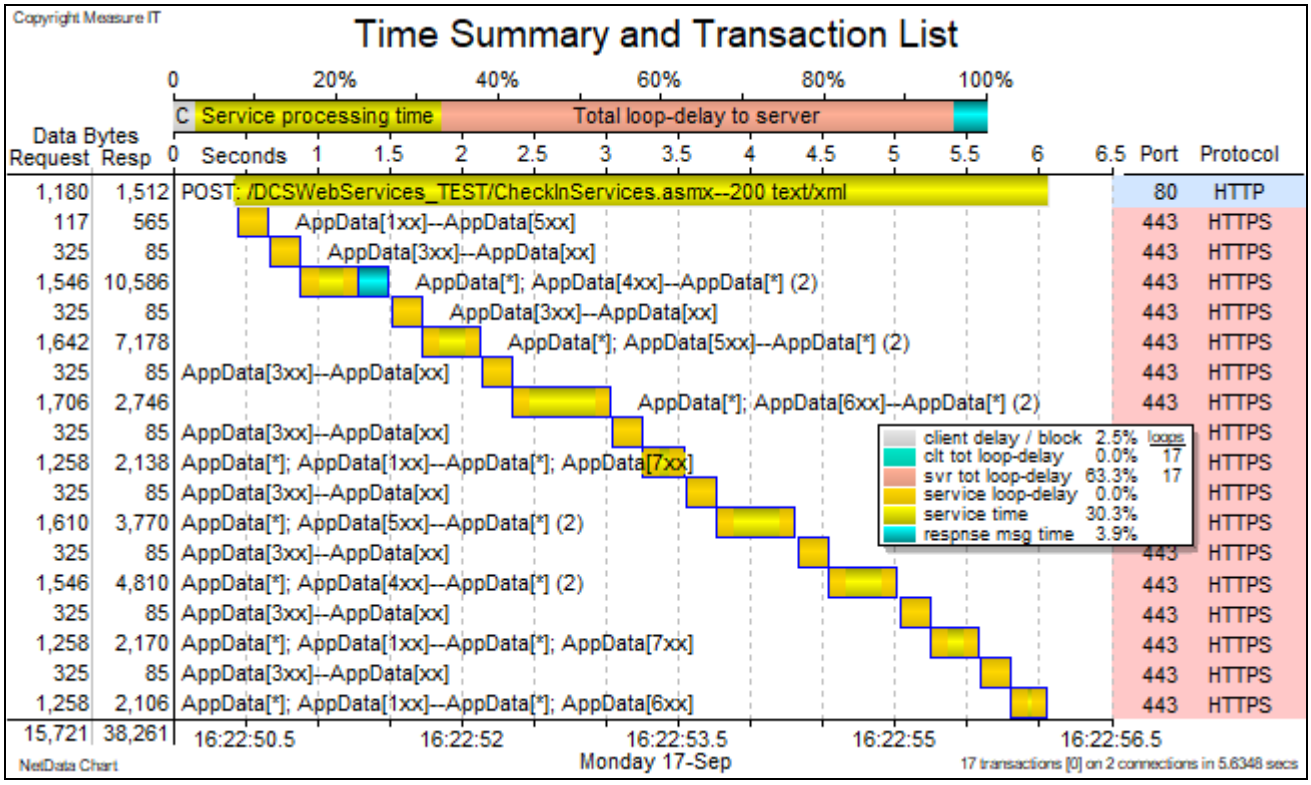


There are two charting techniques that help in identifying the groups of backend transactions that are generated by front-end transactions: one is to enable vertical scrolling and highlight the active backend connection which fixes it on the bottom of the chart while the front-end connections are scrolled past it; the other is to have the front-end connections share a small number of connection bands on the chart, with the 'Share sockets' control:

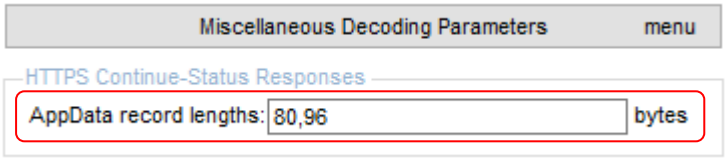




For the above chart families of backend transactions with their front-end parents were identified by NetData's family-finding tool; the transactions in different families were given different colours automatically when transactions were loaded; and sockets were shared to reduce the number of connection bands on the chart. The brown family of transactions is characterised in the following waterfall chart:

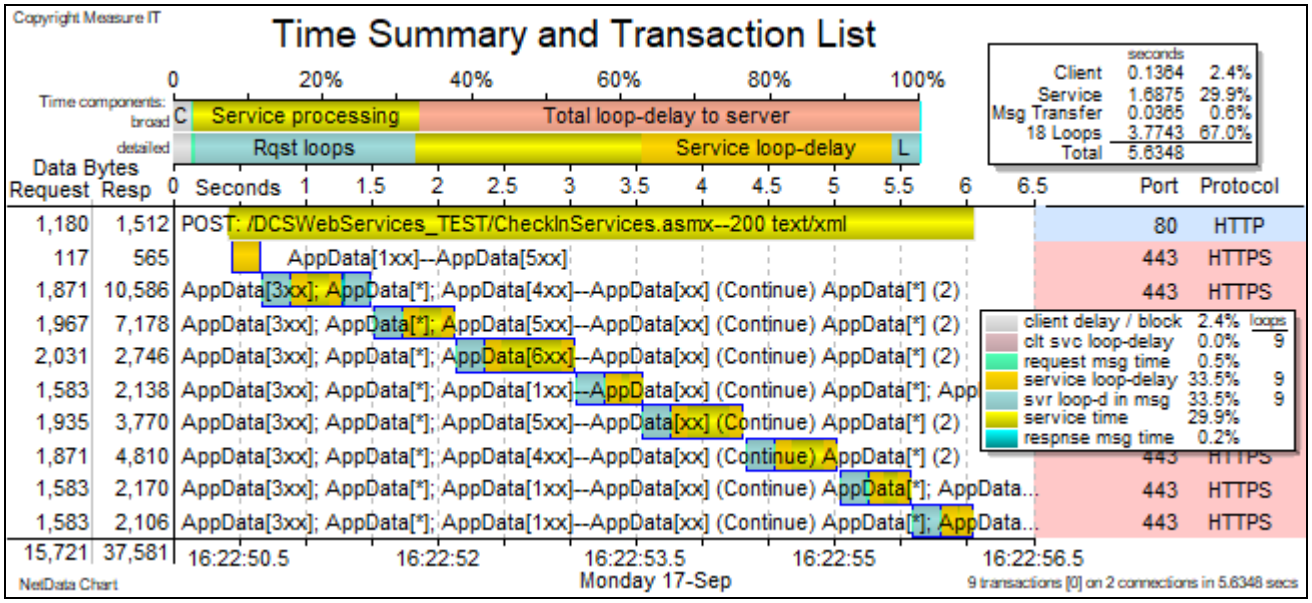


The front-end Post is at the top of the chart and appears to have generated 17 backend HTTPS transactions. However, every second backend transaction has the same type, with a request length of 325 bytes and a response of 85 bytes, and this is a strong indication that Post headers have requested a 100-Continue status; the client has waited for such a response before proceeding to send the Post body.

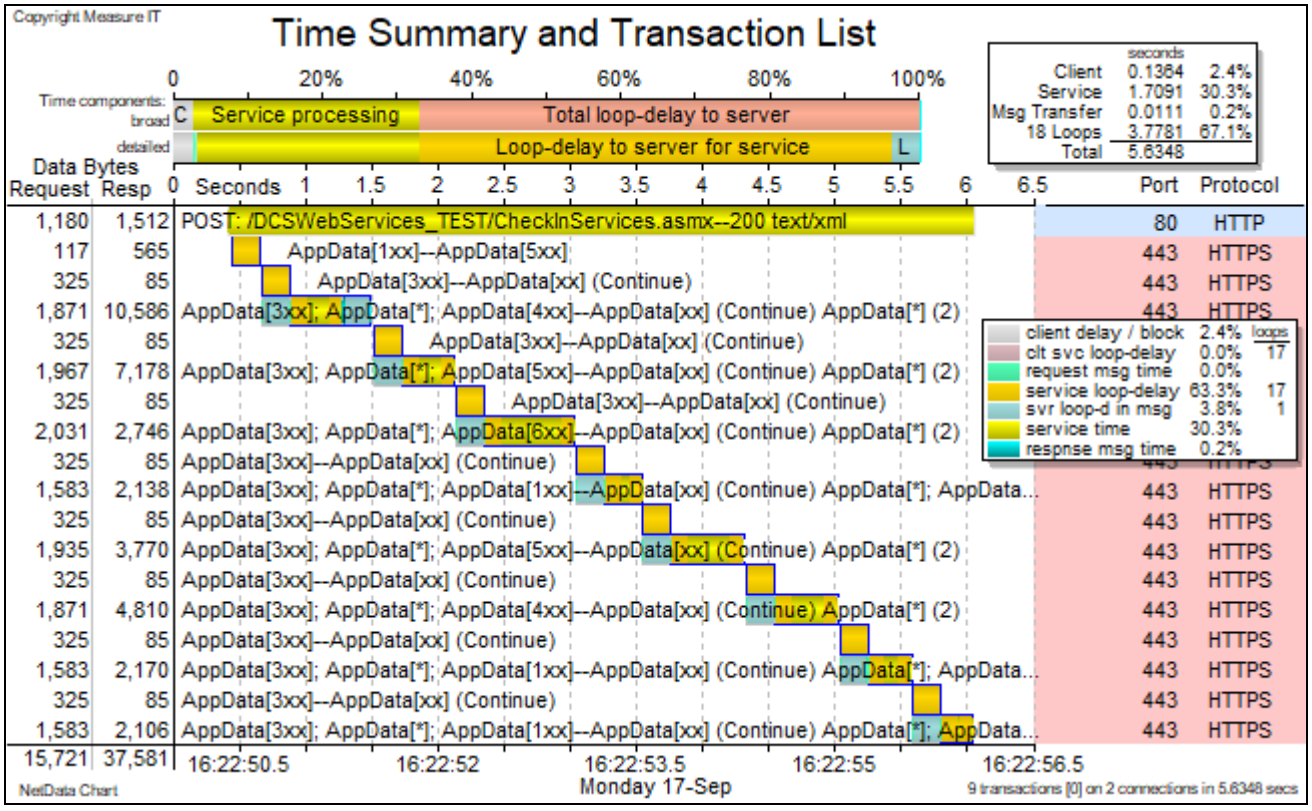


One of the miscellaneous decoding controls (above) specifies the lengths of SSL application-data records that contain 100-Continue messages – 80 bytes in this case and 96 bytes for another server. The SSL record is 5 bytes shorter than the overall message because each record has a 5-byte header.

After repeating the analysis NetData has treated the initial round-trip of each Post as part of the time for its request message and counted it as a TCP message-transfer loop. Packet records were loaded for the next charts to allow the charting module to count TCP round-trips during message transfers and paint their propagation delays a different colour:



The initial round-trips have also been characterised as transactions of the *Partial* class. If loaded and displayed on the waterfall chart they emphasise their severe effect on response times:

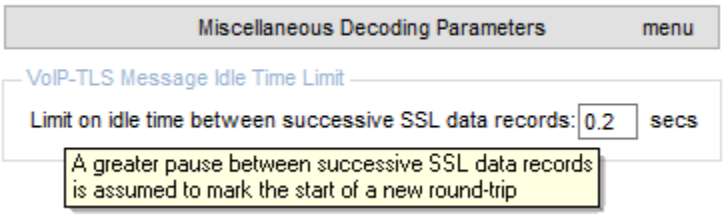


The simple solution to this performance problem is to remove from Post requests the Expect header

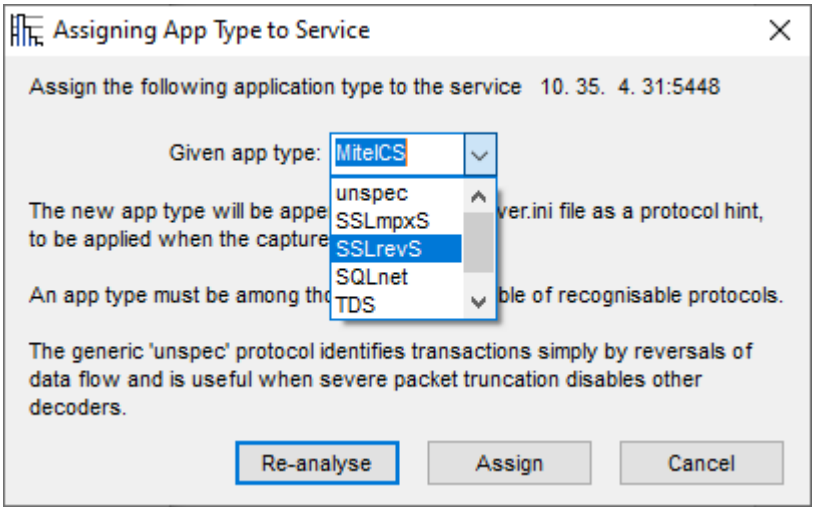
Expect = “100-continue”

2.8 TLS Transactions Initiated by Either Client or Server

Many protocols such as SIP and the Mitel call-control protocol allow either the client or server to initiate transactions. NetData can normally distinguish between request and response by message contents but when the traffic is encrypted with TLS/SSL NetData must rely on idle-time measurements to judge when one transaction has terminated and a new request has been initiated. In other words, if a pause between successive SSL data records exceeds the specified limit, NetData assumes that one message has ended and a new message has started. The default limit is 0.2 seconds and it can be changed by a control in the list of miscellaneous decoding parameters:



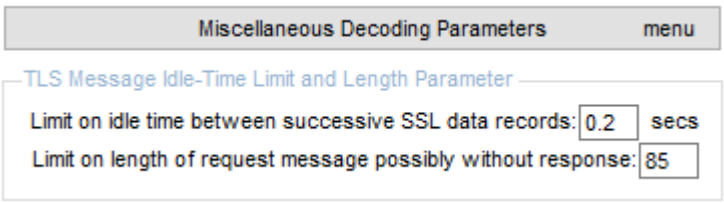
If an unknown TLS/SSL protocol appears to initiate round-trips from both the client and server at different times, the application protocol tagged as ‘SSLrevS (for Reversing SSL) should be assigned to the service. This can be done by right-clicking the service on the dialogue chart, choosing to ‘Assign App Type to Service’, and choosing the new app type from the drop-down list:

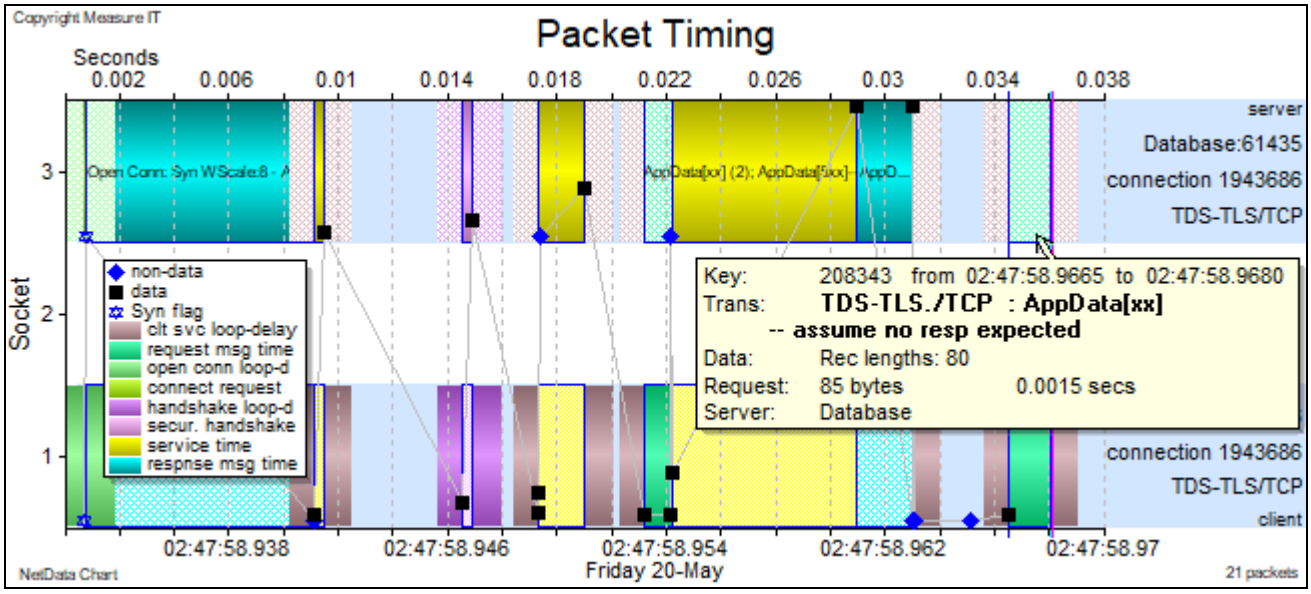


2.9 TLS Client Messages without Response

Some protocols that use TLS sometimes send a short client message that has a supervisory function and does not solicit a response. If that message is considered part of the subsequent transaction, the transaction will record a false message-transfer time and may dominate a chart with an unrealistic response time.

A control in the menu of miscellaneous decoding parameters on the Decoding page can specify a limit on the length of a TLS request message that may not expect a response. An associated control specifies an idle-time limit, and if a TLS connection is idle for a longer time, after transmitting a client message no longer than the specified length, the client message is recorded as a request without response.





2.10 Displaying HTTP Chunk Contents

Some web applications use their socket interface incorrectly. When building a web page individual elements are passed through the socket with an indication that causes the element to be processed immediately, without waiting for a packet to be filled. Each element, no matter how small, is sent as a separate chunk, and, if the network is fast, each chunk is sent in its own packet. When a burst of packets is sent to fill a transmit window the resulting high packet rate can stress a router and cause packets to be dropped; the retransmission timeouts produce large response times. If the traffic is encrypted each chunk is sent in its own SSL record, and the high processing overhead for each record can stress both sender and receiver.

To help reveal this behaviour when the traffic is not encrypted, the HTTP decoder puts chunk-length indicators (specified in hex characters) in the Length-indicator column of the packet table, and, if verbose output is selected on the Output page of controls, displays the data contents of each packet in the Contents column. The packets table browser then reveals how the application breaks a web page into chunks.

2.11 Quick UDP Internet Connection (QUIC) Versions Q043, Q044

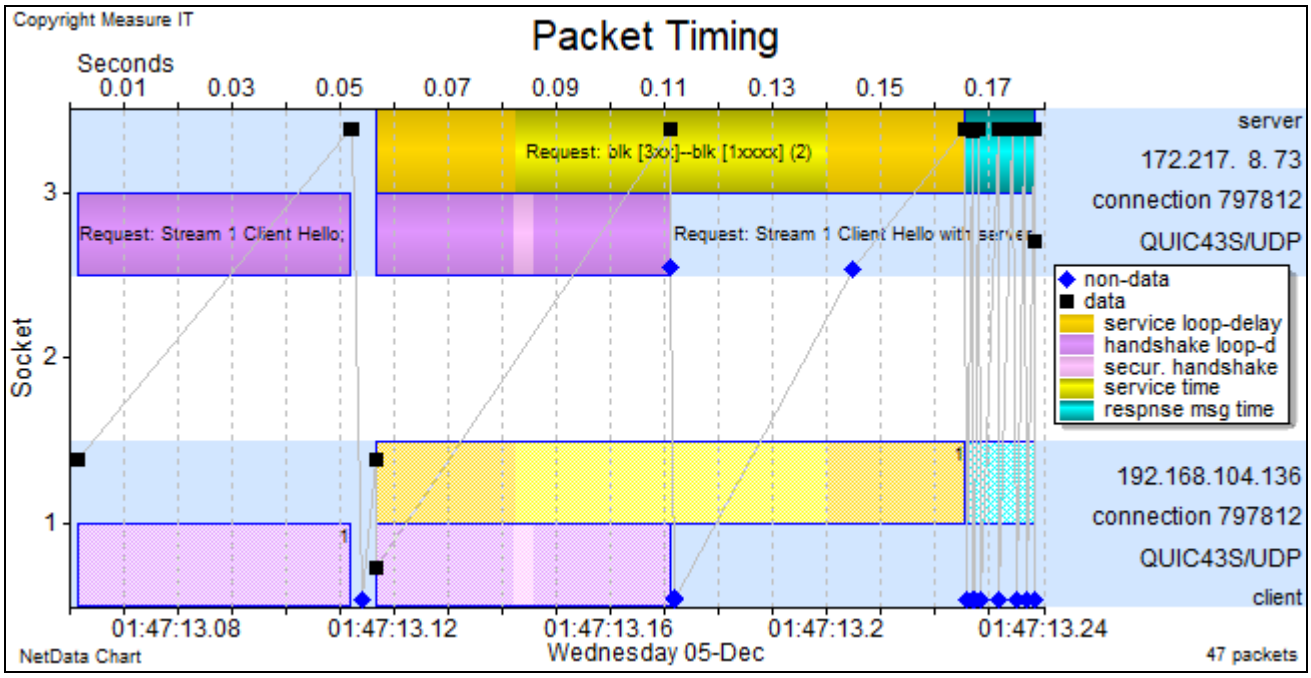
Since 2012 Google has been experimenting with a better-performing connection-oriented transport-layer protocol for web access, a replacement for TCP. The protocol is known as QUIC, for Quick UDP Internet Connection, and minimises the number of round-trips needed to establish connections and secure sessions. By building on UDP it is able to move control of congestion avoidance from the operating system kernel to application space, and it is designed to evolve with a forward error correction (FEC) scheme and a multiplexing scheme that supports independently flow-controlled data streams. NetData has been tested so far only with the widely-used version Q043, not yet with the latest version Q044 that introduces long and short public (non-encrypted) headers and three different packet-number spaces for different phases of a connection's life. Q044 is destined to become the basis for HTTP 3.

Packet sequence numbers provide valuable diagnostic information and NetData is able to identify sequence gaps and overtaken packets on both packet-timing and flow charts, even though only the low-order bits of the 64-bit numbers are transmitted in the packets. NetData records the true sequence numbers in the data-sequence column of the packet table. The transmitted number, prefixed with 'p', appears in the KeyData column, together with QUIC version numbers when transmitted. Connection IDs from the public headers appear in the Context column.

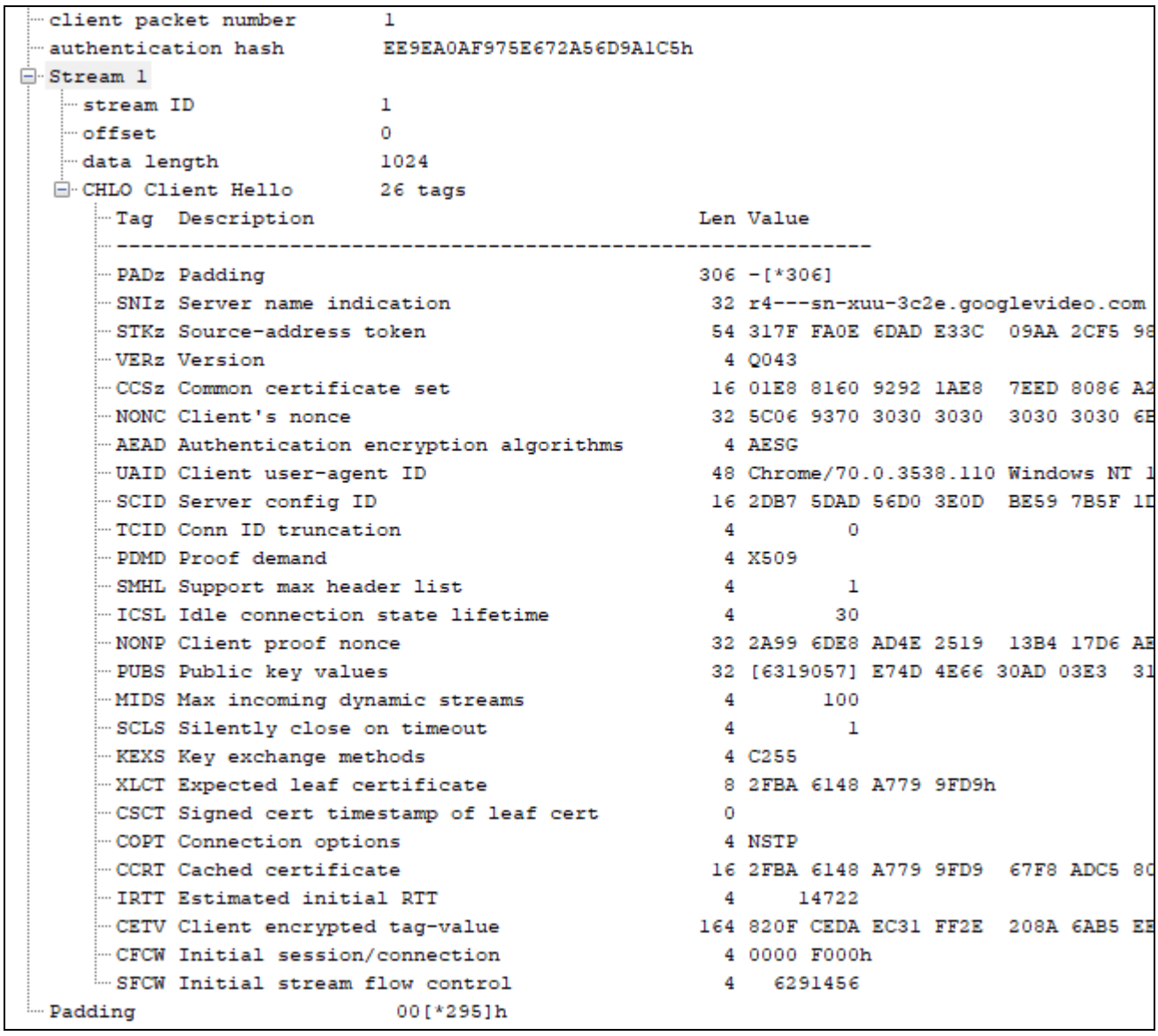
2.11.1 One Extra Round-Trip Time (1-RTT)

A connection begins with the client sending a Client Hello in a special *Stream* frame comprising a list of tags that NetData displays in a table, as below.

For security reasons these handshake frames are always padded out to the maximum segment size. For the strongest security the server responds to the Client Hello by sending a nonce (security token) in a Stream frame with a Rejection tag, and the client then resends its Client Hello with the nonce included.



Although there are two handshake round-trips as in SSL, application requests can be issued before the second handshake response (as above) and the result is described as a 1-RTT connection because connection setup incurs an extra delay of only one round-trip time.



```
client packet number        1
authentication hash         EE9EA0AF975E672A56D9A1C5h
Stream 1
  stream ID                 1
  offset                    0
  data length               1024
  CHLO Client Hello         26 tags
    Tag  Description                                  Len Value
    ---------------------------------------------------------
    PADz Padding                                     306 -[*306]
    SNIz Server name indication                       32 r4---sn-xuu-3c2e.googlevideo.com
    STKz Source-address token                         54 317F FA0E 6DAD E33C  09AA 2CF5 98
    VERz Version                                       4 Q043
    CCSz Common certificate set                       16 01E8 8160 9292 1AE8  7EED 8086 A2
    NONC Client's nonce                               32 5C06 9370 3030 3030  3030 3030 6E
    AEAD Authentication encryption algorithms          4 AESG
    UAID Client user-agent ID                         48 Chrome/70.0.3538.110 Windows NT 1
    SCID Server config ID                             16 2DB7 5DAD 56D0 3E0D  BE59 7B5F 1D
    TCID Conn ID truncation                            4          0
    PDMD Proof demand                                  4 X509
    SMHL Support max header list                       4          1
    ICSL Idle connection state lifetime                4         30
    NONP Client proof nonce                           32 2A99 6DE8 AD4E 2519  13B4 17D6 AE
    PUBS Public key values                            32 [6319057] E74D 4E66 30AD 03E3  31
    MIDS Max incoming dynamic streams                  4        100
    SCLS Silently close on timeout                     4          1
    KEXS Key exchange methods                          4 C255
    XLCT Expected leaf certificate                     8 2FBA 6148 A779 9FD9h
    CSCT Signed cert timestamp of leaf cert            0
    COPT Connection options                            4 NSTP
    CCRT Cached certificate                           16 2FBA 6148 A779 9FD9  67F8 ADC5 80
    IRTT Estimated initial RTT                         4     14722
    CETV Client encrypted tag-value                  164 820F CEDA EC31 FF2E  208A 6AB5 EE
    CFCW Initial session/connection                    4 0000 F000h
    SFCW Initial stream flow control                   4   6291456
Padding                      00[*295]h
```

A Client Hello packet

```
server packet number       4
authentication hash        E657D2382C868A716320805Ch
Stream 1
  stream ID                1
  offset                   0
  REJz Rejection           8 tags
    Tag  Description                                  Len Value
    ------------------------------------------------------------
    STKz Source-address token                         56 4635 D48D F049 2768  02D3 F53E E9
    SNOz Server's nonce                               52 2A0F 1B89 74ED 3002  F3E0 E465 EE
    PROF Proof (signature)                           256 2A74 AF86 9A9A 00C3  7B1E FB7C 62
    SCFG Server config                               147 7 tags
      Tag  Description                                 Len Value
      ------------------------------------------------------------
      AEAD Authentication encryption algorithms         8 AESGCC20
      SCID Server config ID                            16 2DB7 5DAD 56D0 3E0D  BE59 7B5F
      PDMD Proof demand                                 4 CHID
      PUBS Public key values                           35 [32] 9111 C41A C5E3 ACE6  89C6
      KEXS Key exchange methods                         4 C255
      OBIT Server orbit                                 8  00000000
      EXPY Expiry                                       8 1544097600
    RREJ Reasons for server sending                     4 The expected leaf certificate has
    STTL Server config TTL                              8    162768
    CSCT Signed cert timestamp of leaf cert           241 00EF 0076 00A4 B909  90B4 1858 14
    CRT# Certificate chain                            945 [short by 445] 0102 67F8 ADC5 8015
```

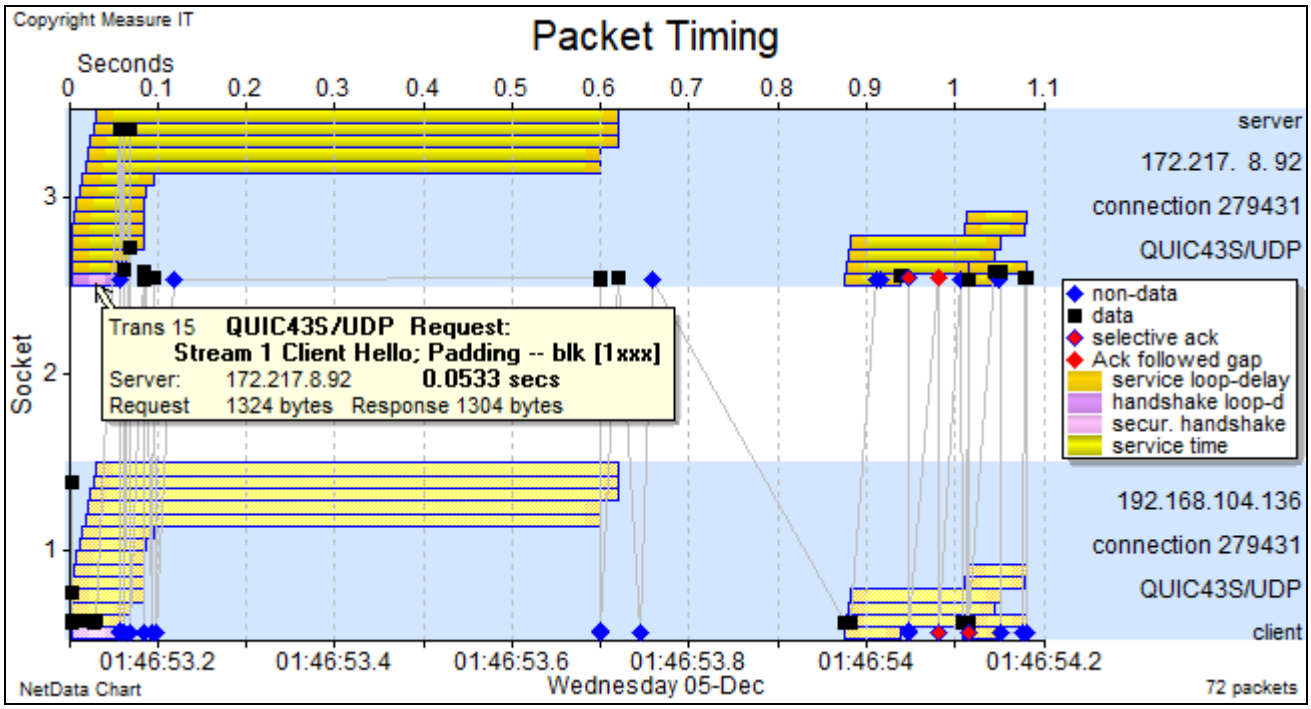
The Rejection tag ends with a certificate chain that doesn't fit in a single packet (here it is short by 445 bytes) and must be continued in subsequent packets which NetData recognises as non-encrypted:

```
server packet number       5
authentication hash        D07474FE7C656E2D0EC5EB3Bh
Stream 1
  stream ID                1
  offset                   1326
  data length              455
  tag continuation         7E98 4555 9905 7AC0  44D6 C405 1428 2B31  044B 8A6B E9E5 A7A5
Padding                     00[*866]h
```

The continued Stream frame (offset 1326) is followed in this packet by a *Regular* Padding frame to ensure that the packet has the standard full size.

2.11.2 Zero Extra Round-Trip Times (0-RTT)

QUIC can also establish 0-RTT connections that require only one handshake round-trip, and that trip can run concurrently with any number of application requests, as in the chart below. A 0-RTT connection has slightly weaker security in that it is vulnerable to replay attacks, but this weakness is considered quite acceptable for idempotent transactions such as simple Get requests.



2.11.3 Tracking Sequence Gaps

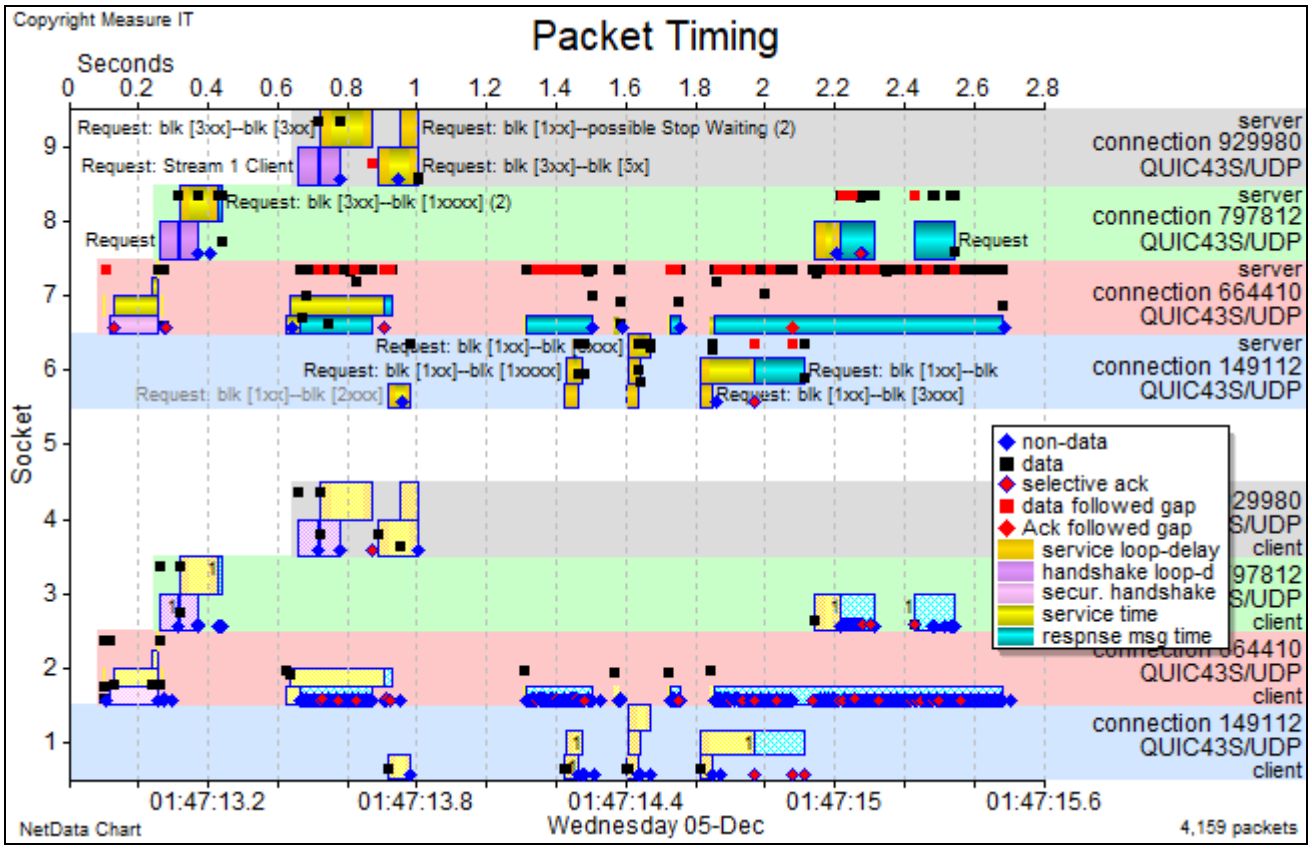
Acknowledgement frames detail all the sequence gaps in received packets, but virtually all these frames are encrypted which means that it is not possible for an observer to determine such vital flow-control information that is equivalent to TCP bytes-in-flight. Nevertheless, NetData judges the presence of ack packets by their short length, and can infer the number of sequence gaps by changes in their lengths. NetData also recognises increments in ack-packet length by 13 bytes and attributes them to the presence of a Window Update frame accompanying the Acknowledgement frame. NetData labels ack packets with one or more sequence gaps as iSACK packets (for inferred selective acknowledgement) and indicates the estimated number of sequence gaps. To facilitate the charting of this information, the number of sequence gaps is also displayed in the Window column of the packet table.

2.11.4 Transaction Characterisation

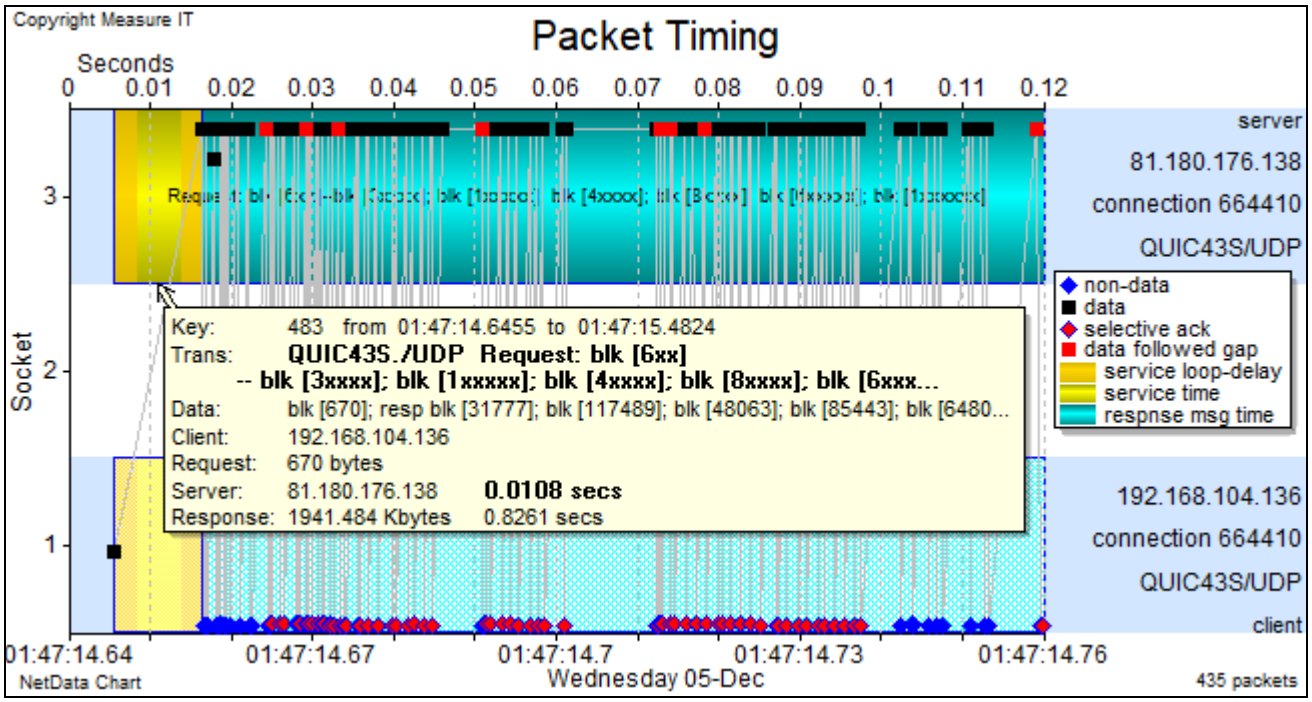
Because NetData is able to separate ack packets from data packets fairly reliably, it is also able to characterise transactions even when many transactions run concurrently on a single connection. NetData assumes that transactions take the form of request-response pairs like most web transactions and, because it impossible to definitively associate responses with their requests, NetData assumes that response messages appear in the same order as their corresponding requests, as in HTTP 1. NetData attempts to recognise the end of application message blocks by the appearance of partially-filled packets or by longer idle periods between packet bursts.

NetData's assumptions are likely to be quite unreliable when packets are lost in the network because packet retransmissions do not have the same packet numbers as their original packets. NetData users should draw their own inferences when viewing all the packet-length and -timing clues presented on the timing and flow charts.

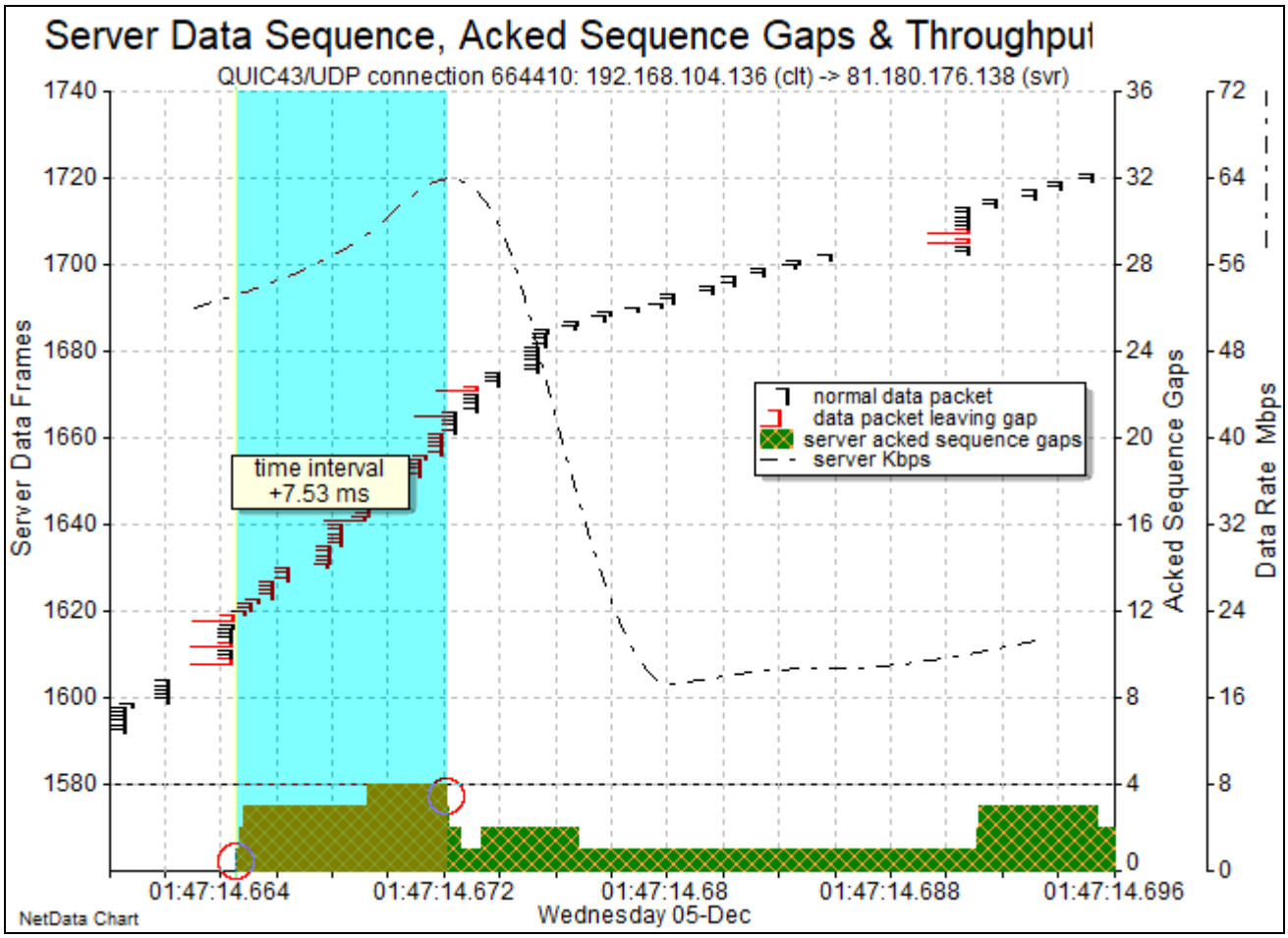
The following charts demonstrate the effectiveness of the transaction characterisation techniques.



As in all packet-timing charts, red square markers indicate packets that follow a sequence gap, and pink diamonds indicate the equivalent of SACK packets – those reporting one or more sequence gaps. Security handshake round-trips have pink bars like SSL handshakes.



This chart shows the packets involved in the first part of an HTTP request and response for a file of 1.96 Mbytes. Sequence gaps were reported by the receiving client almost immediately after it noticed a gap in the packet sequence numbers.



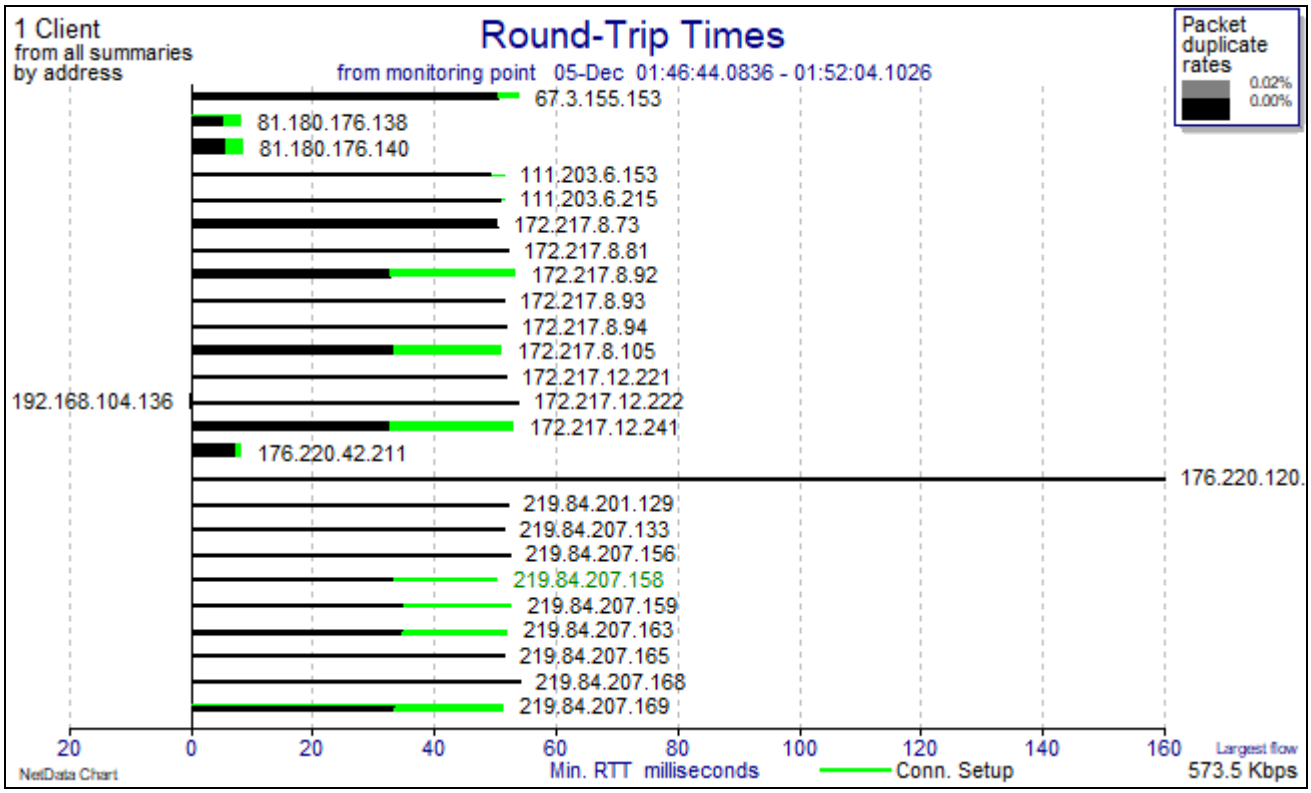
Further flow-control insight is provided by the data sequence chart that plots packet strips at their time of day and against the left-hand scale of packet numbers. An initial short burst of packets left three gaps in the packet numbers and after little more than one round-trip time the flow rate, indicated by the slope to the band of packet strips, was suddenly halved. However, this is believed to be the consequence of a packet shaper in the network, not QUIC flow control.

The area graph at the bottom of the chart displays the number of acknowledged sequence gaps determined by the lengths of ack packets. The blue rectangle measures the time interval (7.5 ms) from the first ack reporting a sequence gap (circled in red) to the first reduction in the number of reported sequence gaps (also circled in red). Because this interval spans at least one round-trip, to report the sequence gap and receive a stop-waiting frame, it provides an estimate of the path's round-trip time. Lost packets are retransmitted with new packet numbers and can't be identified on the chart.

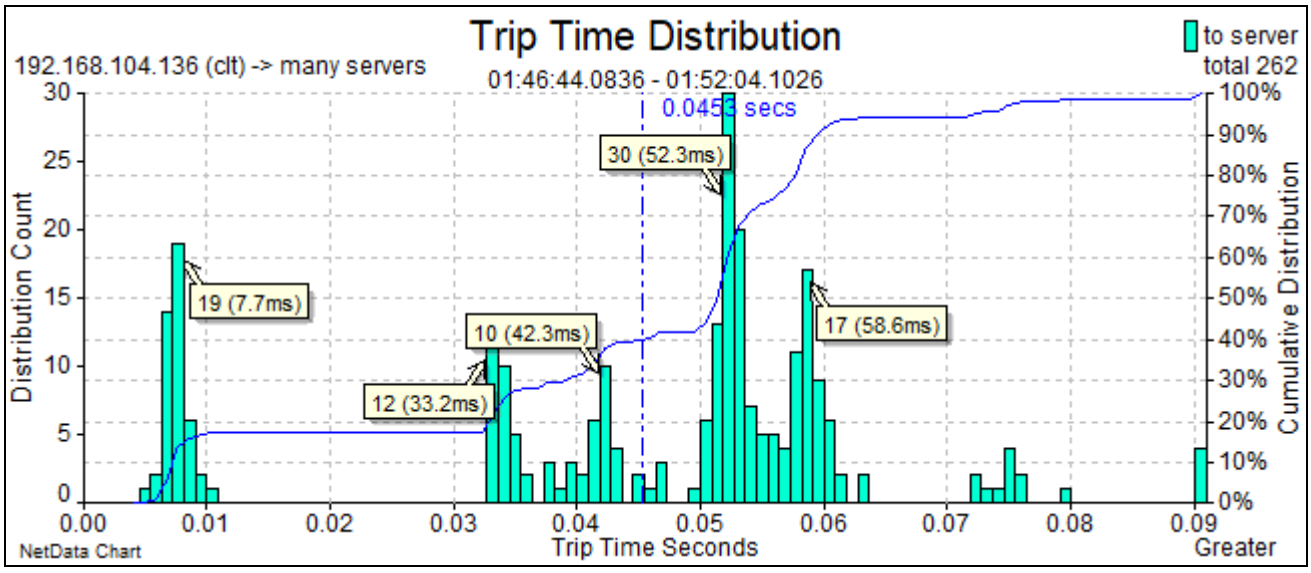
2.11.5 Measuring Round-Trip Times

In the circumstances in which acknowledgements can be reliably associated with their data packets NetData measures their round-trip times and records the minimum values for each connection and each dialogue, from sniffer to server and from sniffer to client. As with TCP round-trip time measurements, these measurements provide useful estimates of path loop-delays (mostly propagation).

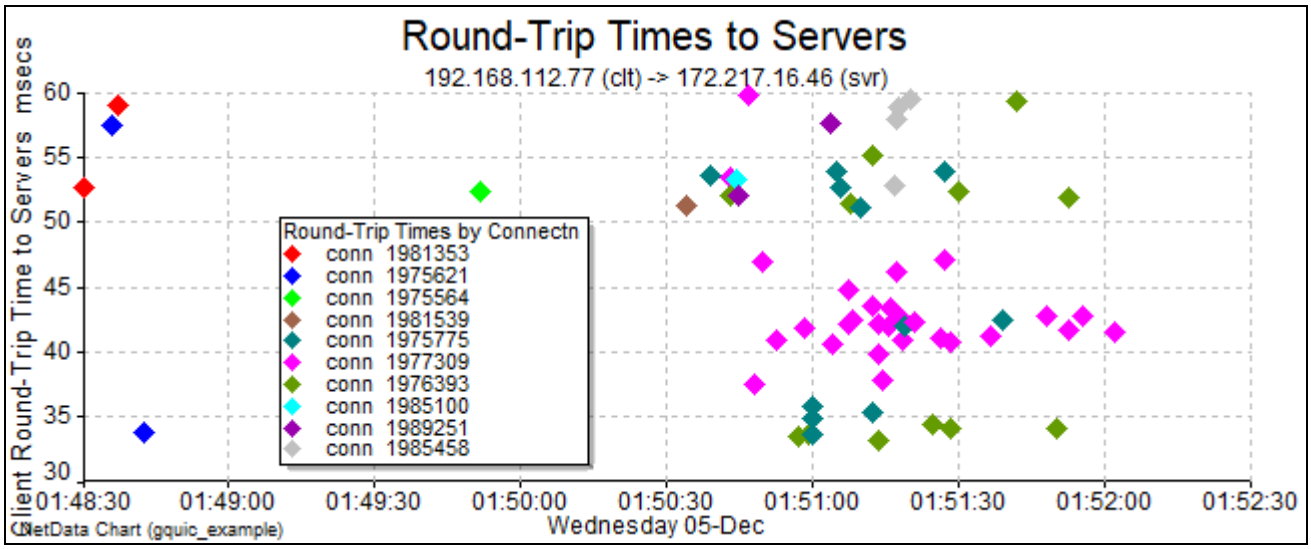
The association of QUIC packets assumes that an immediate acknowledgement is issued only after two data packets are received, and the delayed-ack delay is 25 ms.



This chart of the minimum round-trip times for all the paths handling QUIC traffic shows that the sniffer was located at the client workstation, and the majority of servers were 52 ms (round-trip) from the client. The green bars indicate the difference in the minimum times over all types of round-trips (black) and the times for handshake round-trips.



The more common round-trip times were around 7, 33, 42, 52 and 58 ms.



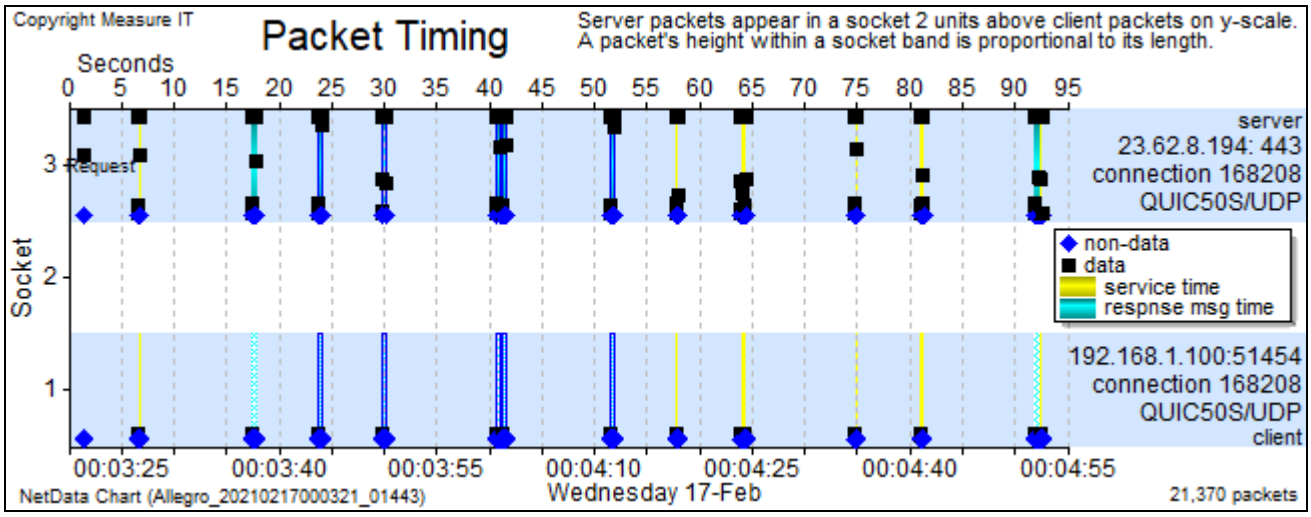
Unlike TCP, the packets of UDP connections are not constrained by network equipment to take the same path. Round-trip times can vary widely even in the absence of significant congestion, when packets take different routes. This chart plots the round-trip times of QUIC packets all addressed to the same YouTube server, in 10 different connections. Most of the round-trip times of packets in the pink connection were clustered around 42 ms, whereas the times of the dark-green connection jumped between 33 and 52 ms.

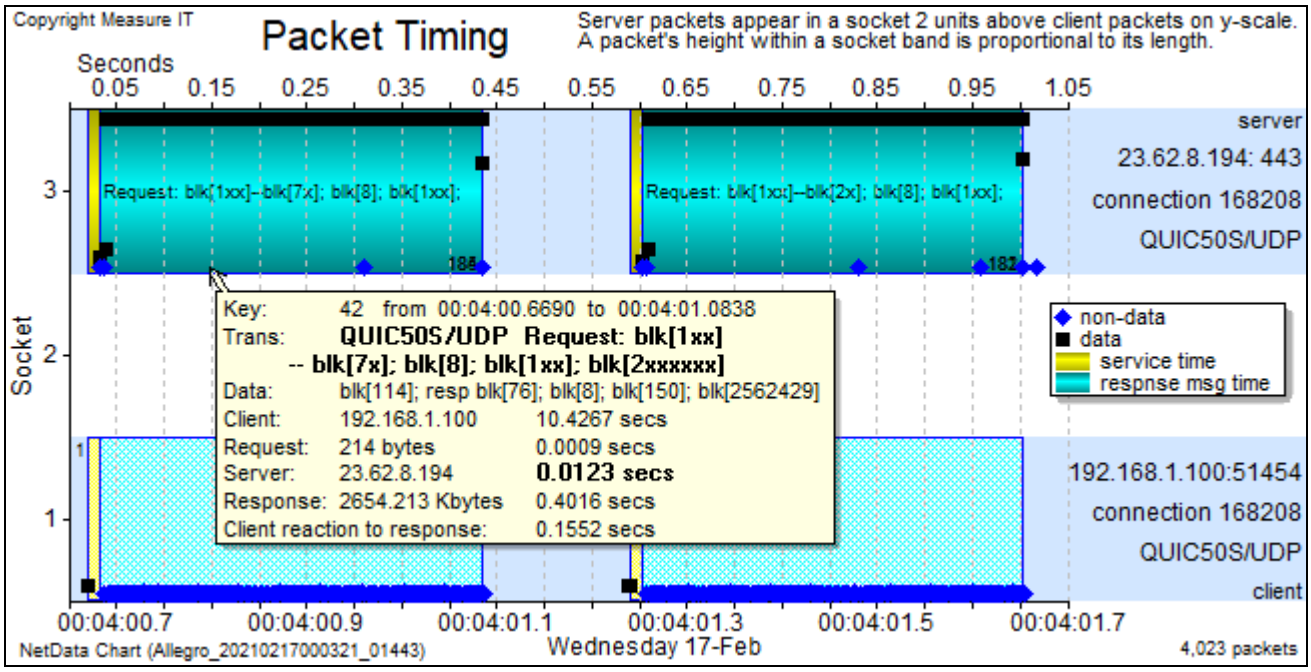
2.12 Google QUIC Q050

Google's latest version of QUIC (Quick UDP Internet Connections) to be found in widespread use is Q050. If NetData sees the long-header packets at the start of a session it will recognise the version; otherwise, it analyses short headers and looks for a common connection ID in successive packets to detect the QUIC protocol and, by assumption, label it as Q050.

In client packets, all but the first two bits in the first byte, and the connection ID in the next 8 bytes, are 'protected' (i.e. encrypted) which effectively hides packet numbers, acknowledgements and flow-control information as well as the application payload. Nevertheless, NetData labels short packets as non-data packets and characterises transactions according to reversals of data flow. When the application protocol is non-multiplexed HTTP this characterisation provides an effective indication of transaction response times on the performance and timing charts.

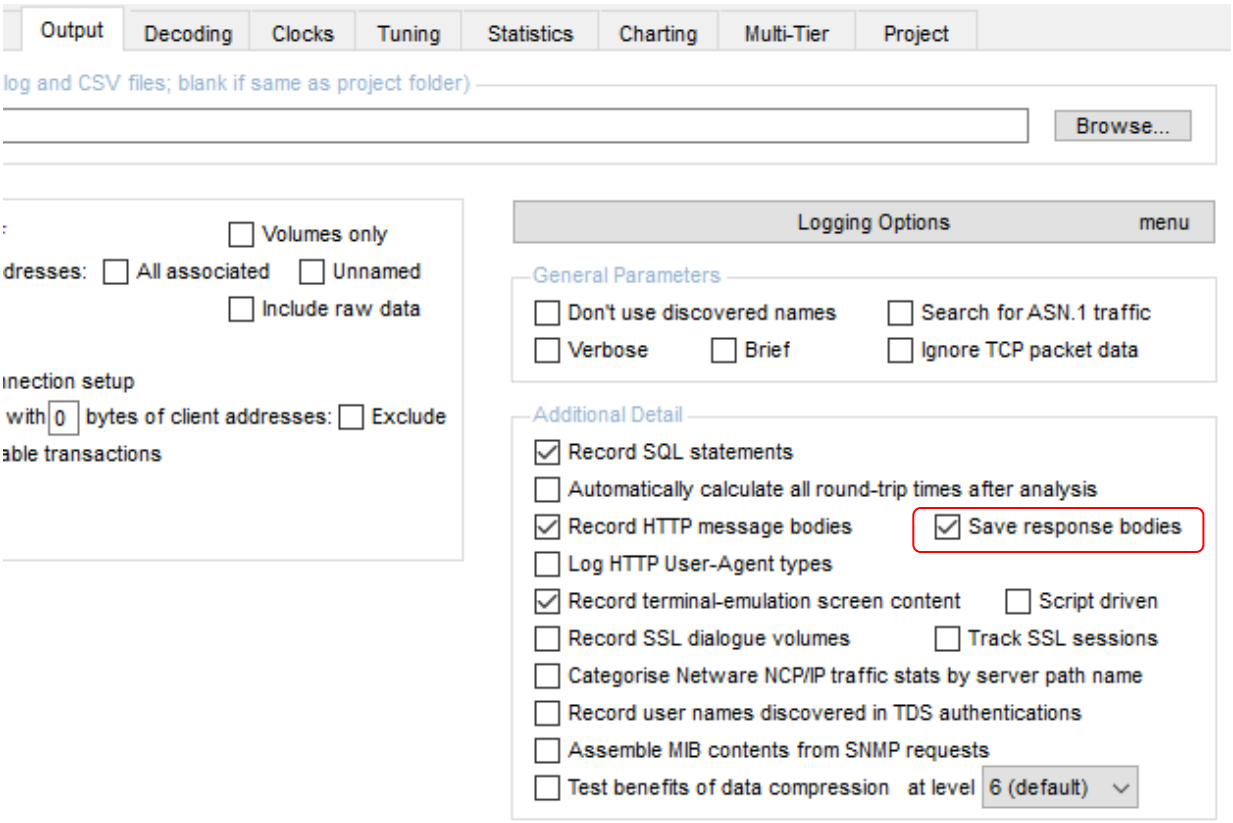
The following charts were drawn from a live TV program streamed from a local Akamai edge server:



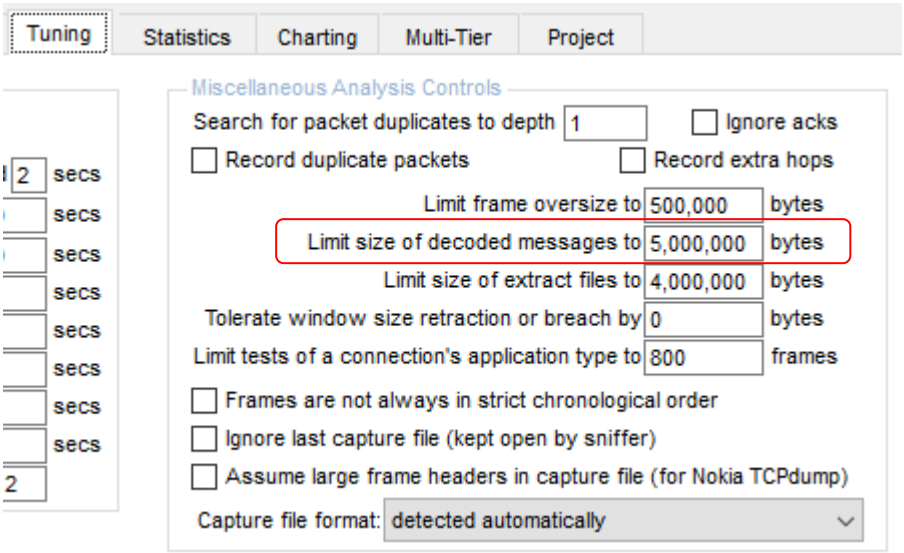


2.13 Extracting Files Downloaded by HTTP

A new checkbox on the Output page of controls causes NetData during analysis to write the contents of all HTTP response-message bodies in files in the project folder. The file names are derived from a relevant response header. Any existing files of the same name are overwritten.

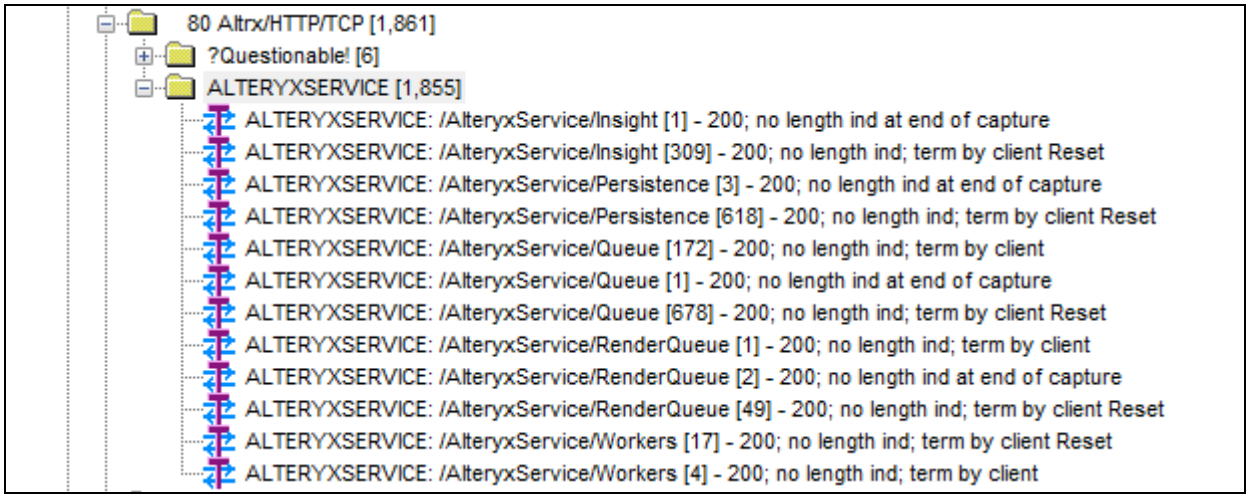


For efficiency NetData doesn't record network messages larger than 5 Mbytes, by default. To extract larger messages the limit needs to be changed on the Tuning page of controls:



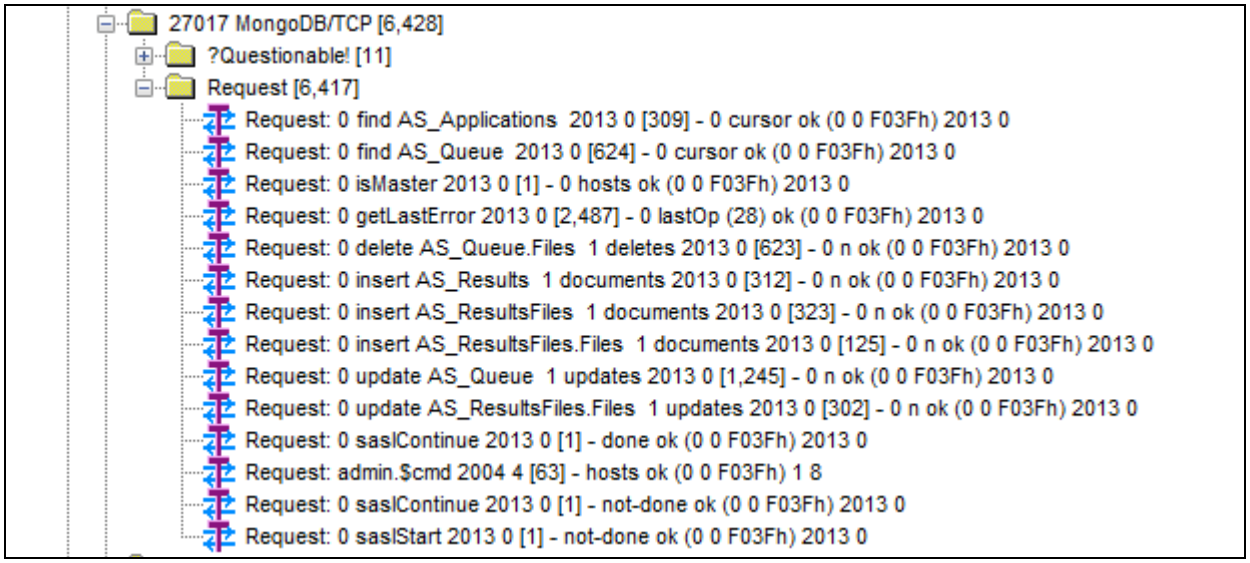
2.14 Alteryx Data Analytics and Workflow

Alteryx is a user-friendly data analytics and workflow package from a US software firm of the same name. NetData now recognises Alteryx HTTP requests from worker nodes to services in a controller node, labelling such traffic in the dialogue chart with the application tag ‘Altrx/HTTP’. This extract from a NetData transaction-class tree lists some different types of transactions with Alteryx services:



Transactions with the Queue and RenderQueue services wait for an item to be queued or time out after 25 seconds. They are put in the Waiting transaction class and not normally loaded for charting.

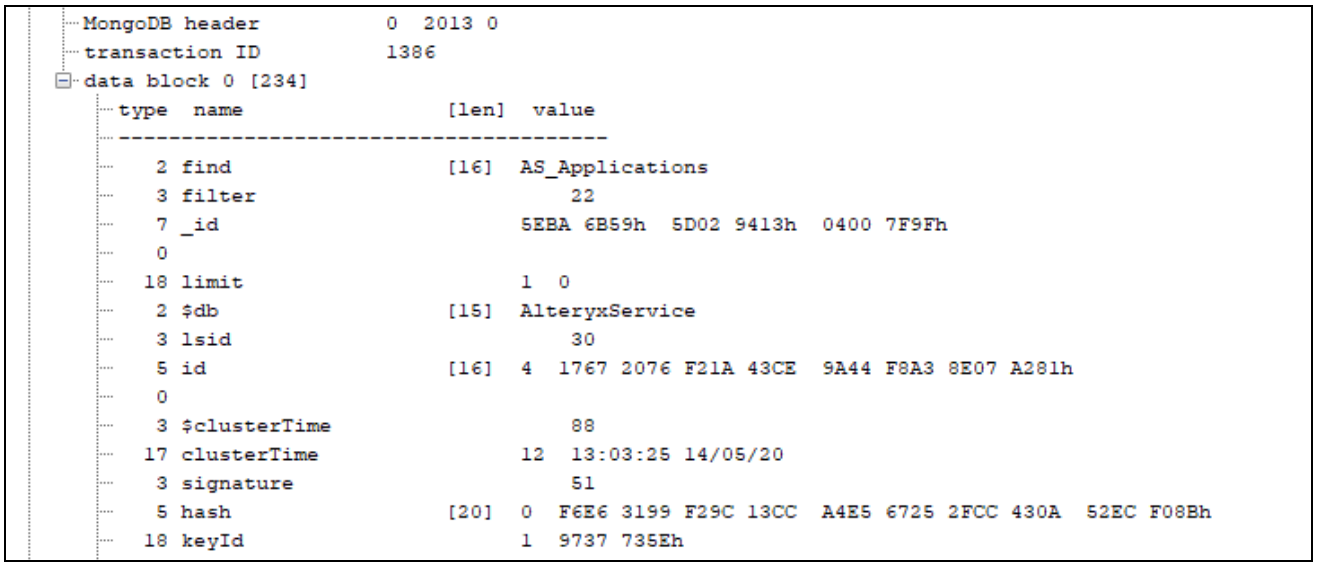
NetData also decodes transactions with Alteryx MongoDB, a scalable, open source, non-SQL database that stores data in a binary JSON format called BSON.



To distinguish different types of MongoDB transactions, NetData assembles request and response signatures with two numbers from the message header (such as ‘2013 0’), the name of the first message parameter and the values of significant parameters such as the ok parameter (e.g. ‘0 0 F03Fh’) in response messages.

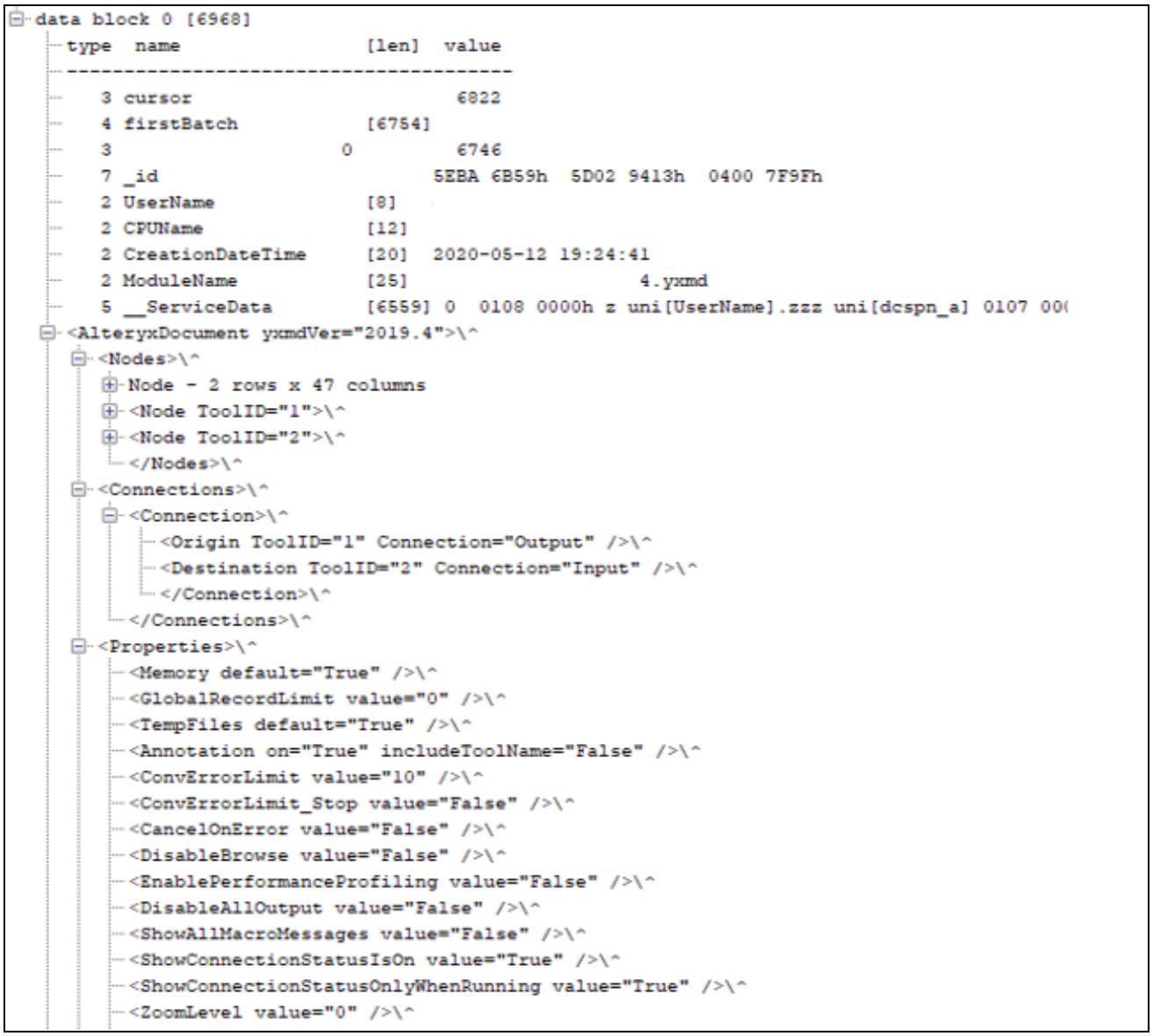
NetData describes message contents with a table of parameters, each represented by a type code, parameter name, and value preceded by an optional length indicator. In the response example below the value of the parameter called ‘__ServiceData’ has a string of 6559 characters starting with Unicode descriptions of parameters, and ending with a large XML document which NetData displays in its usual structured form.

Request message:



```
MongoDB header          0  2013 0
transaction ID          1386
data block 0 [234]
    type  name                [len]  value
    ---------------------------------------
       2 find                 [16]  AS_Applications
       3 filter                         22
       7 _id                          5EBA 6B59h  5D02 9413h  0400 7F9Fh
       0
      18 limit                        1  0
       2 $db                  [15]  AlteryxService
       3 lsid                           30
       5 id                   [16]  4  1767 2076 F21A 43CE  9A44 F8A3 8E07 A281h
       0
       3 $clusterTime                   88
      17 clusterTime                   12  13:03:25 14/05/20
       3 signature                      51
       5 hash                 [20]  0  F6E6 3199 F29C 13CC  A4E5 6725 2FCC 430A  52EC F08Bh
      18 keyId                        1  9737 735Eh
```

Response message:

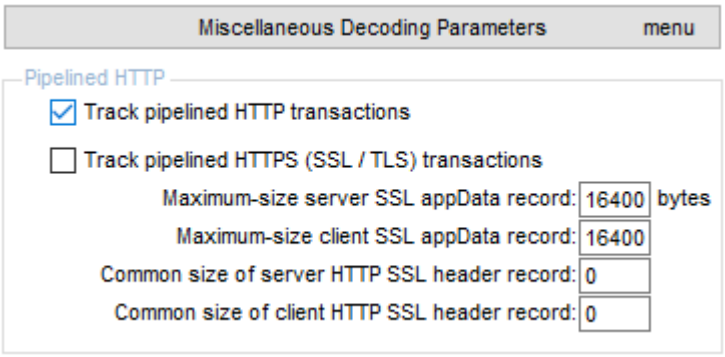


```
data block 0 [6968]
    type  name                [len]  value
    ---------------------------------------
       3 cursor                         6822
       4 firstBatch           [6754]
       3                    0           6746
       7 _id                          5EBA 6B59h  5D02 9413h  0400 7F9Fh
       2 UserName             [8]
       2 CPUName              [12]
       2 CreationDateTime     [20]  2020-05-12 19:24:41
       2 ModuleName           [25]                    4.yxmd
       5 __ServiceData        [6559] 0  0108 0000h z uni[UserName].zzz uni[dcspn_a] 0107 00(
<AlteryxDocument yxmdVer="2019.4">\^
  <Nodes>\^
    Node - 2 rows x 47 columns
    <Node ToolID="1">\^
    <Node ToolID="2">\^
    </Nodes>\^
  <Connections>\^
    <Connection>\^
      <Origin ToolID="1" Connection="Output" />\^
      <Destination ToolID="2" Connection="Input" />\^
      </Connection>\^
    </Connections>\^
  <Properties>\^
    <Memory default="True" />\^
    <GlobalRecordLimit value="0" />\^
    <TempFiles default="True" />\^
    <Annotation on="True" includeToolName="False" />\^
    <ConvErrorLimit value="10" />\^
    <ConvErrorLimit_Stop value="False" />\^
    <CancelOnError value="False" />\^
    <DisableBrowse value="False" />\^
    <EnablePerformanceProfiling value="False" />\^
    <DisableAllOutput value="False" />\^
    <ShowAllMacroMessages value="False" />\^
    <ShowConnectionStatusIsOn value="True" />\^
    <ShowConnectionStatusOnlyWhenRunning value="True" />\^
    <ZoomLevel value="0" />\^
```

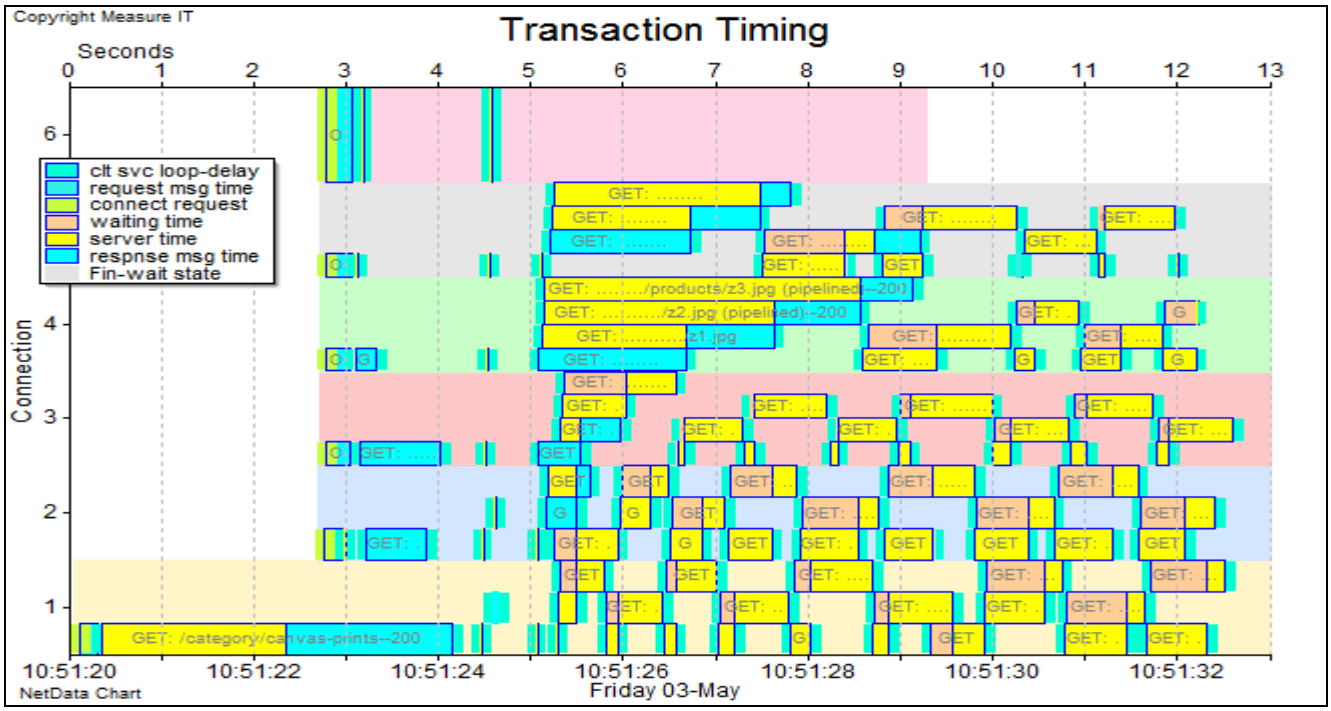
2.15 HTTP Pipelining

A little-used capability of HTTP 1.1 allows multiple requests to be pipelined – issued on the one connection without waiting for corresponding responses. This practice effectively eliminates the propagation delays and request-transmission delays associated with many round-trips.

According to Wikipedia in 2013 only Opera enabled pipelining by default, but some other browsers, including Firefox but not Internet Explorer, offered pipelining as a non-default option. Many Android phones use pipelining and NetData has seen iPhones and iPads use pipelining. If requested by a checkbox on the Decoding page of controls, NetData recognises, tracks and measures pipelined HTTP transactions, rather than assuming that a response packet has been dropped by the sniffer when it sees two requests in succession.



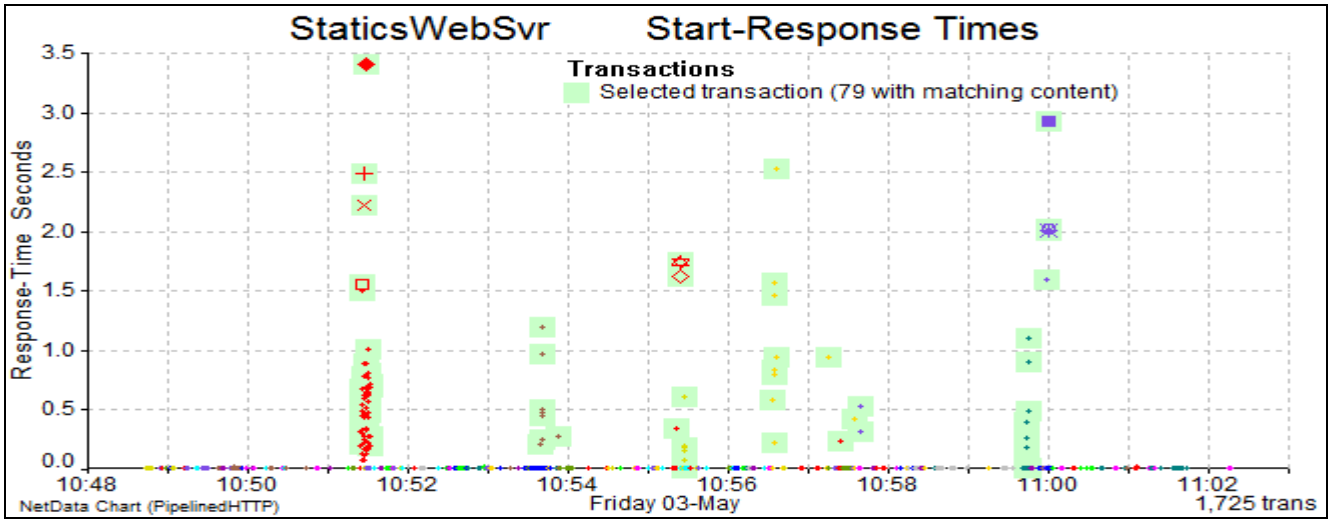
In order that the browser can match responses with requests, the server must return responses in the same order as the requests, and NetData assumes that the order is preserved by using only one thread per connection. If the server receives a request while another is being processed, NetData is able to measure the interval between the time that the request arrives and the earliest time that it can be processed, that is, when the response to the preceding request first appears. That time interval is displayed on timing charts with a light-brown bar, between the blue request-transfer bar and the yellow processing bar. In other circumstances – when there is evidence of a queue of response messages waiting to be transmitted – it is not possible to tell when server processing starts and stops, in which case the brown bar indicates only the smallest possible waiting time and the yellow bar indicates the largest possible (i.e. worst case) processing time.



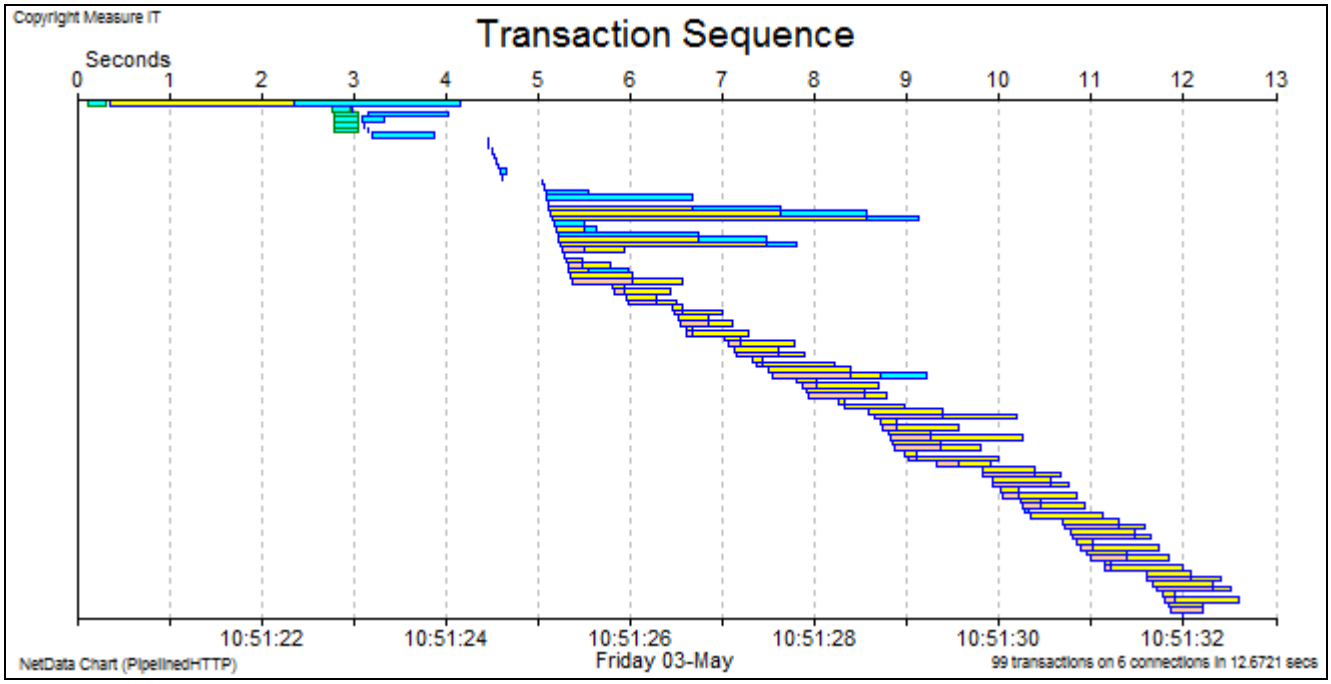
This page loading to an iPhone required 99 file requests and took 12.6 seconds. While receiving the page file the browser requested six style-sheet files on another five connections, and then requested eight javascript files on all six connections. Requests for the 84 image files were pipelined in five connections. NetData appends the text '(pipelined)' to the request signatures of all the transactions that wait in a pipeline.

The true server processing times of the requests in the initial pipeline bursts are quite unknown because all but the first response message had to wait their turn for transmission. Even when there is only one transaction in the pipe the yellow bar that ends with the appearance of the transaction's first response packet is not reliable because that packet might have been delayed by a round-trip time, waiting for an ack to open the transmit window, or release a send-buffer.

The next chart confirms that IIS was able to serve all non-pipelined requests very quickly. For this chart all the captured transactions requesting jpg files were selected in the transaction tree, loaded into the charting module and viewed in its transaction table. The table was searched to mark as selected all the transactions that had the word 'pipelined' in their descriptions, with the result that the markers of pipelined requests appear on pale green squares.

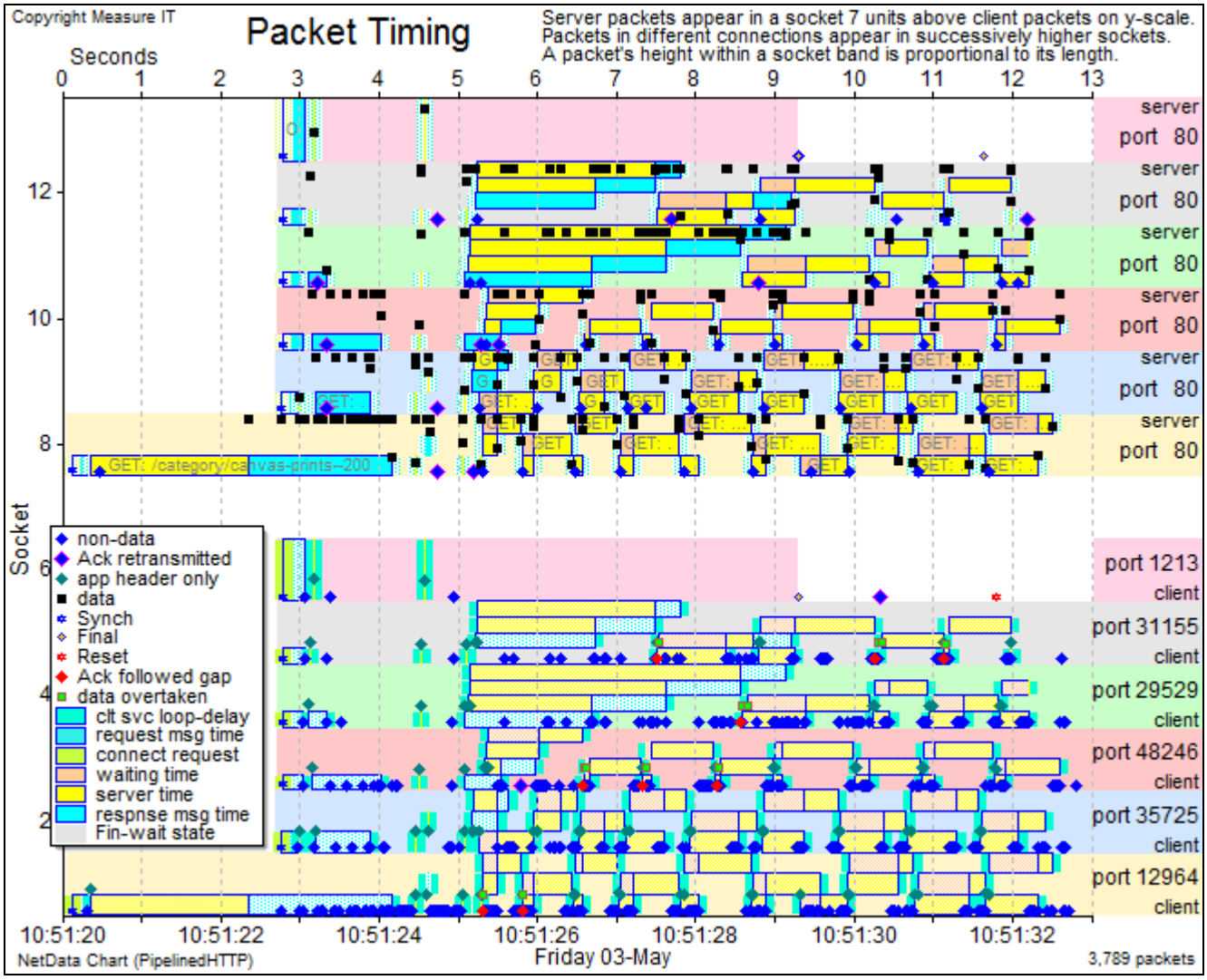


Only the pipelined jpg requests appear to have had response times greater than a few milliseconds.

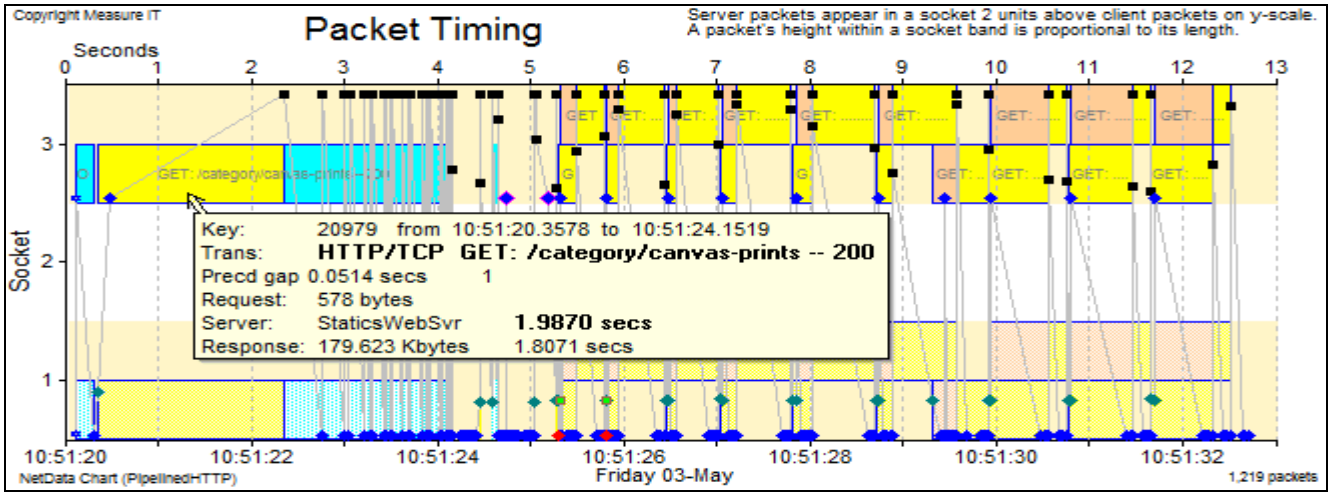


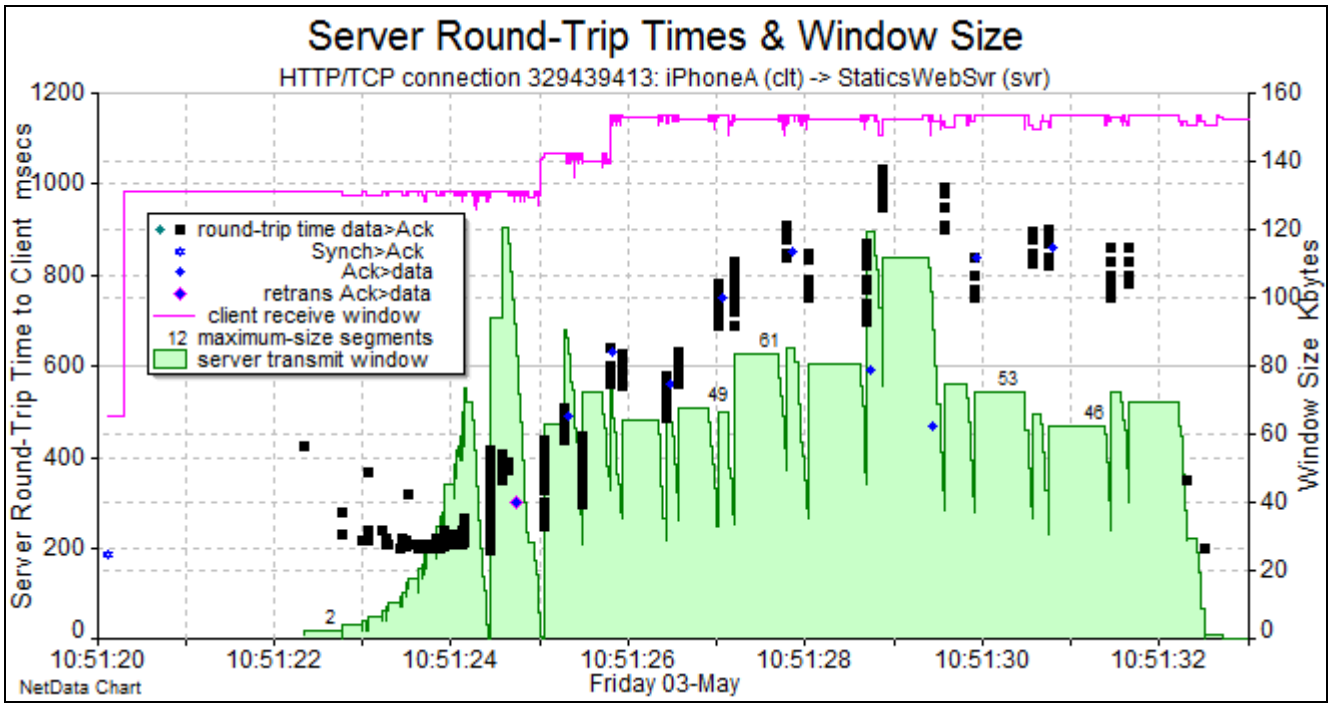
The waterfall chart shows distinct pauses before the javascript requests and before the image requests.

To check that the above characterisation of pipelining by a Mobile Safari browser is not distorted by any network abnormalities such as packet losses or delayed acks, the following packet-timing chart displays all the packets involved in the page loading.

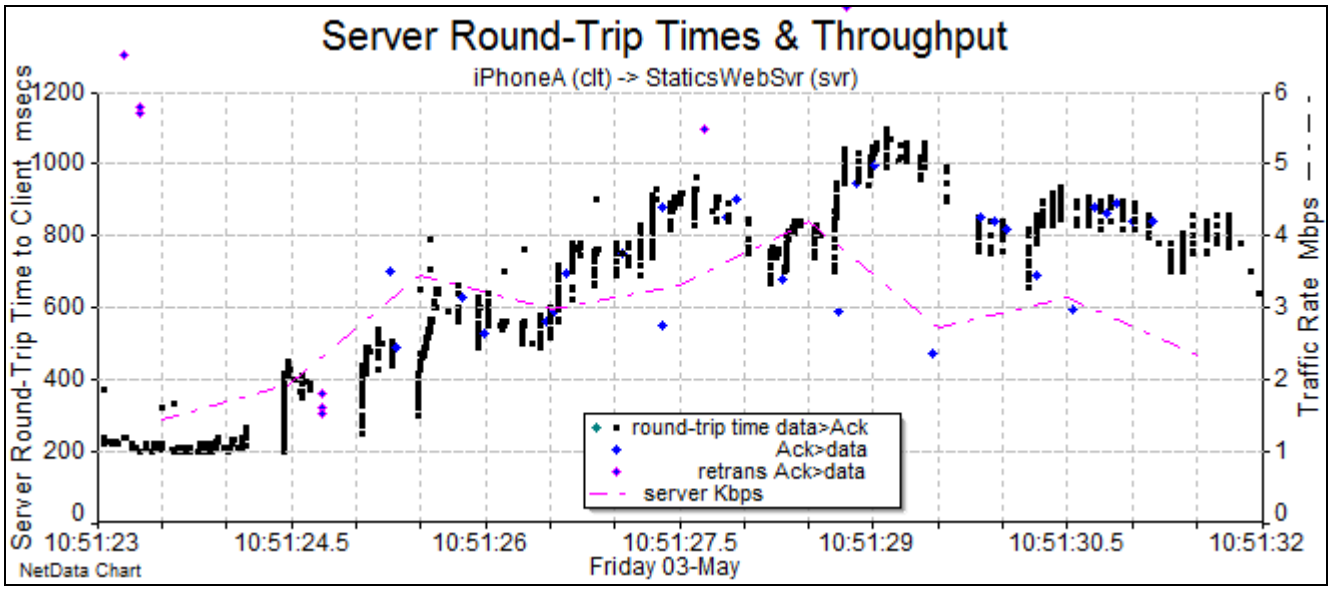


No packets were lost but several request packets from the browser were overtaken by shorter ack packets. The performance of the connection (with cream socket band) that delivered the page file is explored further in the next two charts with the same time scale.



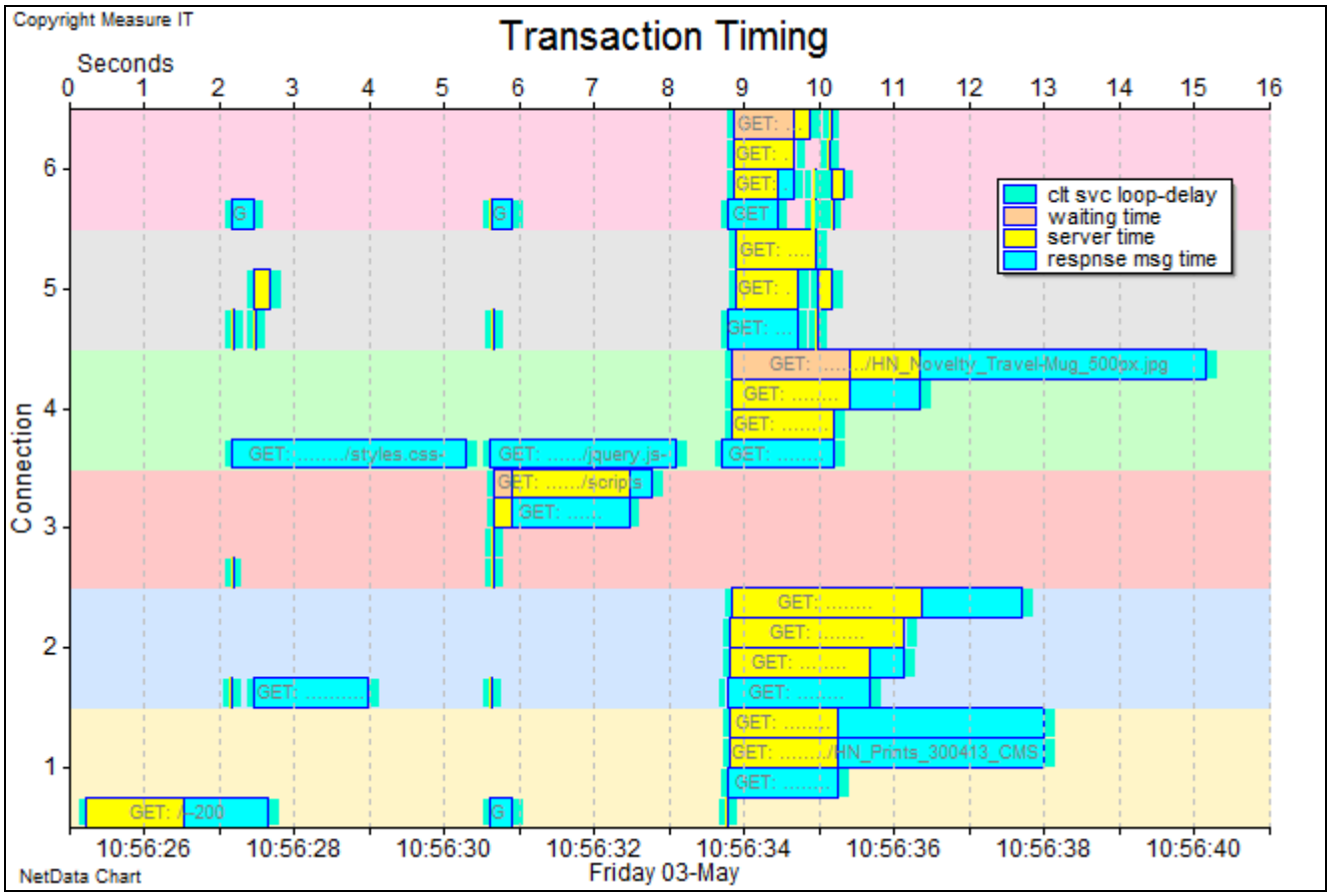


The page file of 179 Kbytes took 1.8 seconds to transfer because the server began with a cautious slow-start, but pipelining was effective in keeping the receive window half full throughout the transfer of image files. The round-trip times of the Synch packet and the first group of data packets were close to the loop-delay of 189 ms, but as the transmit-window size was increased the extra data packets pushed into the network must have queued for a slow link, and round-trip times increased to more than a second. The large round-trip times that delayed acks and consequently delayed the appearance of successive image files on the network, account for the long yellow bars on the timing chart – they indicate queuing delays in the network, not large server processing times.

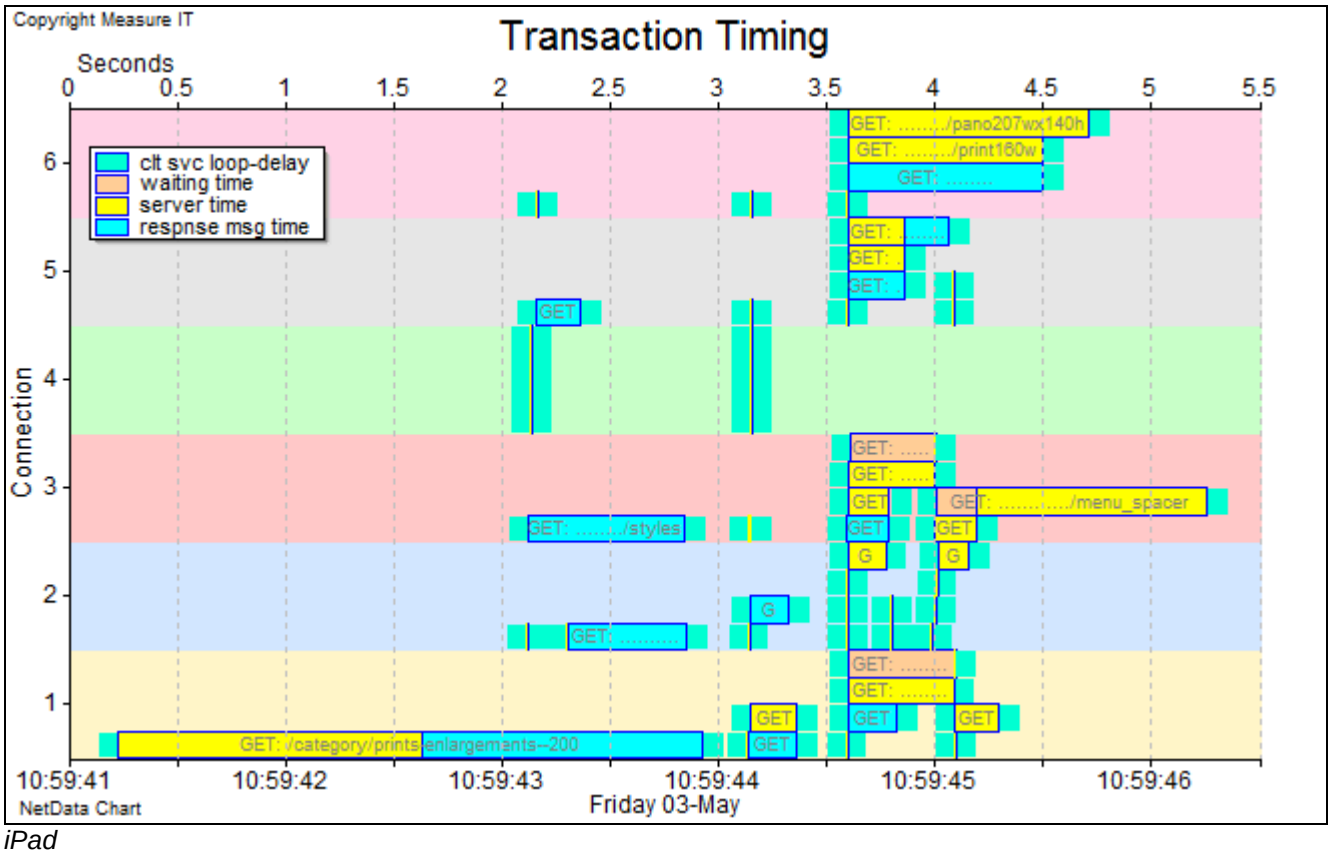


This chart plots the round-trip times and throughput of all packets sent to the iPhone. The path could sustain a throughput of only 3 Mbps.

The following charts display the loading of different pages by an iPhone and an iPad. They confirm that requests for style sheets, javascript and images are issued in distinct stages, but pipelining was used in all stages. The blue-green bars that book-end the bars of a transaction indicate the propagation delay of the transaction's first and last packets.

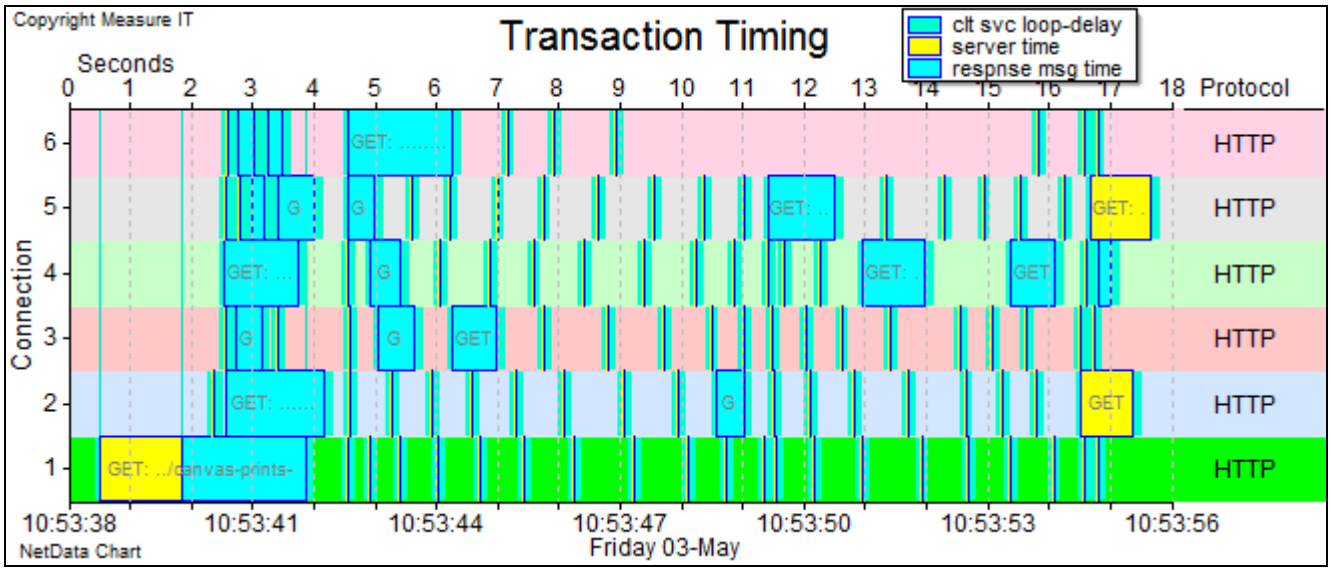


iPhone

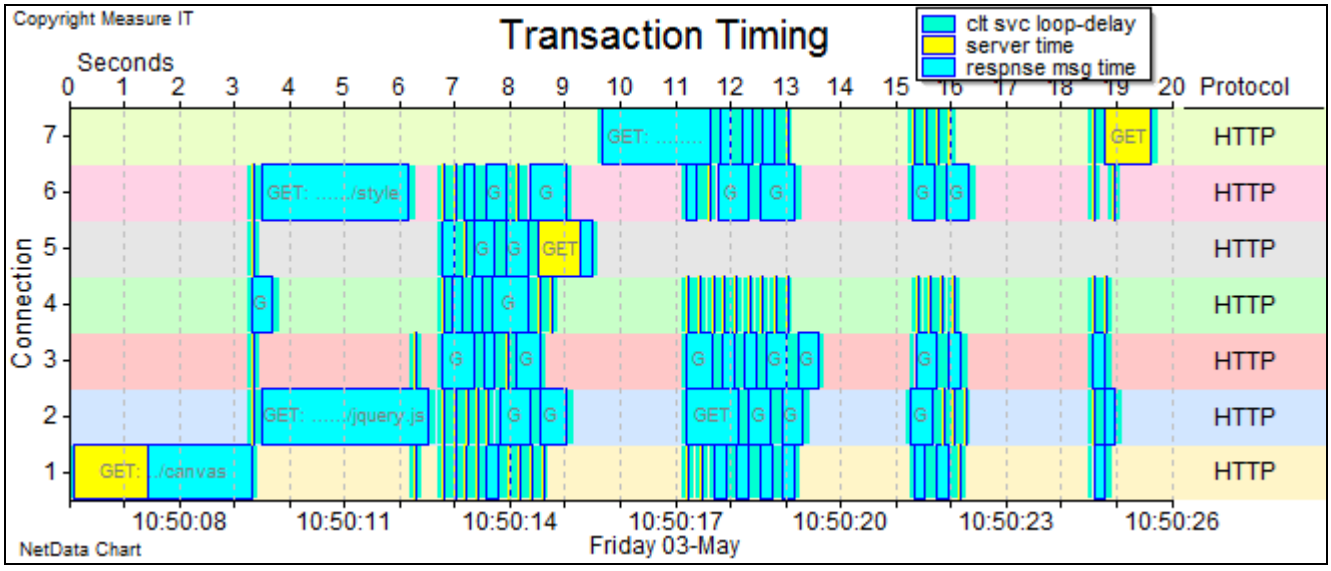


iPad

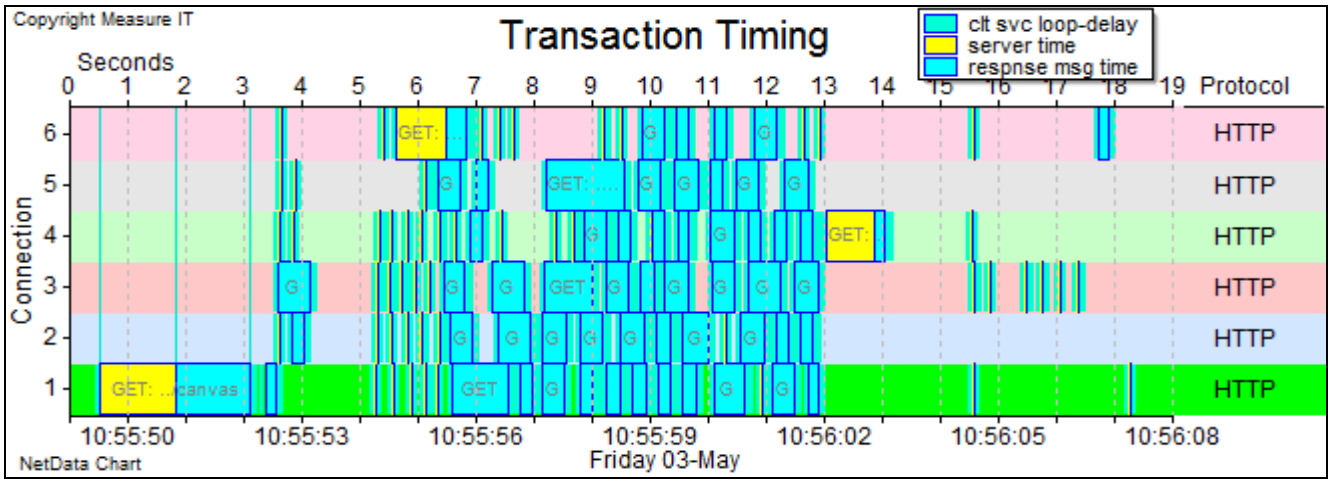
The next three charts display the same *canvas-prints* page being loaded from a US server by unknown PCs in Australia using Internet Explorer.



IE 9.0



IE 8.0



IE 8.0

The two requests with large response times yielded a 404 error code.

These charts show that pipelining by the Apple Safari browser was quite effective in keeping the outbound channel busy and, compared to Internet Explorer, reduced the page-loading time by approximately 5 seconds.

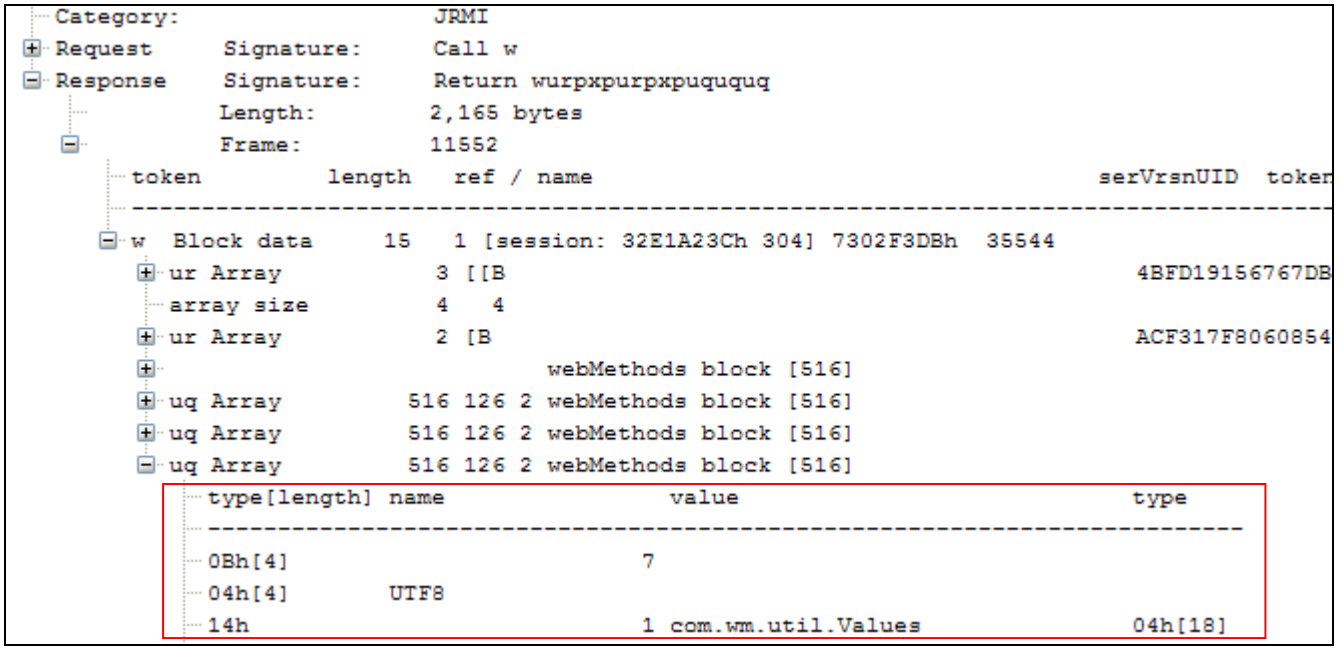
2.16 WebMethods and Java RMI

Messages involved in webMethods broker administration, in traffic labelled by NetData as wmAdmin, use a unique type-length-value format. Some values contain field names and subsequent values contain field contents. NetData displays admin messages with two columns of values to present field names and contents side by side. Type codes are displayed in hex and are followed by a length in brackets, in a column to the left of names or to the right of values.

```
webMethods admin message
   type[length] name                 value                                  type
   ------------------------------------------------------------------------------
   0Bh[4]                          1
   05h                             1 com.wm.data.xxMemDataxxxx              04h[25]
   04h[4]       UUID                 f3d4de30925e11e08xxxxxxxxxxxxxxx      04h[32]
   04h[7]       COMMAND              XXX_CLUSTER_SESSION_CMD               04h[23]
   04h[6]       HEADER                                                   1 05h
   00h                             2
   04h[7]       Session              false                                 04h[5]
   04h[13]      Authorization        QWRtaW5pc3RyYXRvcjzzzzzzzzzzzzzz      04h[32]
   04h[11]      MessageType                                              3 04h[1]
   04h[12]      Content-Type         application/x-wmidatabin              04h[24]
   04h[6]       Cookie
   02h                             0
```

Although NetData decodes all admin messages it measures the response times of only ping messages and their replies. Isolated commands without replies are not recorded as part of any transaction.

A server running webMethods is likely to handle service requests conveyed in the form of Java serialised objects, in traffic labelled by NetData as JavaRMI (Remote Method Invocation) or TSRMclt (client of Tivoli Service Request Manager), and those requests and their responses often contain embedded data blocks in the webMethods admin message format. NetData handles Java objects embedded within other objects, and recognises embedded webMethods blocks. It displays them in their native tabular format within the structure of a Java object:



This JRMI response contained a Java array with four embedded webMethods blocks. The last block (in the red box) has been expanded to show part of its contents.

2.17 AJPv1.3 (Apache JServ Protocol) and Engaged Threads

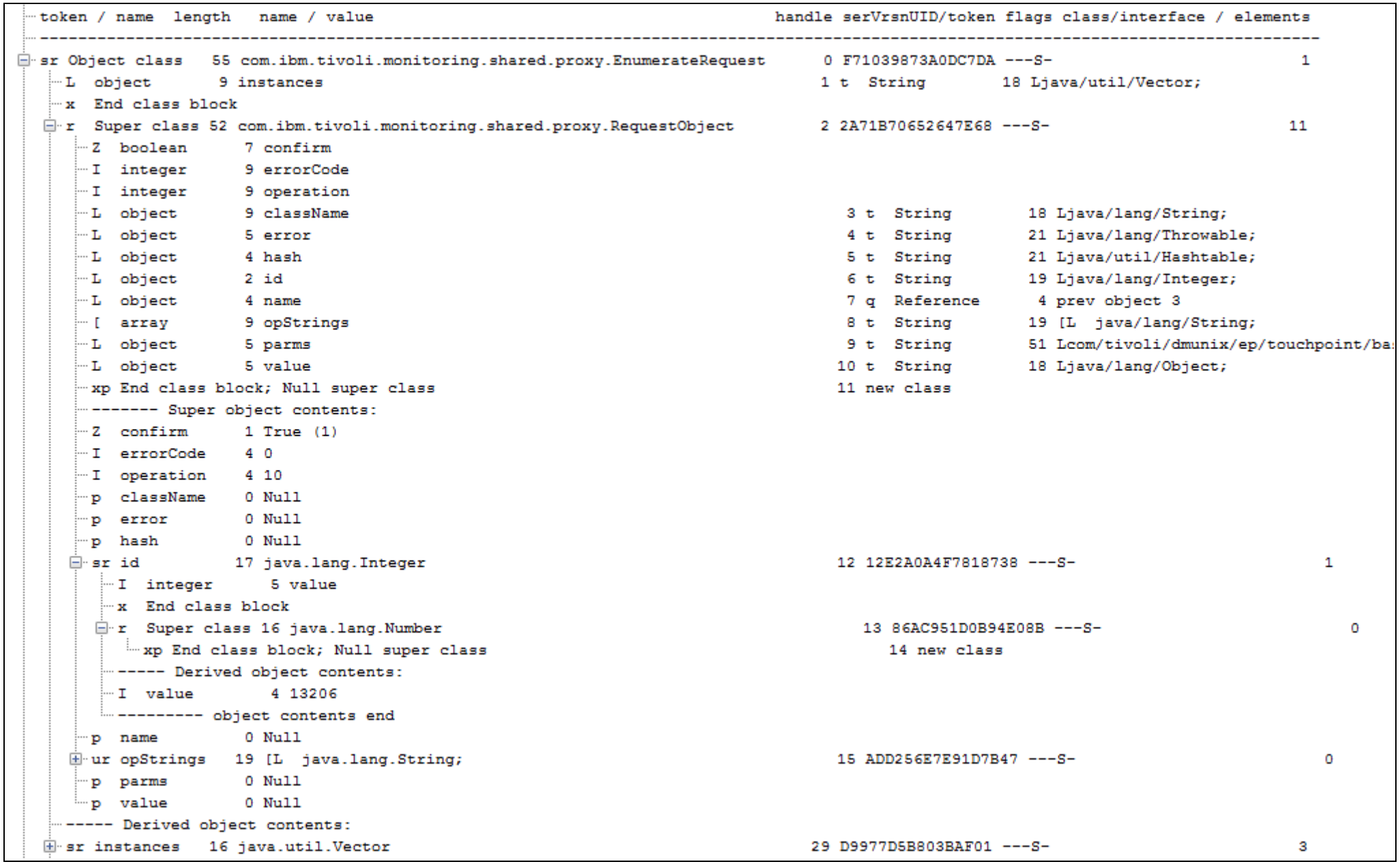
An Apache web server may be used as a load balancer, parsing HTTP requests and their cookies to determine the most appropriate backend web server to answer each request. A farm of backend web servers may use Java servlets in Tomcat containers, running under Apache or IIS. Connectors between the front-end and backend web servers convey the parsed HTTP requests in a special protocol known as the Apache JServ Protocol version 1.3 (AJP13). NetData fully decodes messages in this protocol and treats it much like HTTP with the option of extracting transaction signatures, user IDs and key data from headers, cookies and message bodies. If NetData is asked to define users by their client IP address, the AJP decoder will extract the remote IP address from AJP request messages.

An Apache front-end web server accepts new connection requests from users immediately, even if there is no thread available for assignment to the connection. If there is a shortage of threads, the first request on a connection may be delayed many seconds until a thread is available. NetData can display evidence of such a bottleneck in several ways, but the simplest is to load all the records of user connections and plot the number of concurrent connections. The format control for this graph has an option to count new connections only when a thread has been engaged, as indicated by the appearance of the first transaction response. If NetData finds AJP traffic in a capture it will refine the thread engagement time by recording the time at which a connection's first request message is relayed to a backend server. The resulting graph of concurrent, engaged threads provides a clear indication of any thread bottleneck, and the total number of configured threads.

2.18 Serialised Java Objects

NetData decodes streams of serialised Java objects that are found in a wide range of message contexts. The grammar of these streams is defined in Chapter 6 of *Java Object Serialization Specification*, Revision 1.5.0, by Sun Microsystems (2001), and in later editions published by Oracle. As illustrated in the panel on the next page, NetData displays a stream as a table with rows of two types: object definitions and object contents. Most definition rows present the attributes of either a class or one of its fields. Although rows are indented to indicate an object’s tree structure and form sub-tables, the heading is displayed only once to save space, and some columns have different types of entries for definition and content rows.

The first column always contains a stream token and is followed by either a *token* description and object *name*, or a field *name* and *value*. All object names and field values are preceded by a length indicator. The next column contains an object serial number which serves as its *handle* when referenced by an object later in the stream. In the panel below for example, Object 7 is defined as a string object by referring to the previous Object 3. Following the handle, a class definition is represented by an 8-byte hex code conveying a Stream Unique ID, a set of five property flags, and the number of serializable fields; each object field is defined by a string in ‘field descriptor’ format.


token / name length name / value handle serVrsnUID/token flags class/interface / elements
sr Object class 55 com.ibm.tivoli.monitoring.shared.proxy.EnumerateRequest 0 F71039873A0DC7DA ---S- 1
L object 9 instances 1 t String 18 Ljava/util/Vector;
x End class block
r Super class 52 com.ibm.tivoli.monitoring.shared.proxy.RequestObject 2 2A71B70652647E68 ---S- 11
Z boolean 7 confirm
I integer 9 errorCode
I integer 9 operation
L object 9 className 3 t String 18 Ljava/lang/String;
L object 5 error 4 t String 21 Ljava/lang/Throwable;
L object 4 hash 5 t String 21 Ljava/util/Hashtable;
L object 2 id 6 t String 19 Ljava/lang/Integer;
L object 4 name 7 q Reference 4 prev object 3
[array 9 opStrings 8 t String 19 [L java/lang/String;
L object 5 parms 9 t String 51 Lcom/tivoli/dmunix/ep/touchpoint/ba
L object 5 value 10 t String 18 Ljava/lang/Object;
xp End class block; Null super class 11 new class
------- Super object contents:
Z confirm 1 True (1)
I errorCode 4 0
I operation 4 10
p className 0 Null
p error 0 Null
p hash 0 Null
sr id 17 java.lang.Integer 12 12E2A0A4F7818738 ---S- 1
I integer 5 value
x End class block
r Super class 16 java.lang.Number 13 86AC951D0B94E08B ---S- 0
xp End class block; Null super class 14 new class
----- Derived object contents:
I value 4 13206
--------- object contents end
p name 0 Null
ur opStrings 19 [L java.lang.String; 15 ADD256E7E91D7B47 ---S- 0
p parms 0 Null
p value 0 Null
----- Derived object contents:
sr instances 16 java.util.Vector 29 D9977D5B803BAF01 ---S- 3

The set of five flags in a class description indicate various properties of the class:

W	defines a writeObject method
S	is serializable
X	is externalizable
B	contains block data
E	is an enumerated type

The object serialised in the panel above is in the application's *EnumerateRequest* class that is derived from the *RequestObject* class – its Super (parent) class. An object may have a chain of super classes and the end of the chain is indicated by a null token ('p').

The object's contents follow the chain of class definitions and begin with the fields of the highest (i.e. last) super class. Fields which are themselves objects and are not represented by a null token require the definition of a new object. For example, the object field called 'id' belongs to the *lang.Integer* class derived from the *lang.Number* class, and here contains the 4-byte integer 13206.

If NetData loses track of an object's structure or data, it immediately terminates the existing indentation and starts again looking for significant tokens.

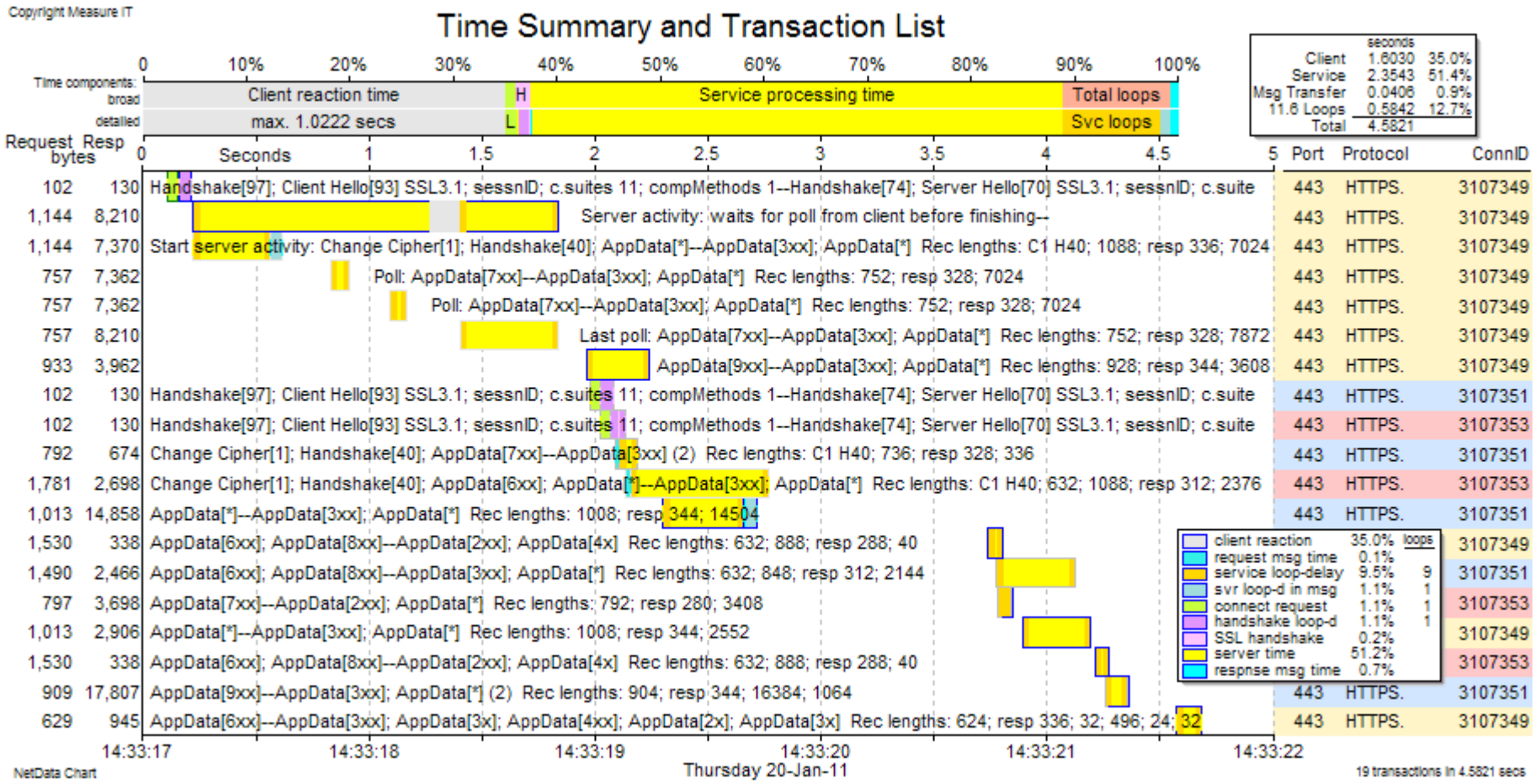
2.19 Modelling Web Transactions with a Polling Mechanism

Widespread adoption of Ajax techniques in web applications has produced some novel means of running transactions in the server while rendering a new web page. For example, a Post request might launch a substantial processing job in the server, but a response is returned immediately to indicate only that the request has been accepted. While the server runs the job, the browser updates the displayed page and alternates between checking for new user inputs and sending Get requests to the server to check the status of its job. If the job has finished, the response to such a poll contains the results; otherwise the response indicates that the job is still running.

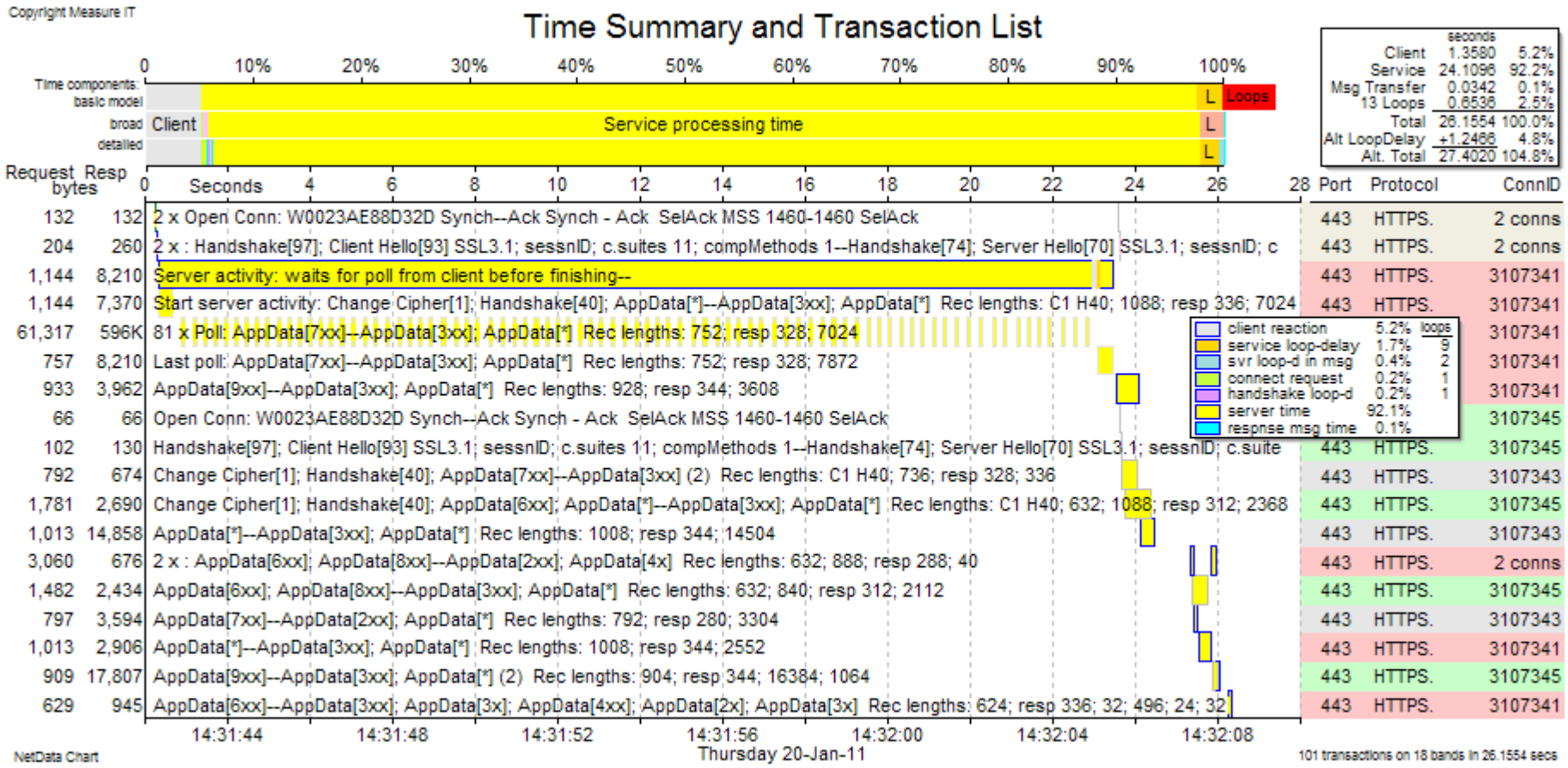
A Timing or Transaction List chart will correctly display the repeated polls (status requests) but the count of round-trips (loop-delays) is misleading because none is on the user-transaction's critical path. New options in the context menu of a Transaction List chart allow you to nominate the server transaction (such as a Post request) that starts server activity, and also to nominate the type of transaction that polls the server to determine whether it has finished. NetData then adds a composite transaction to the chart that represents the critical-path server activity, and removes the Start and Poll transactions from the critical path. The resulting summary of time components then accurately reflects the number of non-overlapping round-trips, server time and client time.

The inevitable consequence of a polling mechanism is that some time is wasted while the server waits for a poll, before it can return its results. NetData measures the time interval between the last two polls and calculates an average poll delay which is half the last polling interval. Because each polling interval includes one loop-delay, the average poll delay therefore includes half a loop-delay, and half a loop is added to the overall number of service loops. The remainder of the average poll delay is attributed to client time, and this is indicated by a grey (client) bar plotted part way through the yellow (service) bar of the composite transaction.

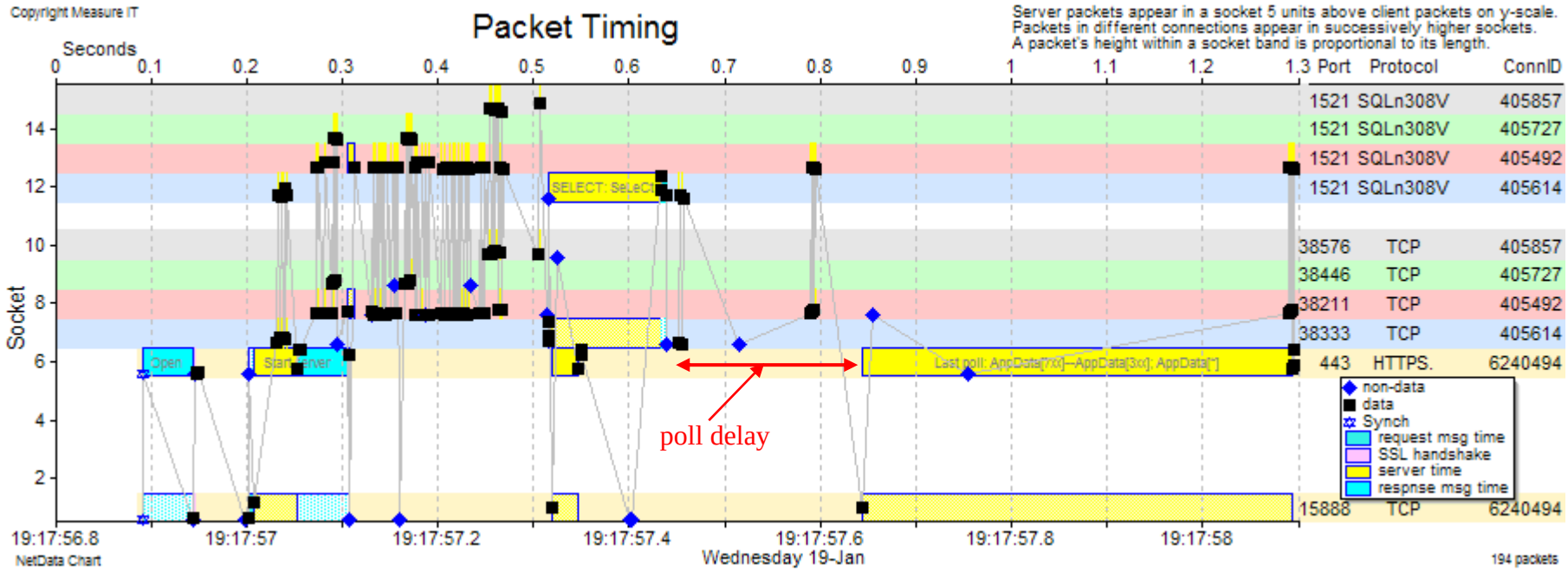
The sample chart below shows a composite transaction on the second row.



Below the composite transaction are its four constituents: a request that starts server activity, and three polls to check the status of the server activity. As usual, the server transactions (round-trips) on the critical path have a dark blue outline. The javascript controlling this polling mechanism paused for a tenth of a second before each poll, but javascript processing and one loop-delay increased the last poll interval to 300 ms, making the average poll delay 150 ms.



A similar transaction took 23 seconds for the main job in the server and saw 82 polls before it could return its results to the client. Although there were 101 round-trips in total, only 13 trips were on the critical path. If moving the workstation added nearly 100 ms to the loop-delay, the overall response time would increase by only 1.3 seconds.



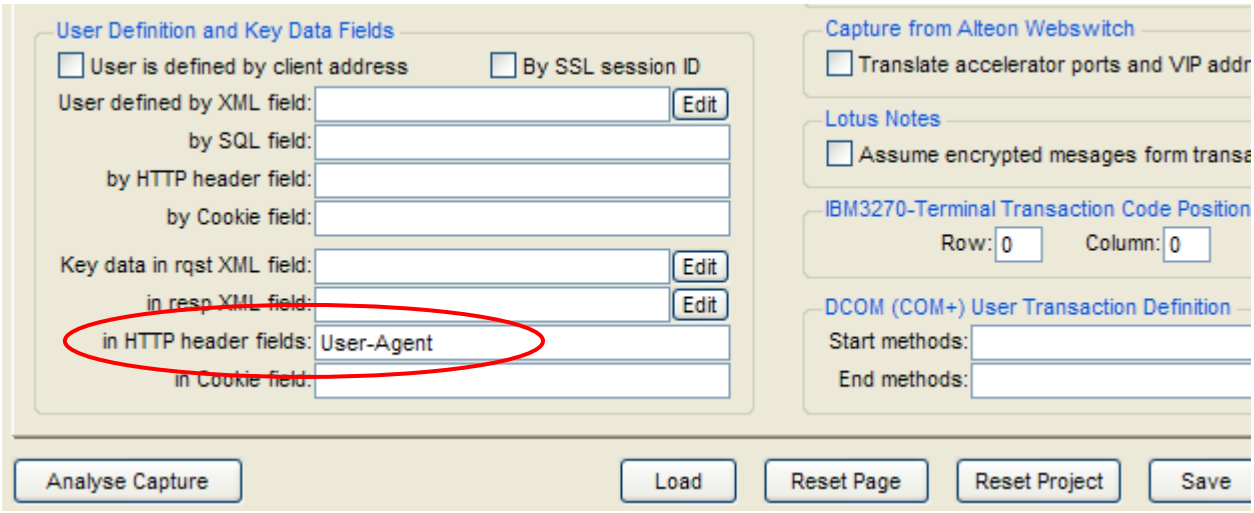
This chart shows both the backend database traffic and the browser traffic while the browser polled the server to get the results of the database activity. The last significant database transaction in this case was a query that took 120 ms, and the server then waited 200 ms (poll delay) for the second and last poll. The database activity involved four connections and included regular database polls at half-second intervals that were not related to the browser's transaction.

.

2.20 User-Agent Types

Today large public web sites handle requests from a wide variety of browsers, some running in workstations with large displays and broadband Internet access, and many in mobile devices with small screens and limited bandwidth – not to mention large numbers of web crawlers, spiders and bots. A successful site will probably 'sniff' the user-agent field in HTTP request headers and tailor its output web pages to suit the browsers.

NetData will extract the contents of any HTTP header field for display in the transaction table's Data column if the field is named on the Decoding page of controls:

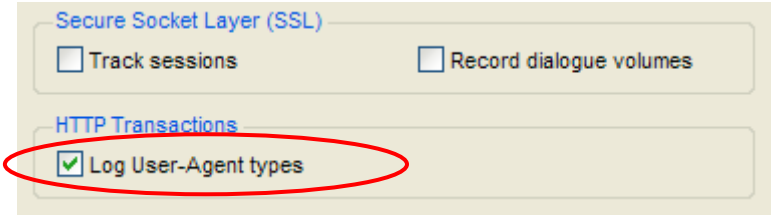


The field name is generally not case sensitive, but, for this field, if its capitalisation matches 'User-Agent' exactly, NetData condenses the agent descriptions for the Data column:

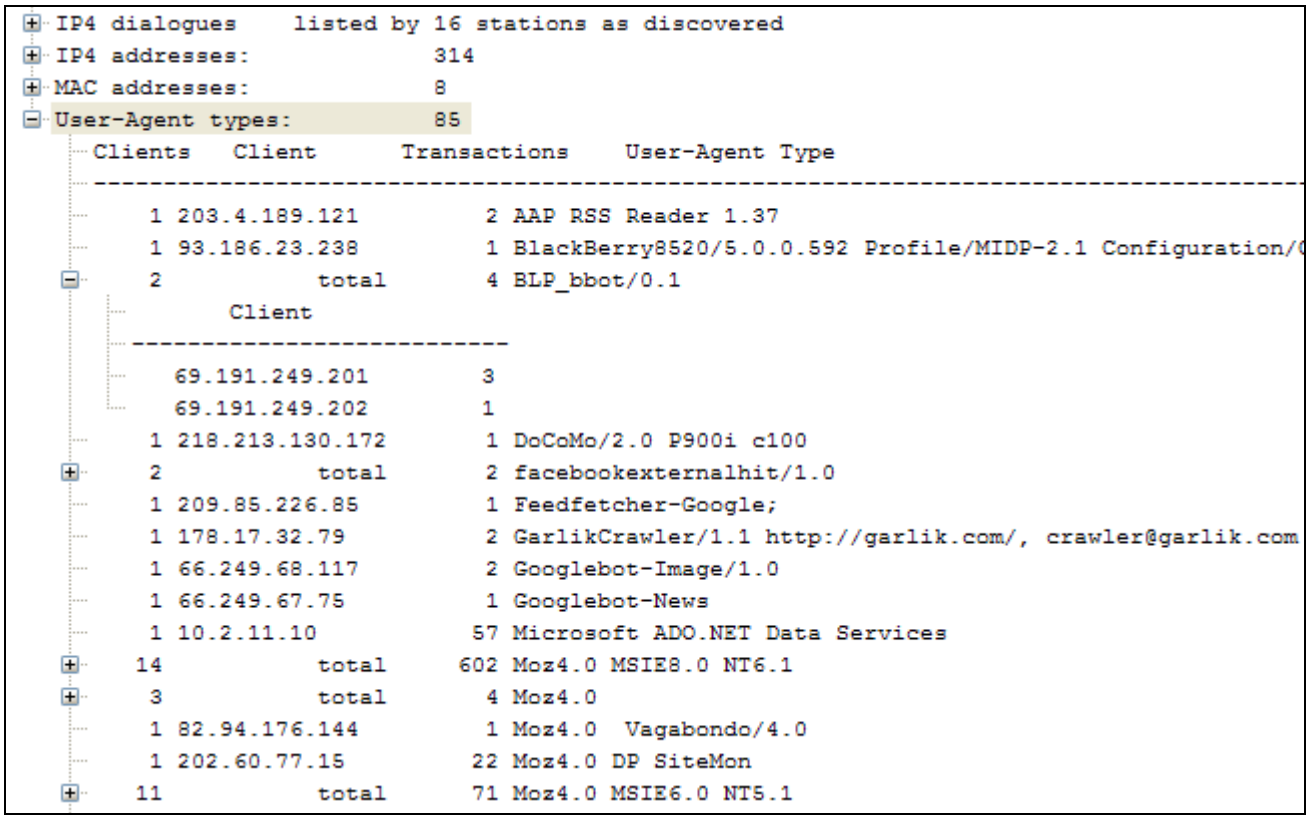
	Trn Key	Request Strt	Description	End Rsp	Client	Data
•	843	14:26:29.2565	GET: /js/jquery.timers-1....	0.0005	203.47.147.42	Moz5.0 NT5.1 Gecko/20110803 Firefox/3...
•	1659	14:26:29.3715	GET: /LinkClick.aspx--200	2.7480	58.163.175.191	Moz5.0 iPhone OS 4_1 Mobile/8B117 Saf...
•	865	14:26:29.3718	GET: /Resources/Share...	0.0003	203.47.147.42	Moz5.0 NT5.1 Gecko/20110803 Firefox/3...
•	873	14:26:29.375	GET: /Resources/Share...	0.0148	58.163.175.191	Moz5.0 iPhone OS 4_1 Mobile/8B117 Saf...
•	870	14:26:29.3784	GET: /Portals/11/Skins...	0.0001	203.47.147.42	Moz5.0 NT5.1 Gecko/20110803 Firefox/3...
•	1367	14:26:29.3788	GET: /Portals/16/Skins...	1.2918	41.134.23.98	Moz4.0 MSIE7.0 NT5.1
•	1295	14:26:29.3899	GET: /NewsEvents/Ne...	0.8804	203.59.155.95	Moz4.0 MSIE8.0 NT6.1
•	875	14:26:29.3945	GET: /Portals/11/Image...	0.0001	203.47.147.42	Moz5.0 NT5.1 Gecko/20110803 Firefox/3...
•	876	14:26:29.3948	GET: /Portals/11/Image...	0.0001	203.47.147.42	Moz5.0 NT5.1 Gecko/20110803 Firefox/3...

In the condensed descriptions 'Mozilla' is abbreviated to 'Moz', and the names of some loaded programs are suppressed.

Statistics for agent types can be investigated in another way by checking the box 'Log User-Agent types' on the Output page of controls:



At the end of an analysis NetData records all the different agent types, with their usage statistics, in a structured table at the end of the _conn.log file. Each agent type in the table may have a subsidiary table that lists all the clients using the agent type:



```
IP4 dialogues    listed by 16 stations as discovered
IP4 addresses:              314
MAC addresses:              8
User-Agent types:           85
   Clients   Client      Transactions    User-Agent Type
   ------------------------------------------------------------------------------------------
       1 203.4.189.121          2 AAP RSS Reader 1.37
       1 93.186.23.238          1 BlackBerry8520/5.0.0.592 Profile/MIDP-2.1 Configuration/C
       2           total        4 BLP_bbot/0.1
             Client
          --------------------------
            69.191.249.201        3
            69.191.249.202        1
       1 218.213.130.172         1 DoCoMo/2.0 P900i c100
       2           total        2 facebookexternalhit/1.0
       1 209.85.226.85           1 Feedfetcher-Google;
       1 178.17.32.79            2 GarlikCrawler/1.1 http://garlik.com/, crawler@garlik.com
       1 66.249.68.117           2 Googlebot-Image/1.0
       1 66.249.67.75            1 Googlebot-News
       1 10.2.11.10             57 Microsoft ADO.NET Data Services
      14           total       602 Moz4.0 MSIE8.0 NT6.1
       3           total        4 Moz4.0
       1 82.94.176.144           1 Moz4.0  Vagabondo/4.0
       1 202.60.77.15           22 Moz4.0 DP SiteMon
      11           total        71 Moz4.0 MSIE6.0 NT5.1
```

The table displays the total number of transactions generated by each agent type, and the number of transactions generated by each client using the type.

As with any structured table in a log file or description window, double clicking the parent branch – 'User-Agent types:' in this case – displays the table in a separate window that can be sorted, filtered and exported to a CSV or JSON file:

row	Clients	Client	Transactions	User-Agent Type
24	1	203.82.99.170	33	Moz5.0 Macintosh Intel Mac OS X 10_6_6 Safari/533.20.27
25	1	209.18.124.32	1	Moz5.0
26	6	total	6	Moz5.0 bingbot/2.0
27	1	121.218.79.11	1	Moz5.0 BlackBerry Mobile Safari/534.11+
28	1	206.53.148.240	1	Moz5.0 BlackBerry Mobile Safari/534.8+
29	1	38.101.148.126	5	Moz5.0 discobot/1.1
30	11	total	19	Moz5.0 Googlebot/2.1
31	1	58.178.248.160	154	Moz5.0 iPad OS 4_2_1 Mobile/8C148 Safari/6533.18.5
32	1	137.154.73.31	54	Moz5.0 iPad OS 4_3_3 Mobile/8J2 Safari/6533.18.5
33	1	121.212.16.177	10	Moz5.0 iPad OS 4_3_5 Mobile/8L1 Safari/6533.18.5
34	1	58.163.175.185	5	Moz5.0 iPhone OS 3_1_2 Mobile/7D11 Safari/528.16
35	2	total	124	Moz5.0 iPhone OS 4_1 Mobile/8B117 Safari/6531.22.7
36	1	175.33.107.67	3	Moz5.0 iPhone OS 4_2_1 Mobile/8C148
37	1	117.3.4.98	22	Moz5.0 iPhone OS 4_2_1 Mobile/8C148 Safari/6533.18.5
38	3	total	82	Moz5.0 iPhone OS 4_3_3 Mobile/8J2 Safari/6533.18.5
39	2	total	107	Moz5.0 Macintosh Intel Mac OS X 10_5_8 Safari/533.22.3
40	1	121.217.102.96	26	Moz5.0 iPhone OS 4_3_5 Mobile/8L1
41	2	total	41	Moz5.0 iPhone OS 4_3_5 Mobile/8L1 Safari/6533.18.5
42	1	58.172.77.28	38	Moz5.0 Macintosh Intel Mac OS X 10.6 Gecko/20090824 Firefox/3.5.3
43	1	124.179.9.14	61	Moz5.0 Macintosh Intel Mac OS X 10.6 Gecko/20100101 Firefox/4.0.1
44	1	49.191.163.118	4	Moz5.0 Macintosh Intel Mac OS X 10.6 Gecko/20100101 Firefox/5.0.1
45	1	124.169.76.114	24	Moz5.0 Macintosh Intel Mac OS X 10.7 Gecko/20100101 Firefox/5.0.1

2.21 Extracting Fields in HTTP Query Strings

Some application protocols based on HTTP specify schemes for requesting data that may use either XML bodies with HTTP Post requests, or key-value pairs (KVPs) in query strings with HTTP Get requests. Such protocols will allow both. In order to extract the values of named fields, in either XML or KVP strings, NetData searches for given field names in both types of message. NetData's Decoding page of controls has controls for separately naming XML fields that define a user ID, key data, and extensions to a transaction's request signature, and the given names would also be searched in serialised Java objects and .NET objects. The given names are also searched in query strings.

2.22 Tables in JSON Messages

When displaying a message that uses the JSON (JavaScript Object Notation) format NetData constructs tables from lists of array elements and displays each table at the start of its list.

Like all tables appearing in a NetData tree structure they can be expanded in the tree and displayed in a separate window which can be scrolled, sorted, filtered and exported to a spreadsheet.

2.23 Tables in XML Documents

When displayed in a NetData tree structure tables extracted from XML documents include columns for element attributes as well as for element contents. The heading of an attribute column contains the attribute's name with an equals sign appended. The headings of nested elements use the same conventions as the headings for nested objects and arrays in JSON messages: the names of nested elements are preceded by a number of dots equal to the nesting depth.

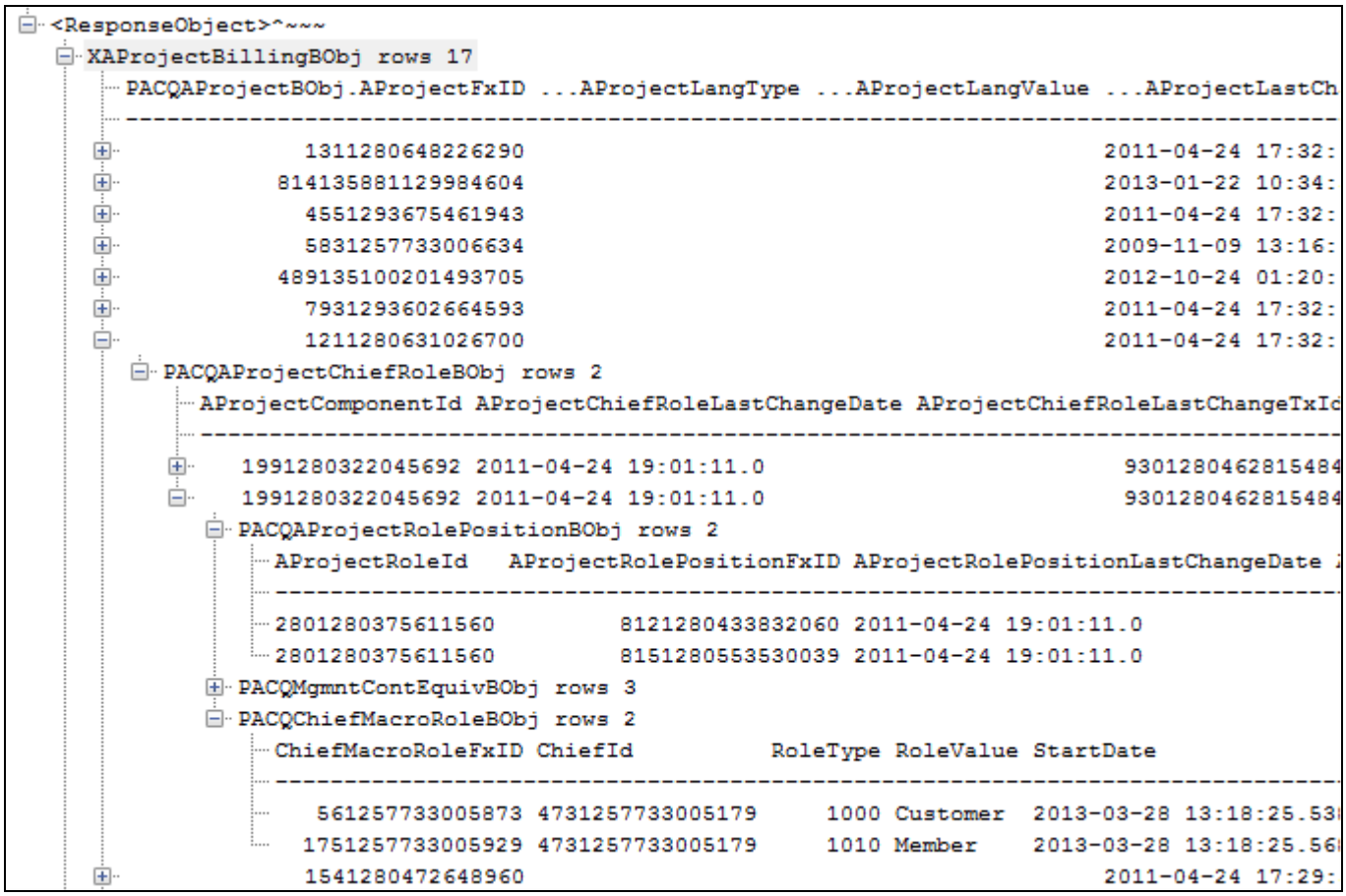
The name of each nested table appears in the parent table at the head of a column whose data entries indicate the number of sub-table rows, as in `array[5]`; an entry of `[]` indicates an empty sub-table or empty nested element. If a sub-table has only one row the contents of that row are promoted to appear in the parent table rather than in a single-row sub-table.

2.23.1 Viewing Tables in XML Messages

To identify inefficiencies and other forms of unhealthy behaviour in application processing it helps to have a clear understanding of the contents of application messages – for example, what questions are asked in individual request messages, and how does the data in response messages relate to those questions? These aspects might be quite clear in a database query, particularly when NetData displays the structure and detail of an SQL Select statement and the table of result-set data, but they are not so clear when message contents are specified with XML.

Although NetData displays the structure and detail of XML messages in its familiar tree form, XML's inherent verbosity tends to obscure the presence of tables and the patterns in their data. XML specifies tables as sequences of similar records, and NetData displays their contents in tabular form as well as the conventional XML form. It handles sub-tables and their sub-tables, nested to any depth.

A sub-table is recognised as a sequence of similar sub-records, but if there is only one sub-record of the same type NetData treats it not as a one-row table but as an extension of the parent record. Record extensions can be nested to any depth and are displayed in extra columns added to the parent table.



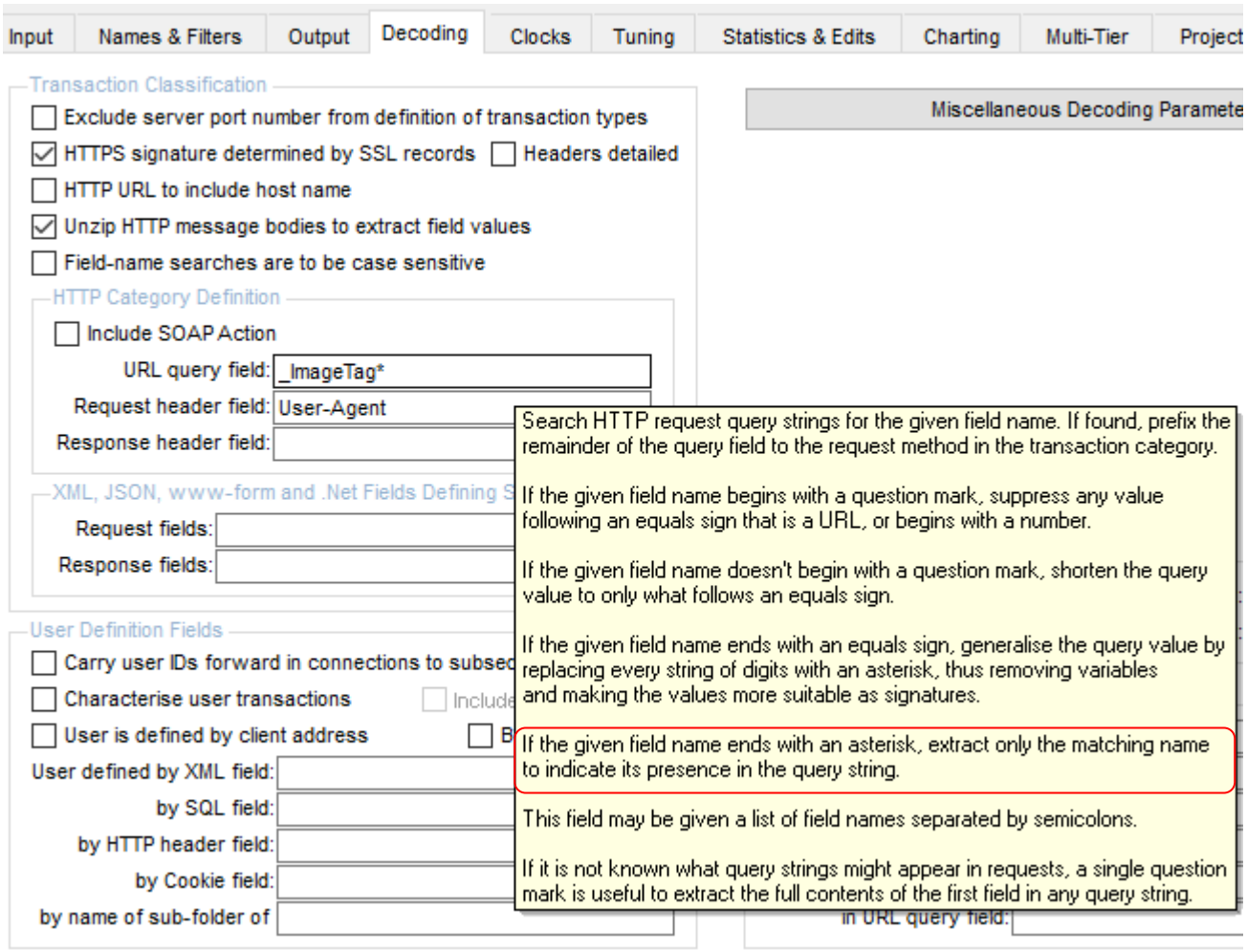
This extract from NetData's display of an XML message shows a parent table with 17 rows. The column headings – concatenations of tag and parent-tag names – indicate that this table has no fields of its own but has a sub-record tagged `PACQAProjectBObj`. Parent tag names are not repeated but replaced by an ellipsis. (…). The seventh row has been expanded to reveal a sub-table with two rows, and its second row has been expanded to reveal three sub-tables. The first and third of those sub-tables have also been expanded. As with the tables in any NetData tree, any table or sub-table can, with a double click, be displayed, browsed, sorted and filtered in a separate window.

2.24 Tables in HTML Web Documents

When displaying a message representing an HTML web document NetData reconstructs tables from TABLE, TR (table row), TH (table header) and TD (table data) tags, even when tables are nested within the data cells of other tables, to any depth. The content of a data cell is represented by text between tags. If a cell contains many DIV, SPAN, A (hyper link) and other tags with many texts between tags, the texts are concatenated with semicolons between them. The presence of an A, IMG or MENU tag is indicated by its tag name between angle brackets, such as <A>, <IMG> and <MENU>.

2.25 Presence of Particular Fields in HTTP Query Strings

The URL in an HTTP request message may contain a query string with one or more fields separated by ampersands. These fields often help to distinguish different types of transactions that are likely to have different performance characteristics, and for that reason a control on NetData's page of Decoding controls can specify query fields that are to help define a transaction type. When a field with a specified name is found in a query string, its value is prefixed to the transaction's category (usually GET or POST). The pop-up help for this control describes variants that generalise the field's value by ignoring numeric values entirely or suppressing digits within the field value.



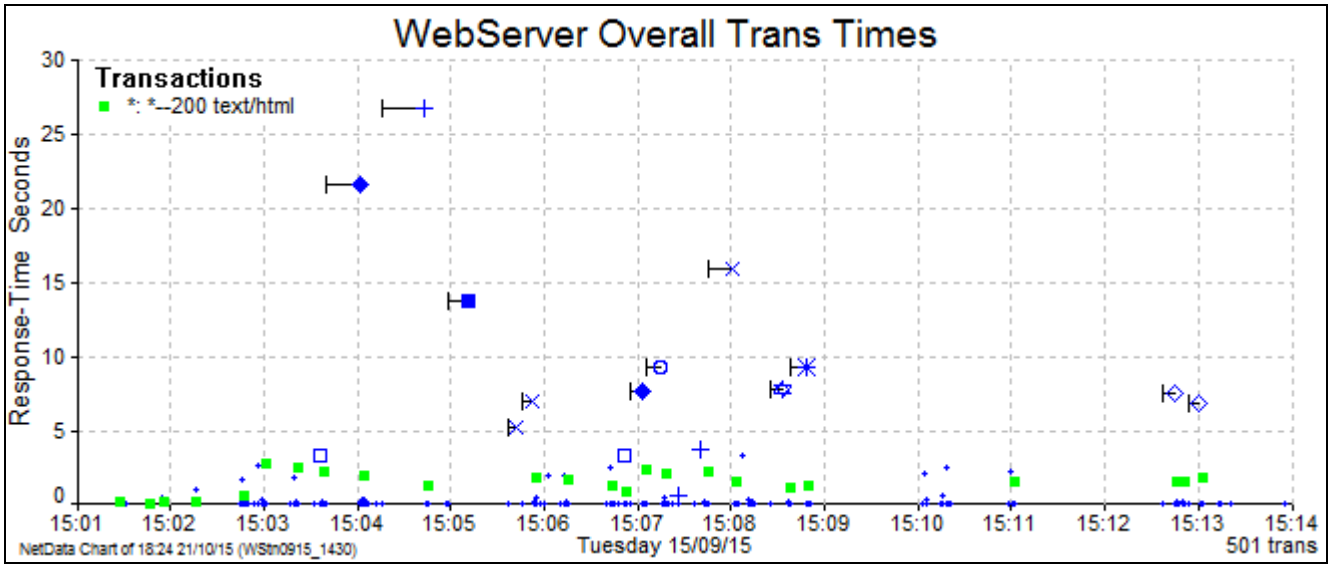
A new variant, specified by an asterisk after the field name, will ignore all field values and simply indicate the presence of the field in the query string, to distinguish transactions with different *types* of query parameters rather than with different *values* of the one parameter.

GET /Login.aspx?ReturnUrl=%2fdefault.aspx&_ImageTag=1jR9AOZ5uodubPTyKQWGSw%3d%3d HTTP/1.1

In this example requests to Login.aspx without a specified ImageTag produce a quite different response compared to those queries that do specify an ImageTag, irrespective of the tag's value. For transactions with a tag NetData would assign the category '_ImageTag GET'.

2.26 Extended HTTP Response Signatures

NetData appends the value of the response content-type header (if it exists) to an HTTP transaction's response signature. It helps to distinguish the different types of Post and Get requests with the same URL, and highlight on the performance chart all the transactions that return, say, an HTML page file.



All the transactions displayed on the above chart have the same URL and most returned data in the JSON format, but those with green highlighting markers produced an HTML file. One of the HTML transactions was chosen in the transaction-type table, assigned a highlighting marker, and chosen again for the 'Highlight Same Response' command.

2.27 Symantec (ex MessageLabs) Proxy Server

Symantec acquired MessageLabs, a provider of online messaging and web security services, in 2008. MessageLabs's cloud-based proxy server that provides URL filtering and other security services uses port 3128. NetData tags HTTP traffic using this port as 'HTTProxy' and tags HTTPS traffic using the same port as 'MsgLabS'.

2.28 Web NTLM Proxy Authentication

NetData decodes NTLM authentication messages that are encoded in base-64 format and conveyed in HTTP headers. The authentication phase is appended to either the transaction category (e.g. CONNECT negotiate) or the status code in the response signature (e.g. 407 challenge). Significant message details appear in the Data column of the transaction table, and the 8-byte challenge word appears in the UserID column to help correlate front-end and backend authentication transactions.

2.29 Port 1080 VPN Detection

NetData recognises several protocols that use port 1080, the proxy port normally associated with the Socks protocol. Two other types of traffic probably relates to different types of virtual private networks (VPNs) because they appear to be encrypted except for very short message headers, and it is not possible to recognise request-response pairs. These protocols are labeled ‘VPNa’ and ‘VPNd’.

2.30 OpenUI Protocol (2012)

NetData decodes the traffic of OpenUI, a platform-independent object-oriented GUI developed by Open Software Associates and popular before the widespread adoption of HTTP browsers. Because OpenUI was also designed to run on any transport protocol it implements its own data-acknowledgement scheme. Data is transferred with a sequence of usually small SendMessage commands, initiated by either the client or the server and each individually acknowledged by a short data packet in the opposite direction.

NetData characterises each SendMessage-DataAck round-trip as a *server* transaction, and aggregates sequences of SendMessage commands into what NetData classes as *group* transactions. The request message of a typical group transaction comprises a sequence of SendMessage commands sent to the server, and its response message comprises a sequence of SendMessage commands sent to the client.

Some user transactions involve many group transactions and there is often a significant pause between group transactions as the workstation software processes one response and prepares the next request. It depends on how the OpenUI client program is written and how deeply objects are structured, but OpenUI programs can demand considerable processing power.

3 Database Management

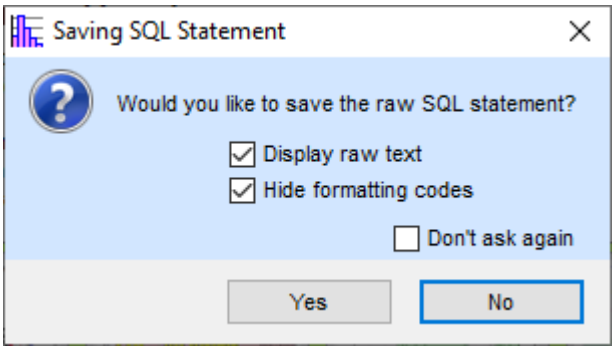
3.1 SQL (Structured Query Language)

NetData decodes the messages of 12 application protocols that convey SQL commands to a database manager: Oracle SQLnet, IBM DB2, Microsoft TDS, Sybase, Interbase, Teradata, IPFX, SAS, DST Systems, MySQL, Dharma, and EMC Documentum.

Every SQL statement found in a message is handled the same way and can be displayed in three different forms. The main verb in the statement—such as Insert, Select, Update and Delete—determines the transaction's category. The statement is then condensed slightly to serve as a request signature, replacing lists of fields with just a field count as in 'Select 44 columns from...', and replacing constants in Where clauses with asterisks. By generalising SQL statements in this way NetData is able to find and plot response times of all the queries of the same type, irrespective of their search targets. The different types of queries and other SQL commands are listed in the Transaction Class Tree.

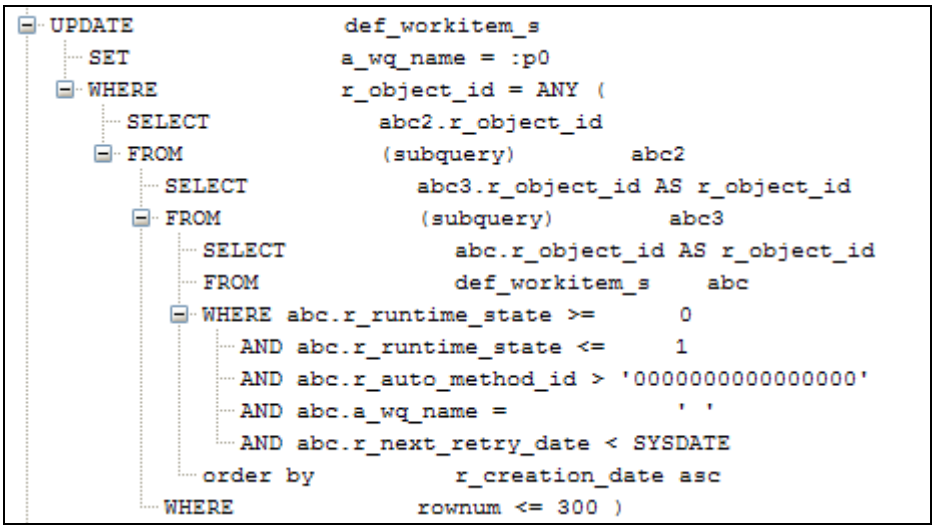
With default settings NetData will extract SQL statements up to 500 KB in length. To extract longer statements change the parameter labelled 'Limit size of decoded messages' in the Miscellaneous group on the Tuning page of controls.

Every SQL statement is recorded in its original form with other transaction details in NetData's database. When the transaction is loaded from the database and a description is requested, NetData offers to display the raw SQL statement in a separate window, and save the complete statement in a text file:



This facility can be used to paste identical copies of the SQL statement into a database management tool, to re-run slow queries and other commands while testing indexes and tuning the database.

The transaction's description includes the generalised version of the statement in the request signature, and also displays the statement in a third form that reveals its syntactic structure. The tree form allows different parts of a statement such as joined tables, sub-queries, and complex Where clauses to be selectively expanded for inspection.



Besides extracting SQL statements from request messages, most NetData decoders of SQL protocols extract the significant input and output variables to display in the Data column of the transaction table. They also reconstruct the data-sets in response messages, allowing them to be browsed in a standard NetData table browser, no matter how large the data-set.

With protocols such as those of Oracle, Microsoft and IBM the database manager may parse a SQL statement with bound variables only once, and execute it many times with different sets of values. A client request refers to the SQL statement by its cursor number or, in the case of DB2, its section number. NetData retains the SQL statement and field names with the cursor's state information, and on subsequent executions reproduces the generalised version of the statement in each transaction's signature. It also recalls previously seen field names when constructing tables to display result data-sets.

The major database managers report application and database errors to the client in considerable detail, often with long explanatory texts. NetData records all errors and warnings in its database as network events in the application category, and also mentions the occurrence of errors in transaction response signatures. Evidence of any application errors will therefore be found in both the transaction-class tree and the events table. They not only indicate problems but also prove a valuable source of diagnostic information, particularly if a transaction failure has been preceded by a trail of database warnings, or other transactions have placed the database in an unexpected state.

A simple and common cause of poor database performance is the need to make many round-trips to fetch successive parts of a large result data-set. The major database managers allow huge volumes of data to be returned in a single round-trip but short cuts or defaults taken by application programmers may fetch only one or a small number of records with each trip. Every Oracle fetch request specifies a maximum number of rows that can be returned, whereas a DB2 open-query request sets two parameters: a query block size, and a maximum number of extra blocks that can be included in a response. A common DB2 round-trip returns a single block of 32767 bytes. The consequence of many round-trips won't be apparent in a development environment which has virtually no loop-delay between workstation and server, but over a WAN they can make an application unusable.

Poor performance caused by many round-trips won't appear on the performance chart as a large response time for a single round-trip, but can be recognised as a long burst of round-trips from the one client, with only a small machine-reaction time between successive trips. Fetch-continuation transactions have the same signature and a large number of them will stand out in a transaction-mix chart and in its underlying table of transaction-type statistics.

If front-end transactions have been captured at the same time as the database transactions, NetData has facilities to find all the database transactions generated by each front-end transaction and form what are called *multi-tier transaction families*. Most families can be found automatically, even if they involve captures from different sniffers, and family groups can be checked and edited on timing charts. Because the transaction timing chart plots transaction bars in different bands for the various front-end and backend connections it can clearly show which bursts of database transactions coincide with and relate to a front-end transaction. The various forms of waterfall chart characterise the composition of transaction families.

The Oracle and DB2 decoders also characterise what NetData calls *group* transactions that measure the elapsed time for all the round-trips involved in a single query. Inefficient queries can be revealed by plotting the response times of group transactions on the performance chart.

The TDS decoder characterises *distributed* transactions as group transactions, measuring the elapsed time for many round-trips from the begin-transaction request through to the ending commit or abort request sent to the database transaction manager. A large response time for such a distributed transaction is not necessarily an indication of inefficiency but possibly a table lock held for a long time and blocking co-incidental queries.

The DB2 decoder performs a similar function, characterising as a single group transaction all the round-trips between and including two Sync Point commands, one to start a unit of work and the other to end a unit of work.

Another cause of unnecessary round-trips and poor performance is poor query design. One symptom is a large number of similar queries, queries that vary only in the search target and which could coalesce into a single query with a different statement of the search criteria. NetData exposes several types of SQL inefficiencies in waterfall charts that dedicate each row to all the queries of the same type, with different types on successive rows. Patterns of transaction bars on these charts reveal the program loops and sub-loops in the client's application thread.

NetData highlights repeated executions of identical queries and sequences of queries with the same search target for different tables. In the latter case round-trips can be eliminated simply by joining tables in a single query. Such a technique can collapse many rows to a single row in the waterfall chart, and the many instances of the query on joined tables can be collapsed to a single query simply by listing all the search targets in an IN clause.

To form a generalised SQL statement to serve in the description of a transaction type, NetData replaces all constants in Where clauses with asterisks. While scanning Where clauses NetData identifies all Like clauses that refer to a bound variable or to a character string that begins with a wildcard. Such clauses generally preclude the use of an index in query execution plans. NetData records these occurrences in the network-event table, and flags the transactions as having an application weakness that may explain abnormally large response times.

3.2 Oracle SQLnet and TNS

Transparent Network Substrate (TNS) forms the lowest layer of the protocol stack in Oracle's networking architecture implemented by middleware products known as SQL*Net (NetData usually drops the asterisk), Oracle Net or Net8 (with the release of Oracle8).

TNS defines a small set of 'packets' that have a simple 8-byte header comprising a length indicator, packet type and two checksums. NetData's names for TNS packets use uppercase and include CONNECT, REFUSED, REDIRECT, RESEND, ACCEPT and MARKER. The first five of these packet types are used when setting up a database session, and the MARKER packet has a special role to interrupt the normal session data flow to handle exceptions. It is likely to be seen when the database manager encounters a more serious problem such as an SQL syntax error. The

server will send a MARKER packet to the client which will reply with another MARKER packet to request details of the error. There can be significant delays in the handling of MARKER packets, and NetData records their occurrence as application-category network events.

Most of the database work, however, is handled by TNS DATA packets, and they are given names determined by their function codes. Every DATA packet begins with two bytes of flags and contains one or more function blocks preceded by a one-byte function code (like a token). There are nearly 20 function codes, and the more common are followed by a sub-function code in a second byte. Cursor-control and data-manipulation commands are conveyed as user-to-server functions (code 3) for which there are more than a hundred sub-functions. The protocol associated with these functions is sometimes regarded as the Two-Task Common (TTC) layer, but strictly speaking TTC handles data conversion and related matters in the first layer above the network layers. Above the TTC layer on the server side, at the top of the protocol stack, is the Oracle Program Interface (OPI). Its peer on the client side is the Oracle Call Interface (OCI).

The formal specification for the contents of DATA packets has not been released by Oracle. There are many different versions of the protocol and up to five different sets of formatting rules for each version. When a client requests a session it indicates a range of versions, usually starting with v300 and extending to the latest version that it can handle. In its response the server selects from this range the latest version that it can handle. The latest version recognised by NetData is v316.

Both the client and server also indicate whether they are little-endian or big-endian, and whether they use 64-bit integers. The joint declarations determine an integer formatting rule that is adopted by both stations: variable-length integers if the machines are different; or their native format if the machines are similar. NetData indicates the formatting rule with one or two letters at the end of the protocol tag: C for Compact formatting with variable-length integers (maybe only with v308 and v310), V for variable length; L or M for Least- or Most-significant byte first; LL or MM for 64-bit machines. NetData refers to a combination of version and formatting rule as a *dialect*.

NetData may not be able to correctly decode all dialects. If NetData encounters a dialect it can't handle please send a sample of the traffic to Measure IT to update the decoder. Even if messages aren't parsed properly, response times will be measured correctly and message contents can be examined by viewing the raw data contents of packets.

3.2.1 Dialect Hints

Unlike most systems the TCP port of an Oracle database service can handle many different dialects, with different clients, simultaneously. If NetData sees each session being opened it will know how to decode the traffic. Otherwise, it may need a dialect hint, and this can be given in three ways:

1. A drop-down menu on the Decoding page of controls sets a default dialect for all SQLnet connections.
2. A particular service can be assigned a default dialect by an entry in NetNames.ini such as

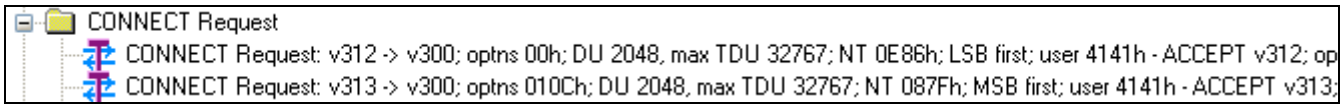
   ```
   OracleSvr = 192.168.78.23:1522=SQLn313V
   ```

3. If some clients of a particular service use different dialects they can be specified with extra entries in netNames.ini. For example, the following lines in NetNames.ini state that clients 10.1.9.11 and 10.1.9.12 use dialect SQLnet308V with port 1522 of an Oracle server, while client 10.1.9.14 uses dialect SQLnet312V:

   ```
   OracleSvr = 192.168.78.23:1522=SQLn308V(10.1.9.11, 10.1.9.12)+
   ```

```
OracleSvr = 192.168.78.23:1522=SQLn312V(10.1.9.14)+
```

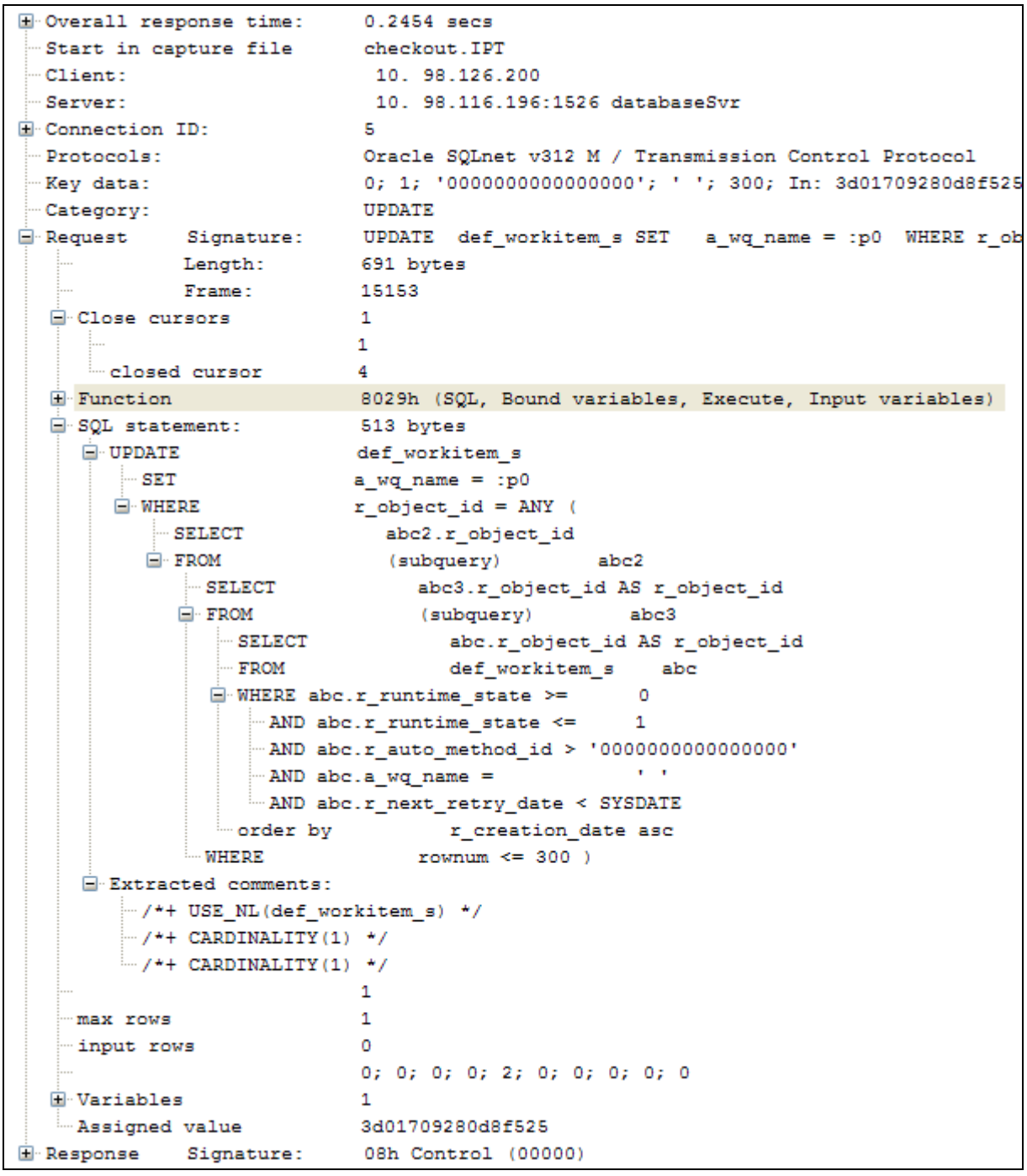
An Oracle SQL session begins with CONNECT Request and Accept messages which negotiate a dialect and are always decoded correctly by NetData, no matter what dialect hint has been given. To find what dialects are in use, look for CONNECT Request transactions under the database server in the transaction-class tree, and see what versions are accepted, and whether client and server send the least- or most-significant byte first. Load all these transactions into the charting module to see what dialects have been negotiated by individual clients.



If NetData finds relevant dialects it records them as dialect hints in Discover.ini, when it has finished decoding the complete capture sequence. In these circumstances NetData offers to re-analyse the capture sequence, to apply what it has learnt to all the traffic and ensure that all messages are decoded as fully as possible.

3.2.2 Composite Commands

SQLnet can convey many commands to the database in a single message and keep the number of round-trips to an absolute minimum, but programmers and tool vendors don't always take advantage of this capability. Inefficient use of round-trips can be a major cause of performance problems, particularly over wide-area networks. The following transaction description, however, illustrates a typical request message that accomplishes several functions:



This message contains two commands. The first closes cursor number 4, and the second is a composite command defined by two bytes of flags at the beginning. The option flags indicate that this command conveys a SQL statement containing a bound variable, provides a value for a variable, and requires the statement to be executed. NetData encounters various undocumented numbers and codes when parsing Oracle messages, and they are simply listed in the description without labels.

The tree displaying the SQL Update statement has been fully expanded to show three sub-queries in its Where clause. This statement included comments that gave execution hints to the database manager, but to clarify the structure NetData extracts all comments and lists them at the end of the statement. The request message ended with a value to be assigned to the bound variable ‘:p0’.

The signature form of the statement also appends the comments at the end of the statement:

```
Request     Signature:     UPDATE  def_workitem_s SET   a_wq_name = :p0
WHERE r_object_id = ANY (SELECT  abc2.r_object_id  FROM  (SELECT
abc3.r_object_id AS r_object_id  FROM  (SELECT  abc.r_object_id AS
r_object_id  FROM  def_workitem_s abc WHERE abc.r_runtime_state >= * AND
abc.r_runtime_state <= * AND abc.r_auto_method_id > * AND abc.a_wq_name = *
AND abc.r_next_retry_date < SYSDATE order by r_creation_date asc ) abc3 WHERE
rownum <= * ) abc2 )/*+ USE_NL(def_workitem_s) *//*+ CARDINALITY(1) *//*+
CARDINALITY(1) */; Close
```

The isolated asterisks in the signature have replaced constants, and those constants are listed in the transaction's key data. The inclusion of an initial cursor-close command is indicated at the end of the signature.

The panel below presents the complete statement in its original form. It is harder to understand because it has no formatting codes but it shows the execution hints in their proper context.

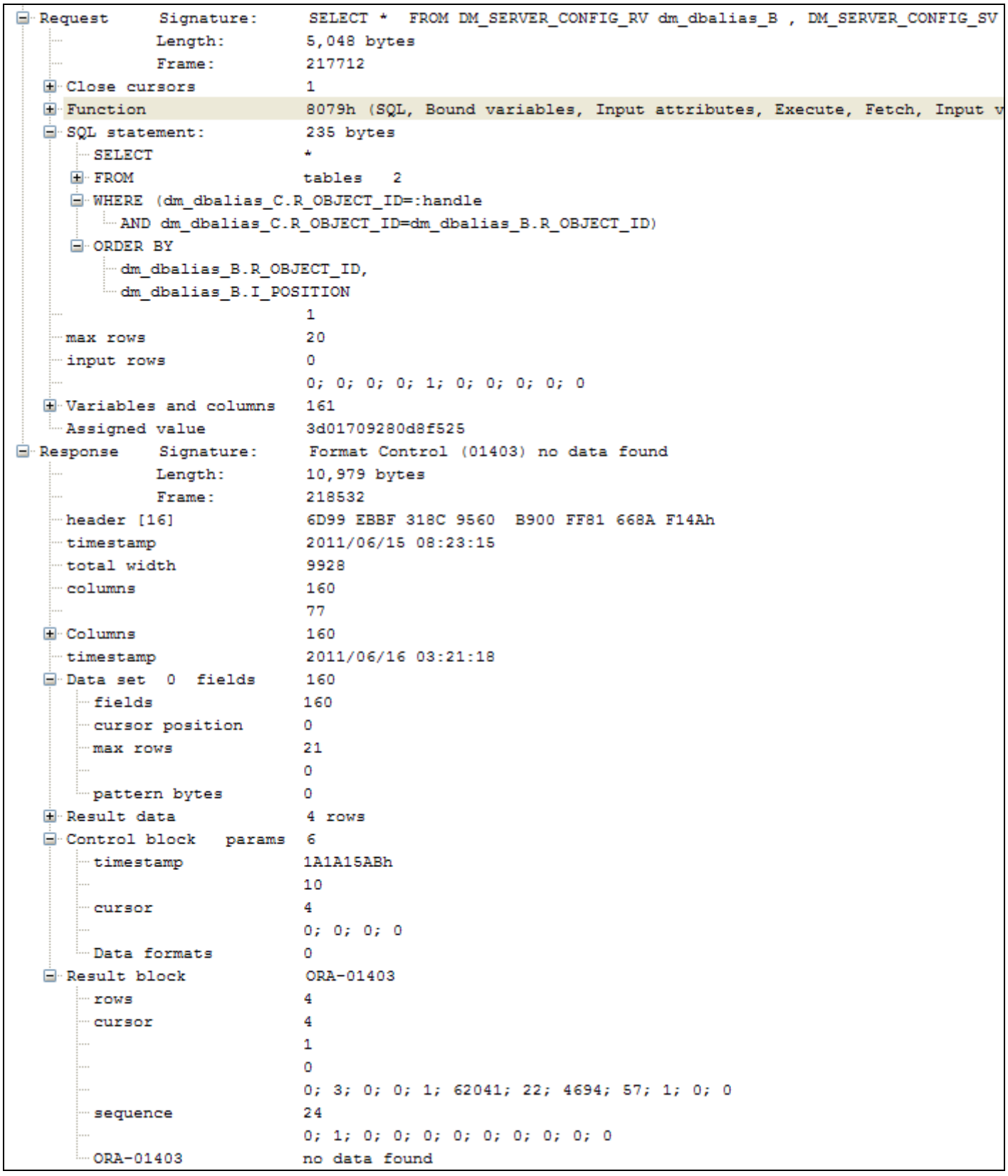
```
UPDATE /*+ USE_NL(def_workitem_s) */ def_workitem_s SET   a_wq_name = :p0
WHERE r_object_id = ANY (SELECT /*+ CARDINALITY(1) */ abc2.r_object_id FROM
(SELECT abc3.r_object_id AS r_object_id FROM  (SELECT /*+ CARDINALITY(1) */
abc.r_object_id AS r_object_id FROM  def_workitem_s abc WHERE
abc.r_runtime_state >= 0 AND abc.r_runtime_state <= 1 AND
abc.r_auto_method_id > '0000000000000000' AND abc.a_wq_name = ' ' AND
abc.r_next_retry_date < SYSDATE order by r_creation_date asc ) abc3 WHERE
rownum <= 300 ) abc2 )
```

NetData recognises both types of SQL comment (-- and /*...*/) and displays them at the end of the statement from which they are extracted.

3.2.3 Response Messages

The following panel describes a typical query transaction. The option flags indicate that this command conveys a SQL statement containing a bound variable, provides column attributes and a value for a variable, requires the statement to be executed and requires results to be fetched. Another parameter limits the number of returned rows to 20.

The response message includes several function blocks in addition to a table that defines each column in the result data-set, and the data-set itself. The result data-set in this example has 4 rows and 160 columns.

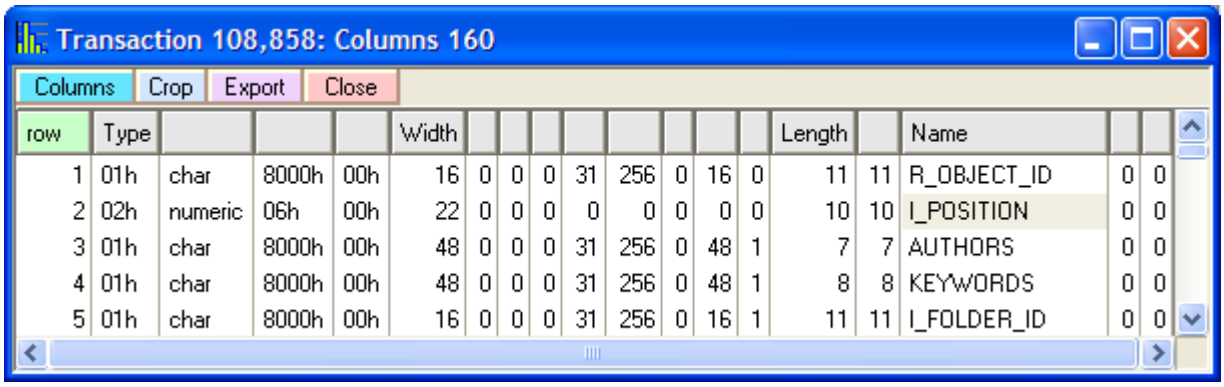


The response message of every transaction involving a SQL statement ends with a Status block that includes a cursor number, a row count, a command sequence number, and an Oracle result code. A non-zero result code is described in text at the end of the block. The most common code is 01403 for ‘no data found’; it doesn’t necessarily mean that no data *was* found, but that there is no more data *to be* found—in other words, the complete result set has been sent to the client.

NetData displays the block of column information as a structured table:

```
Columns                 160
  Type                  Width                                         Length     Name
  ------------------------------------------------------------------------------------------
  01h char     8000h 00h  16   0   0   0  31 256   0  16   0     11  11 R_OBJECT_ID
  02h numeric  06h 00h    22   0   0   0   0   0   0   0   0     10  10 I_POSITION
  01h char     8000h 00h  48   0   0   0  31 256   0  48   1      7   7 AUTHORS
  01h char     8000h 00h  48   0   0   0  31 256   0  48   1      8   8 KEYWORDS
  01h char     8000h 00h  16   0   0   0  31 256   0  16   1     11  11 I_FOLDER_ID
```

Double-clicking the parent item ‘Columns’ produces a new window with a standard NetData table browser that can be sorted, filtered, searched and exported like any other NetData table:

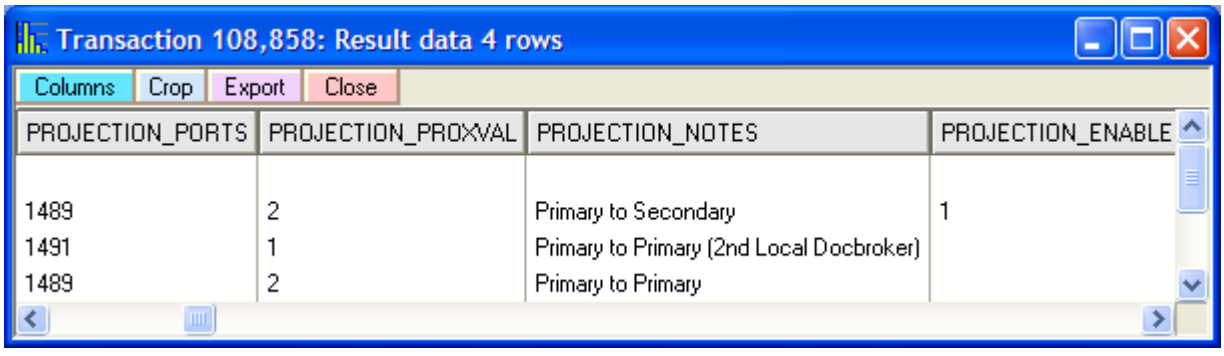


row	Type				Width								Length		Name		
1	01h	char	8000h	00h	16	0	0	0	31	256	0	16	0	11	11	R_OBJECT_ID	0	0
2	02h	numeric	06h	00h	22	0	0	0	0	0	0	0	0	10	10	I_POSITION	0	0
3	01h	char	8000h	00h	48	0	0	0	31	256	0	48	1	7	7	AUTHORS	0	0
4	01h	char	8000h	00h	48	0	0	0	31	256	0	48	1	8	8	KEYWORDS	0	0
5	01h	char	8000h	00h	16	0	0	0	31	256	0	16	1	11	11	I_FOLDER_ID	0	0

The Result data-set is also displayed in the transaction description as a structured table:

```
Result data             4 rows
  R_OBJECT_ID      I_POSITION AUTHORS KEYWORDS I_FOLDER_ID      R_COMPOSITE_ID R_COMPOSITE_LA
  ------------------------------------------------------------------------------------------
  3d01709280d8f525         -4
                           -3
                           -2
                           -1                 0c01709280000106
```

This table has too many columns to be fully displayed in a structured description, but double-clicking its parent item ‘Result data’ also produces a table browser in a window that can be scrolled to any portion of the data-set:

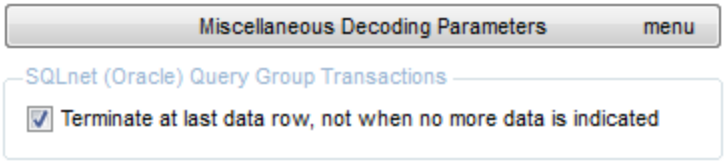


PROJECTION_PORTS	PROJECTION_PROXVAL	PROJECTION_NOTES	PROJECTION_ENABLE
1489	2	Primary to Secondary	1
1491	1	Primary to Primary (2nd Local Docbroker)	
1489	2	Primary to Primary	

3.2.4 Terminating SQLnet Query Group Transactions

Oracle SQLnet queries that involve more than one round-trip to the database are automatically characterised as group transactions that begin with the first round-trip to execute the query and embrace all the additional round-trips to fetch groups of data rows in the result set. The group transaction normally ends with a round-trip that returns the status code 01403 indicating ‘no data found’.

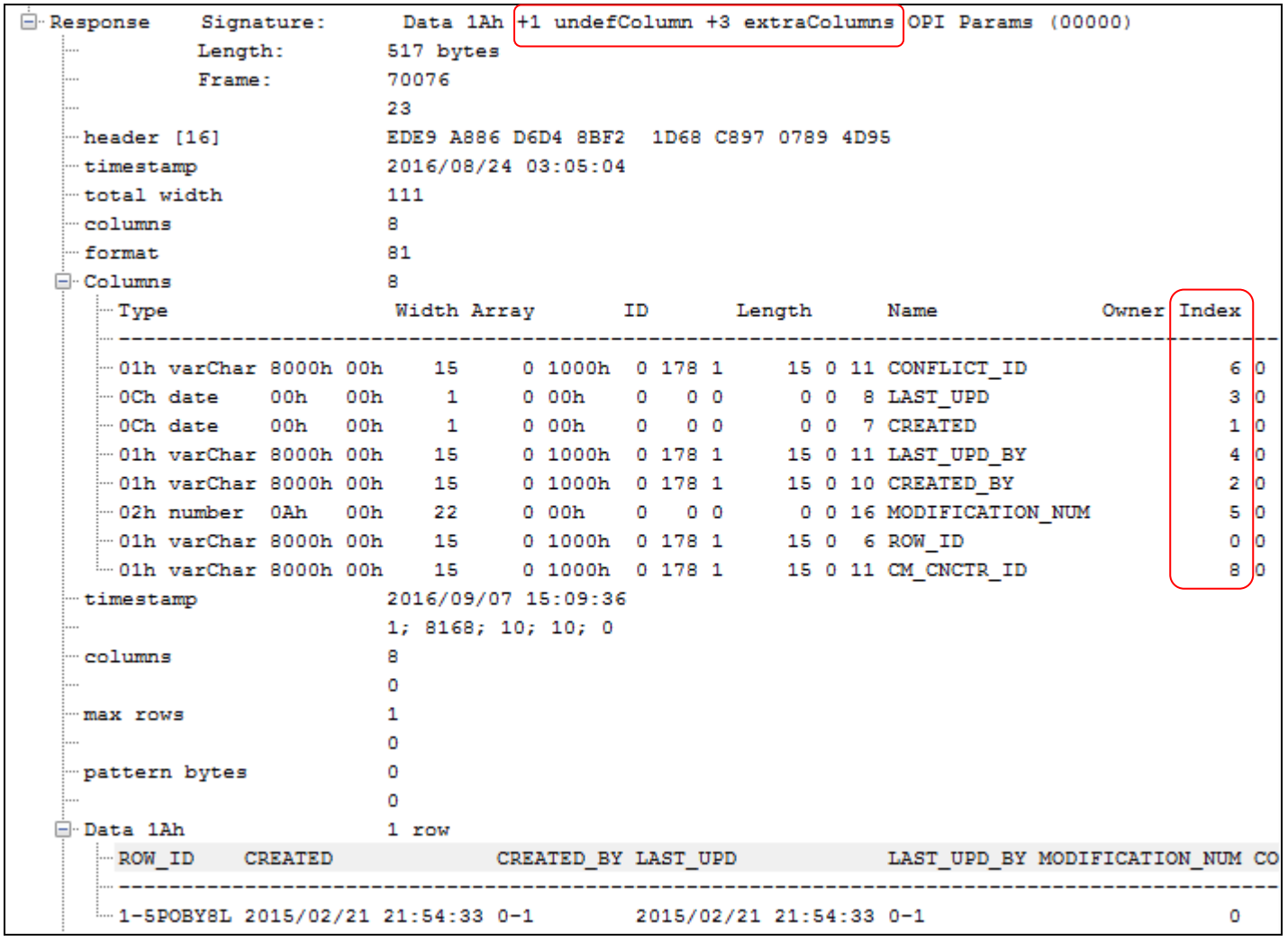
Sometimes a client fetches all the data rows without getting that status code and it is left to a house-keeping function much later to request another row that prompts the terminating status code. To measure a response time that better reflects the delay to the user, a decoding option terminates these group transactions with the last round-trip that returns some data, ignoring the subsequent trip that returns no data.



Even with this optional shortening of a group query its overall response time might overstate the delay to a user. In some cases a user requests only a small set of data rows to populate a table on a screen, and as the user scrolls down the table prompts the database client to fetch successive rows in the result set. The overall query time then includes an arbitrary component of user think time.

3.2.5 Oracle SQLnet Missing Column Definitions

Oracle database messages containing a table of data rows in response to a query usually include also a table of column definitions as in this example:



```
Response     Signature:       Data 1Ah +1 undefColumn +3 extraColumns OPI Params (00000)
             Length:          517 bytes
             Frame:           70076
                              23
header [16]                   EDE9 A886 D6D4 8BF2  1D68 C897 0789 4D95
timestamp                     2016/08/24 03:05:04
total width                   111
columns                       8
format                        81
Columns                       8
  Type                         Width Array       ID       Length      Name              Owner Index
  ---------------------------------------------------------------------------------------------------
  01h varChar 8000h 00h    15      0 1000h  0 178 1     15 0 11 CONFLICT_ID                 6 0
  0Ch date    00h   00h     1      0 00h    0   0 0      0 0  8 LAST_UPD                   3 0
  0Ch date    00h   00h     1      0 00h    0   0 0      0 0  7 CREATED                    1 0
  01h varChar 8000h 00h    15      0 1000h  0 178 1     15 0 11 LAST_UPD_BY               4 0
  01h varChar 8000h 00h    15      0 1000h  0 178 1     15 0 10 CREATED_BY                2 0
  02h number  0Ah   00h    22      0 00h    0   0 0      0 0 16 MODIFICATION_NUM           5 0
  01h varChar 8000h 00h    15      0 1000h  0 178 1     15 0  6 ROW_ID                    0 0
  01h varChar 8000h 00h    15      0 1000h  0 178 1     15 0 11 CM_CNCTR_ID               8 0
timestamp                     2016/09/07 15:09:36
                              1; 8168; 10; 10; 0
columns                       8
                              0
max rows                      1
                              0
pattern bytes                 0
                              0
Data 1Ah                      1 row
  ROW_ID    CREATED             CREATED_BY LAST_UPD            LAST_UPD_BY MODIFICATION_NUM CO
  ---------------------------------------------------------------------------------------------
  1-5POBY8L 2015/02/21 21:54:33 0-1        2015/02/21 21:54:33 0-1                        0
```

Some data formatting rules allow data columns to be output in an order that is different to the order in the definitions table, following instead the order specified by index numbers as in the second-last column of the above definitions table. NetData's transaction description takes these index numbers into account when assigning column headings and interpreting the various data types (as seen in the single data row at the bottom of the above panel, placing dates under headings intended for dates), but this example is unusual in that the definitions table has no entry with index number 7. Nevertheless, the table does define all the columns specified in the query's Select statement.

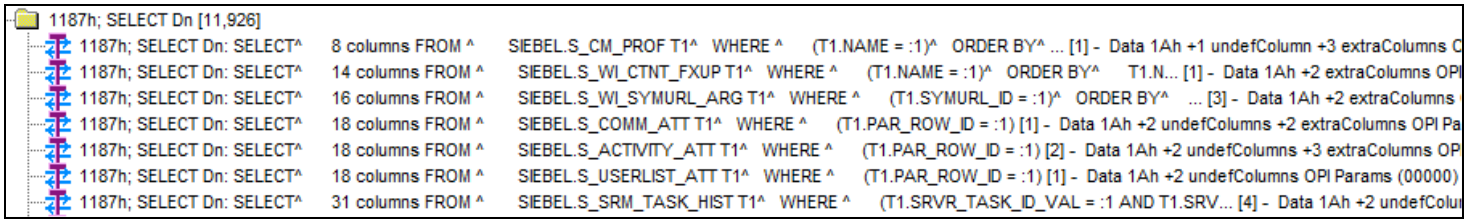
To accommodate the data for the missing index NetData creates a dummy definition that assumes a character string and assigns the column heading '[undefined]', as in the following data-table extract:

MODIFICATION_NUM	CONFLICT_ID	[undefined]	CM_CNCTR_ID	c10	c11	c12
0	0	0-R9NH	0-10MDZ	PAL FARO outbound SMTP Profile	2015/02/21 21:54:50	User

This data table is further unusual in appending an extra three unexpected columns for which NetData has assigned the headings *c10*, *c11* and *c12*. The extra columns are assumed to contain metadata indicating the time and source of the data row.

Transactions with missing column definitions can be found in the transaction-class tree because their response signatures contain phrases specifying the number of extra columns such as

```
Data 1Ah +1 undefColumn +3 extraColumns
```



3.2.6 Different Client Dialects

The pop-up help for the SQLnet-format control on the Decoding page of controls discusses the various dialects and when they are used. It is possible for different clients to choose and operate different SQLnet dialects with the one service (through the one server port).

NetNames.ini allows dialect hints to be specified for different clients of a service. For example, the following lines in NetNames.ini state that clients 10.1.9.11 and 10.1.9.12 use dialect SQLnet308V with port 1522 of an Oracle server, while client 10.1.9.14 uses dialect SQLnet313V:

```
OracleSvr = 192.168.78.23:1522=SQLn308V(10.1.9.11, 10.1.9.12)+
OracleSvr = 192.168.78.23:1522=SQLn313V(10.1.9.14)+
```

3.2.7 Oracle SQLnet Version 314

NetData's Oracle SQLnet decoder handles protocol v314, and has been tested extensively on the two dialects (314V and 314L) that use variable-length and little-endian integers. Among many refinements affecting all dialects NetData extracts from SQL statements the names of bound variables to label their assigned values. They are stored with cursor state information and carried, along with the SQL statement's signature and input and output format definitions, to all new transactions that refer to the same cursor.

The data column of the transaction table presents three types of information: values assigned to bound variables (labelled 'Assign:'); constants in the SQL statement ('In:'); and output values ('Out:') in the response message. They help in finding related and identical transactions.

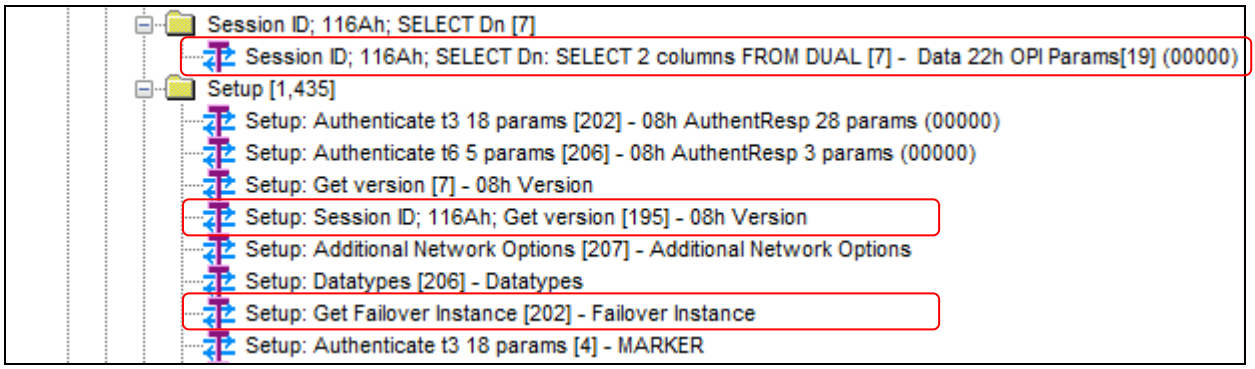
A *group* transaction – one that characterises all the round-trips of an individual query –displays in a single table the complete dataset returned by the query, no matter how many Fetch requests are involved.

3.2.8 Oracle SQLnet Piggyback Functions

Messages from a user to an Oracle database usually contain the details of a request such as a query, and the presence of a request block is signalled by a single-byte token of value 3 that is followed by a sub-function code in a second byte that forms a composite function code such as 035Eh. Request blocks are often preceded by one or more auxiliary function blocks signalled by a token of value 17 that is also followed by a sub-function code. In some unofficial documents the auxiliary function is described as a *piggyback* function. The most common piggyback function has the composite code 1178h and specifies a list of cursors to be closed. Even though NetData isn't able to interpret all the piggyback functions it must parse them successfully to reach the subsequent request block and interpret it successfully.

NetData parses an enlarged set of piggyback functions with codes 1169h (close cursors), 116Ah, 116Bh (session ID), 1178h (close cursors), 1187h (object), and 119Ah (context). It also recognises a request block with code 0374h that is used when setting up a database session and is believed to request details of a server failover instance.

The presence of piggyback functions is indicated in the transaction-class tree because they are mentioned in transaction request signatures either by name or composite code.



3.2.9 Severely Truncated Oracle SQLnet Traffic

If packets have been severely truncated during capture – to less than 200 bytes – the SQLnet decoder relies on reading only the first two bytes of the body of a data message to determine the message's function.

The length indicator in the TNS 10-byte header allows NetData to determine the full length of a message even if it consists of many data blocks, and a generalisation of the message length extends the request and response signatures that define a transaction's type. For example, the following transaction type characterises a common type of query whose request message length is between 600 and 700 bytes and whose response message is 1000 bytes or longer.:

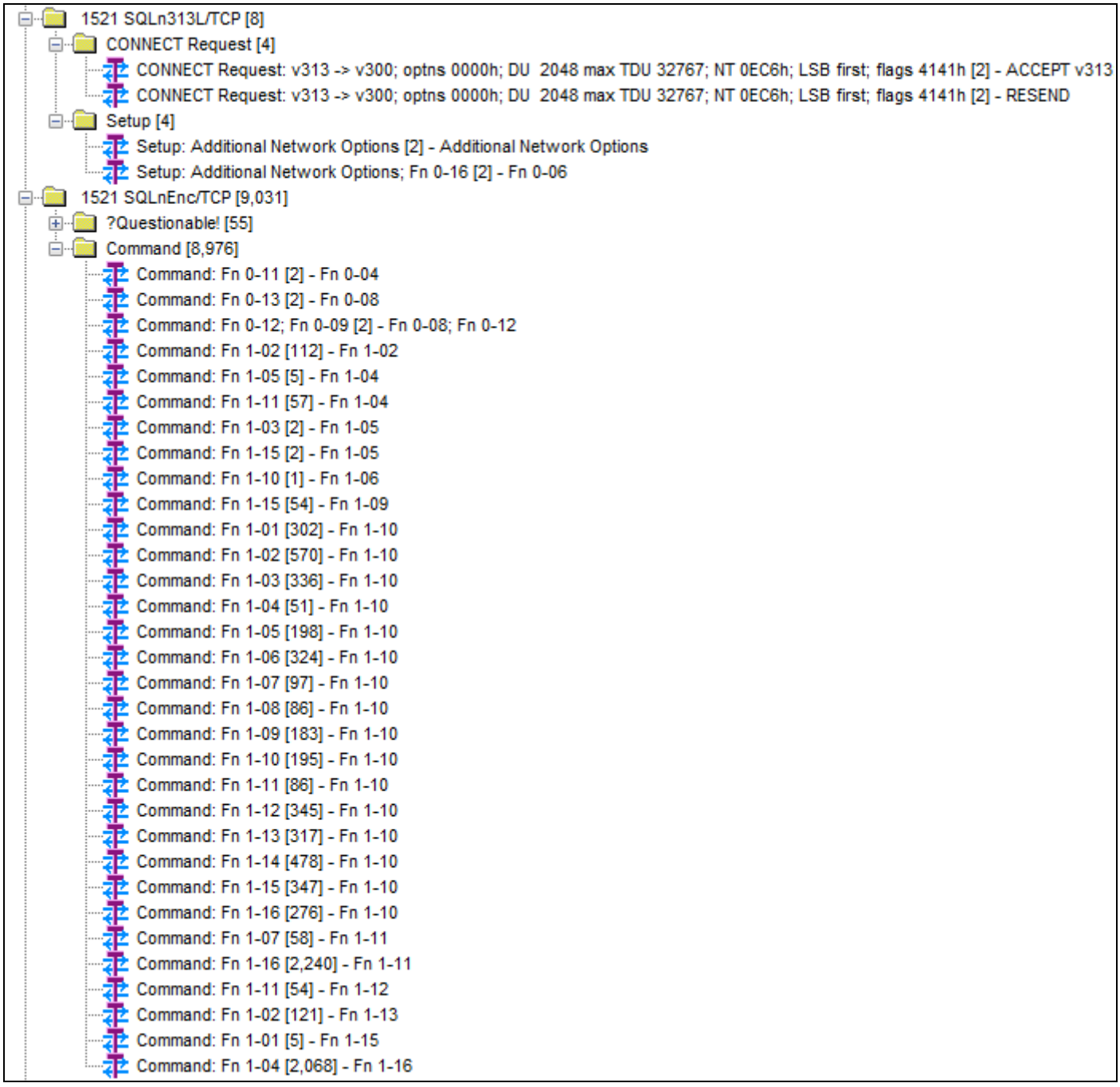
```
Command: 035Eh (Bundled query exec) [6xx]--1017h (metadata) [*]
```

The request and response signatures begin with the hex codes of the first two message bytes and include a brief description in parenthesis

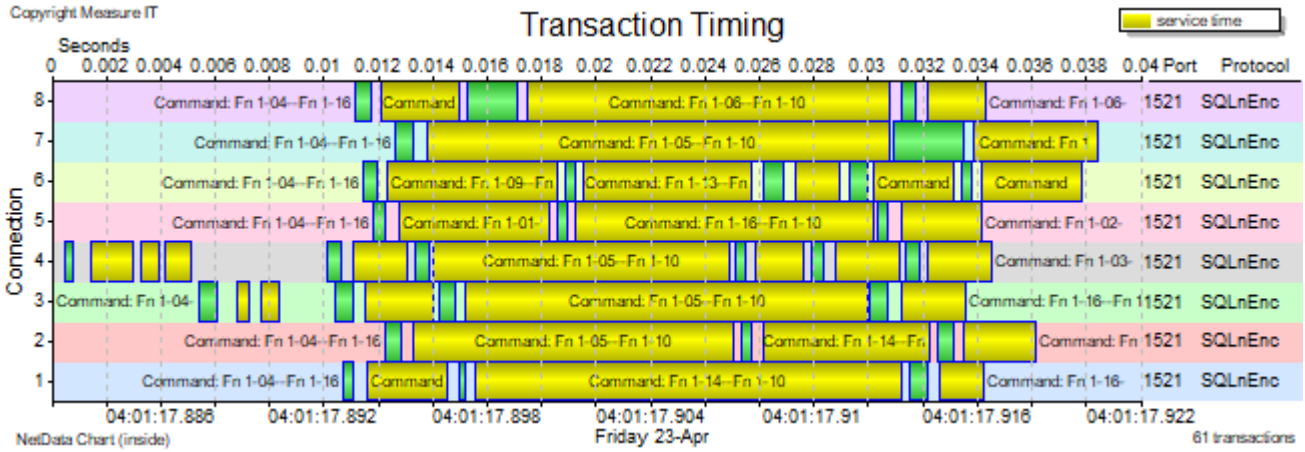
The first byte of a message serves as a token that determines the format of the following information and some tokens such as 03 for a request are qualified by a second byte. Response messages generally comprise a number of information blocks preceded by their defining tokens but when truncation is severe NetData can read only the first token. A similar problem arises for request messages that begin with the token 11h which signifies a *piggyback* operation.

3.2.10 Encrypted Oracle SQLnet Transactions

NetData recognises encrypted Oracle SQLnet data messages. They begin with the standard 10-byte header in clear, and end with two bytes whose codes indicate the function of the message. NetData describes an encrypted message in the form 'Fn x-yy' where x is the value of the last byte (0 or 1) and yy is the value of the second-last byte (1 to 16). The transaction-class tree lists all the different transactions in a capture:



This example suggests that Fn 1-10 is a common response and two transactions – Fn 1-16 -- Fn 1-11 and Fn 1-04 -- Fn 1-16 – are very common. One of these common transactions precedes almost every database query and suggests an inefficiency in the way this particular database was used. SQLnet offers a large range of commands and aggregation options to accomplish many functions in a single round-trip.



Samples of traffic in both clear and cipher should provide clues to the meaning of the function codes in encrypted messages. The initial round-trips of encrypted sessions – Connect Request, Resend, Accept, and the exchange of network options – remain in clear, as the above tree indicates.

Because this decoder depends on reading the last two bytes of every transaction message it is not invoked if packets are truncated during capture.

3.3 Oracle Hyperion SQR Production Reporting

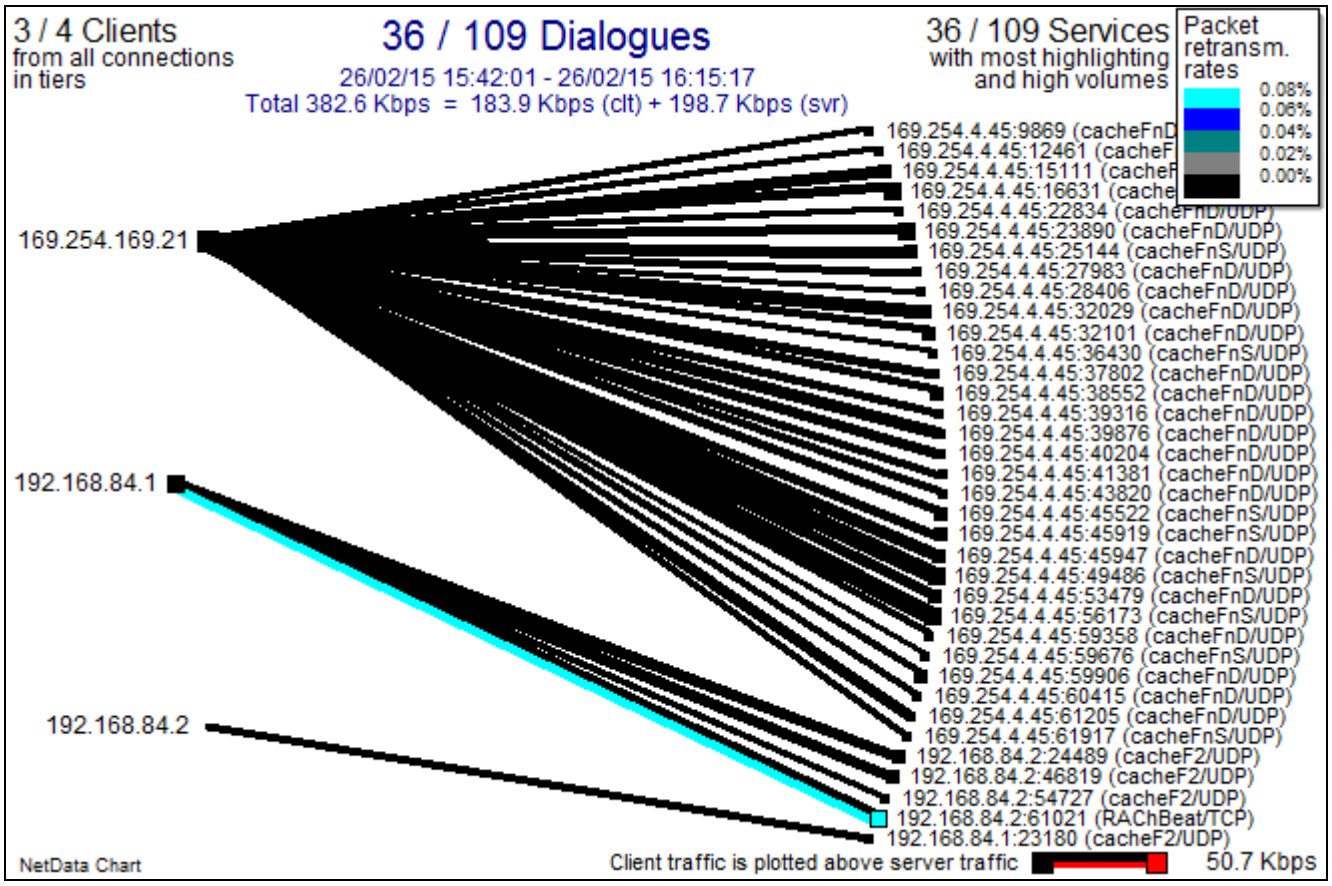
NetData decodes the Service Broker transactions of Oracle's Hyperion SQR (Structured Query Reporter) Production Reporting system. The Hyperion software was created by SQ Software and later acquired by a company which became SQRiBE Technologies and was in turn acquired by Brio Technology and then Hyperion Solutions.

As a pointer to its earlier product name and SQR development language NetData tags the traffic as 'SQRiBEsb'. Its default port is 1497 and it is usually found on a database server.

3.4 Oracle RAC Cache Fusion

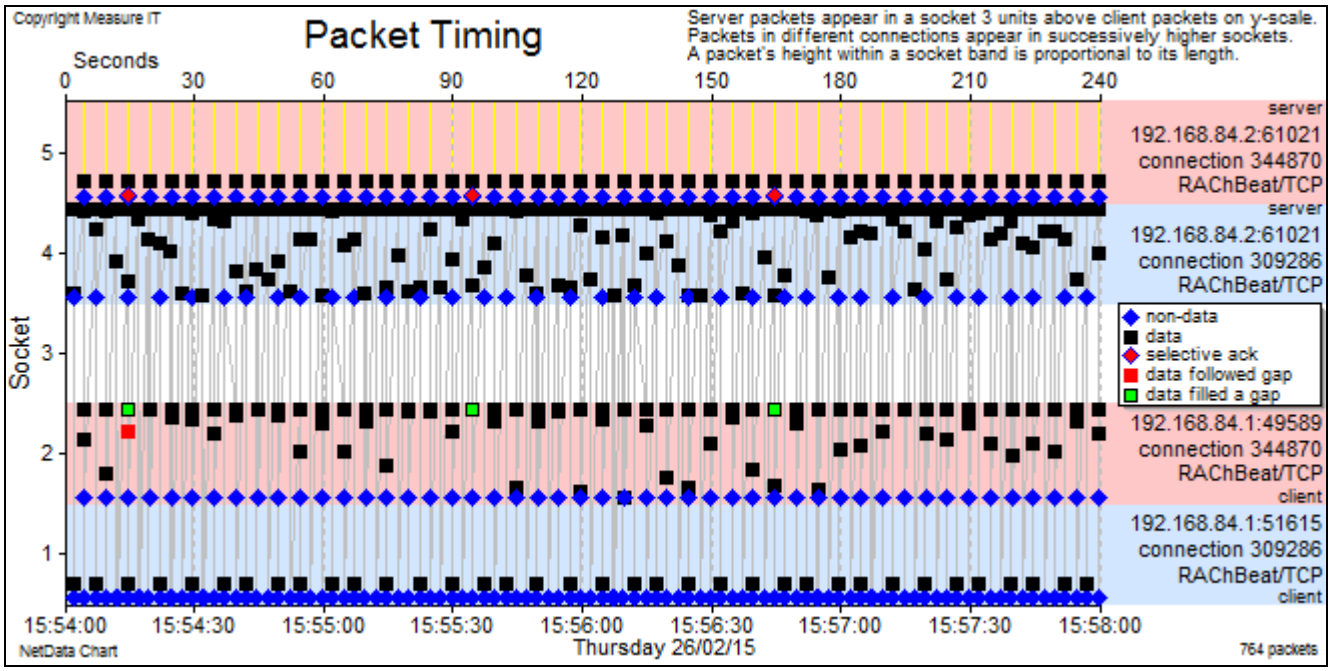
For load sharing and resilience an Oracle database may have two or more instances in a cluster of different machines that use Oracle RAC (Real Application Clusters) to appear as a single high-availability server. They coordinate their caches with RAC Cache Fusion that uses a private interconnecting link between machines.

Because Cache Fusion is required to synchronise the caches in each instance and locks cache segments when the database executes data-manipulation commands, the interconnecting link must have low latency and a high bandwidth to ensure that there are no abnormal transaction delays. NetData recognises three different protocols that appear on an interconnecting link and although the two most critical protocols use UDP, NetData is able to assess the speed and reliability of the interconnect by tracking sequence numbers in the packets, measuring the round-trip times in which messages are acknowledged, noting gaps in sequence numbers, and noting flags in message headers that signify retransmissions. The high-volume protocol has a retransmission timeout that is generally between 12 and 22 msecs.



The traffic captured from the interconnect interface of a database instance includes two different UDP protocols and a TCP protocol that employs a pair of TCP connections between database instances. In this system all interconnect streams were prone to occasional packet losses but the dialogue chart highlights loss and retransmission statistics only for TCP traffic.

The TCP connections exchange status information at 5-second intervals.



Most of the sequence gaps were filled in less than a millisecond and indicate arrivals out of order, but others were filled after a retransmission timeout of 200 ms.

3.5 Microsoft SQL Server TDS

SQL Server manipulates data according to SQL statements that are either submitted directly in a command from the client (in a TDS message type 2), or are contained within a stored procedure called by the client (in a message type 4). The transaction described below executes a stored procedure, not in response to a Call request but with an Exec SQL statement submitted as a command (message type 2).

```
Overall response time:      0.8496 secs
Start in capture file       MIT31101115138.PKT
Client:                      10. 27. 30. 74       SQL2
Server:                      10. 27. 30. 77:1437 SQL_VS
Connection ID:              5,967,419
Protocols:                  Tabular Data Stream version 7
User ID:                    SPID  112
Key data:                   3081; N'2005-10-31'; N'9999-12-31'
Category:                   EXEC
Request      Signature:     exec Firm_ExPremiumSports_BettingAvailable 3 params
             Length:        226 bytes
             Frame:         11924
  Command:
    exec Firm_ExPremiumSports_BettingAvailable parameters 3
      parameter    value
      ----------------------------
      @Lcid =      3081,
      @FromDate =  N'2005-10-31',
      @ToDate =    N'9999-12-31'\^
Response     Signature:     OK
             Length:        59,269 bytes
             Frame:         14810
  ROW FORMAT v7 columns     4
  ORDER-BY columns          {4}
  ROW DATA          rows    12
  IN-PROCED COMPLETN status blocks:1
  ROW FORMAT v7 columns     20
  ROW DATA          rows    56
  IN-PROCED COMPLETN status blocks:1
  RETURNED STATUS:          0
  PROCED COMPLETN status: 00h                E0h                (1)
```

The description lists the names and values of the procedure's three parameters. NetData treats these values as significant data that might help to relate the server transactions of a particular user or user transaction. It records the values in the transaction's field of key data, and when the transaction is loaded from the database these values appear in the Data column of the transaction table:

	Trn Key	Request Strt	Resp End	Type	Description	End Rsp	Data
■	2505	11:51:42.4608	11:51:42.4613	TDS-7/TCP	SET: SET STATISTICS *--OK	0.0005	PROFILE ON
◆	3140	11:51:42.4672	11:51:43.3168	TDS-7/TCP	EXEC: exec Firm_ExPremiumSpo...	0.8496	3081; N'2005-10-31'; N'9999-12-31'
■	3156	11:51:43.3259	11:51:43.3262	TDS-7/TCP	SET: SET STATISTICS *--OK	0.0004	PROFILE OFF

The table also shows that this particular transaction was sandwiched between two Set statements. They turned on the database profiler while the stored procedure was executed and indicate that the database administrator was investigating the execution plan for the query in the procedure. Although the resulting traffic is unlikely to be seen in a production network it illustrates some NetData features that can be used in many situations.

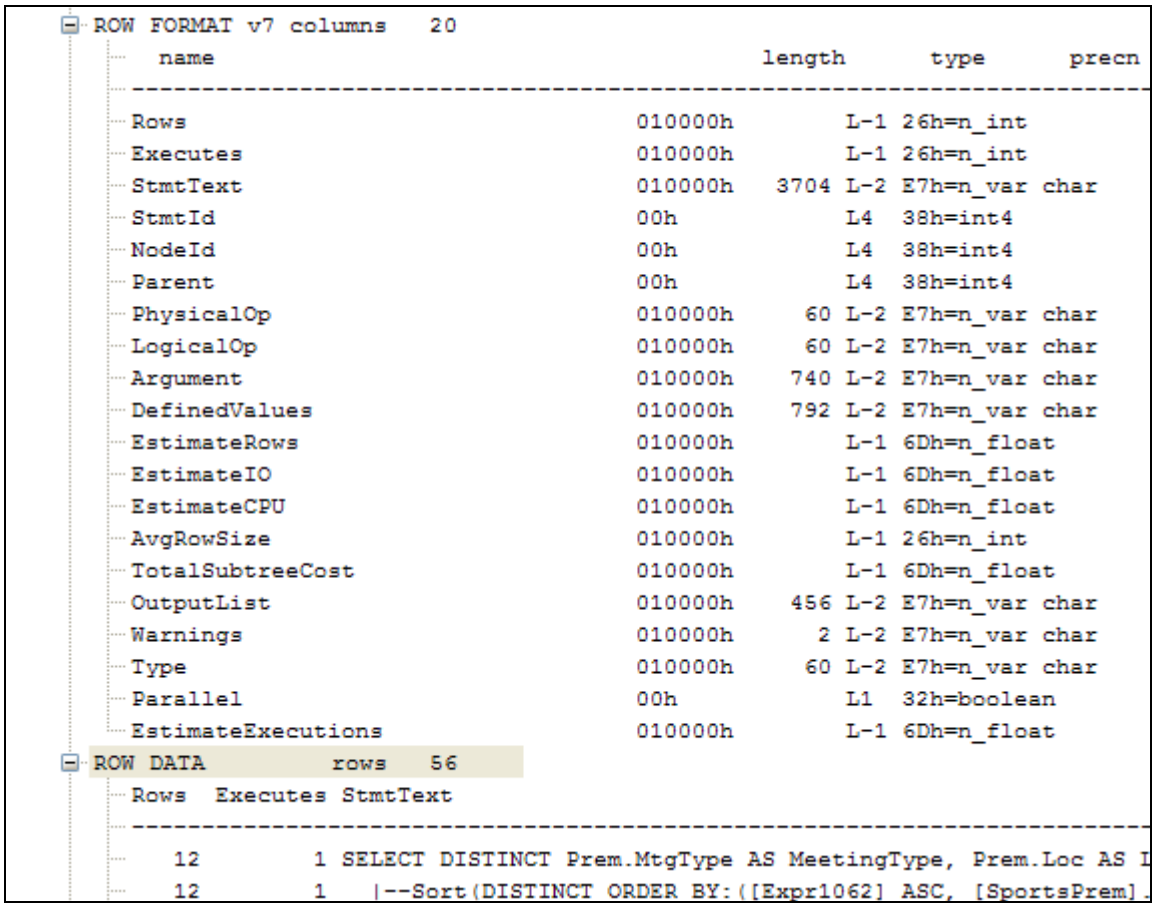
The transaction's response message is typical of a response to one or more queries. Every response comprises a stream of information blocks, each preceded by a defining one-byte token. This response had eight tokens and included two data-sets.

Every data-set table has two tokens, one *row-format* defining the columns in the table and the other conveying the table's *row-data*. There is a status block to report the execution of every *in-procedure* SQL command, and a *procedure-completion* status block for the command as a whole.

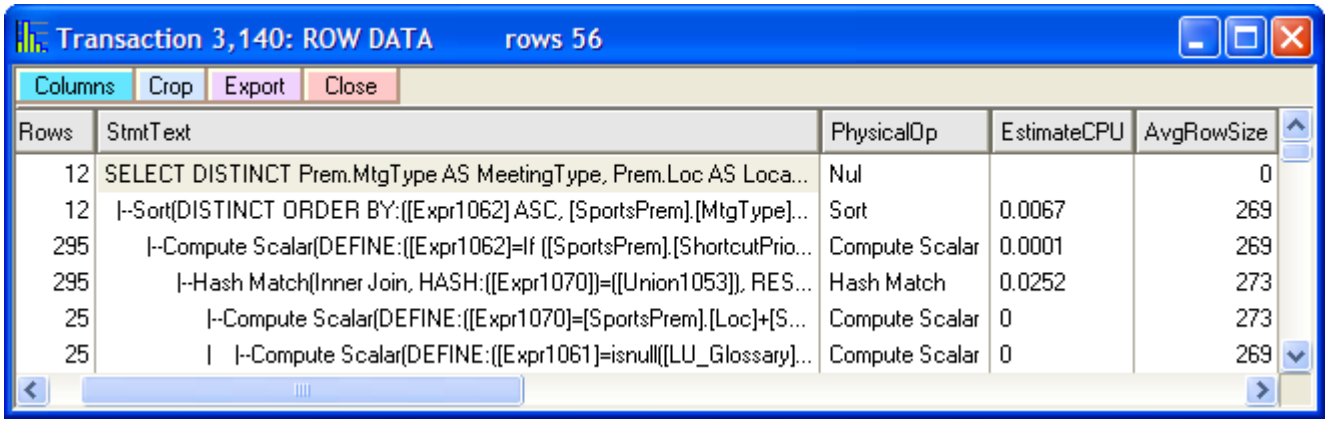
Both the row format and the row data of a result data-set are displayed as structured tables:

```
Response     Signature:      OK
             Length:         59,269 bytes
             Frame:          14810
  ROW FORMAT v7 columns    4
     name                                          length      type      precn
    ------------------------------------------------------------------------
    MeetingType                          080000h      2 L-2 EFh=n_char
    LocationPrem                         080000h      2 L-2 EFh=n_char
    Description                          080000h    500 L-2 E7h=n_var char
    ShortCutPriority                     010000h        L-1 26h=n_int
  ORDER-BY columns         {4}
  ROW DATA          rows   12
    MeetingType LocationPrem Description       ShortCutPriority
    ------------------------------------------------------------
              1            0 RUGBY LEAGUE                     1
              8            0 SOCCER                           2
              7            0 TENNIS                           3
              3            0 GOLF                             4
              9            0 BASKETBALL                       5
              0            5 SPECIAL EVENTS                   6
              2            0 CRICKET                          7
              0            3 BASEBALL                         8
              0            2 BOXING                          10
              6            0 AMERICAN FOOTBALL               12
              4            0 AFL                             13
              0            4 ICE HOCKEY                      14
  IN-PROCED COMPLETN status blocks:1
  ROW FORMAT v7 columns    20
  ROW DATA          rows   56
  IN-PROCED COMPLETN status blocks:1
  RETURNED STATUS:         0
  PROCED COMPLETN status: 00h              E0h              (1)
```

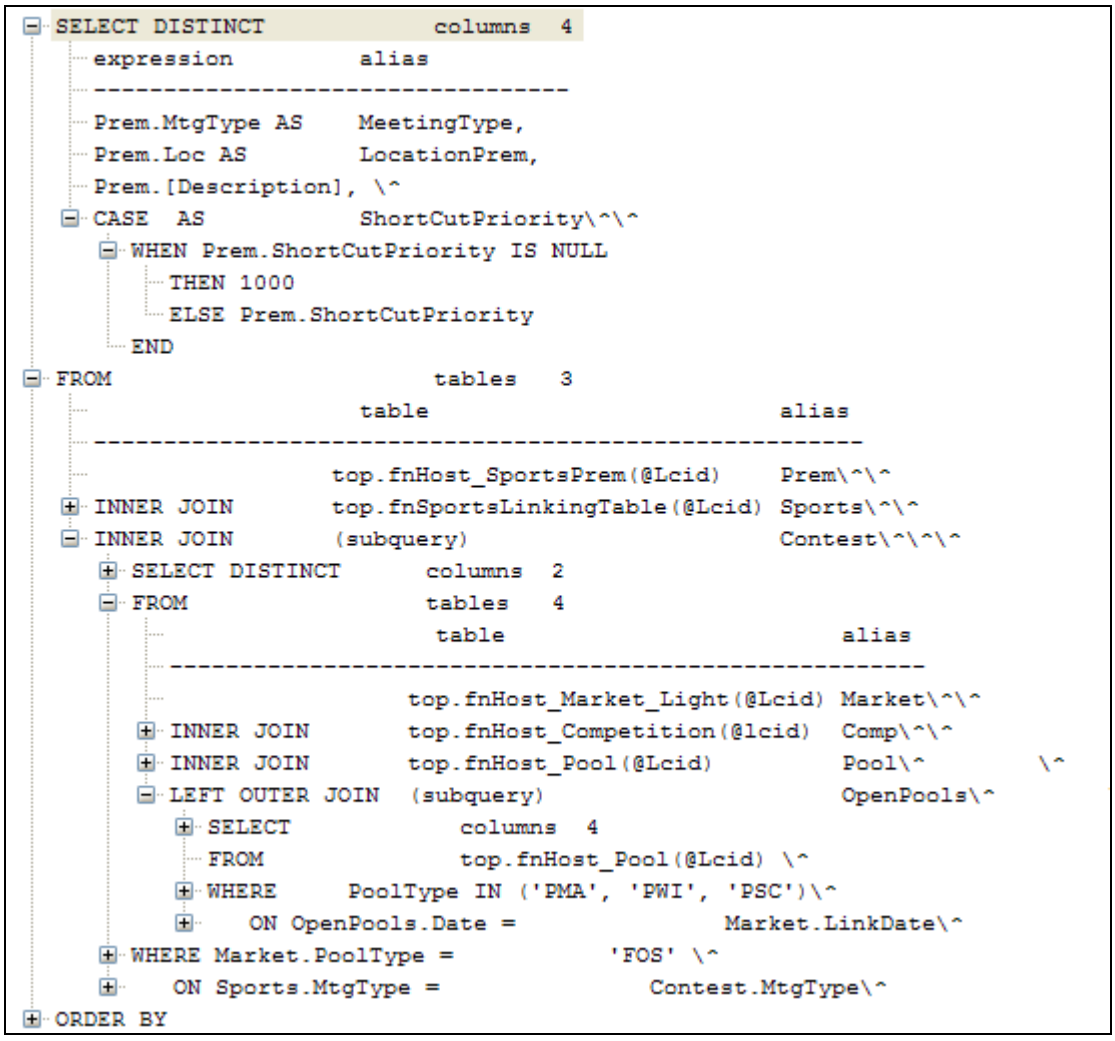
The second data-set in this response was output by the profiler and with 20 fields describes each step in the query's execution:



To view all the data from the profiler, double-click the table's parent branch ('ROW DATA') in the tree, or right-click the parent branch and choose 'Browse Selected Item'. NetData then displays the table in a separate window that can be scrolled to any portion:



The first element in the Stmt Text column contains the complete Select statement of the procedure's query. As in any NetData table browser the contents of a particular cell can be displayed in a separate window simply by double-clicking the cell. In this case NetData detects a SQL statement in the cell and offers to display the statement either in its original (raw) format, or as a structured SQL statement:

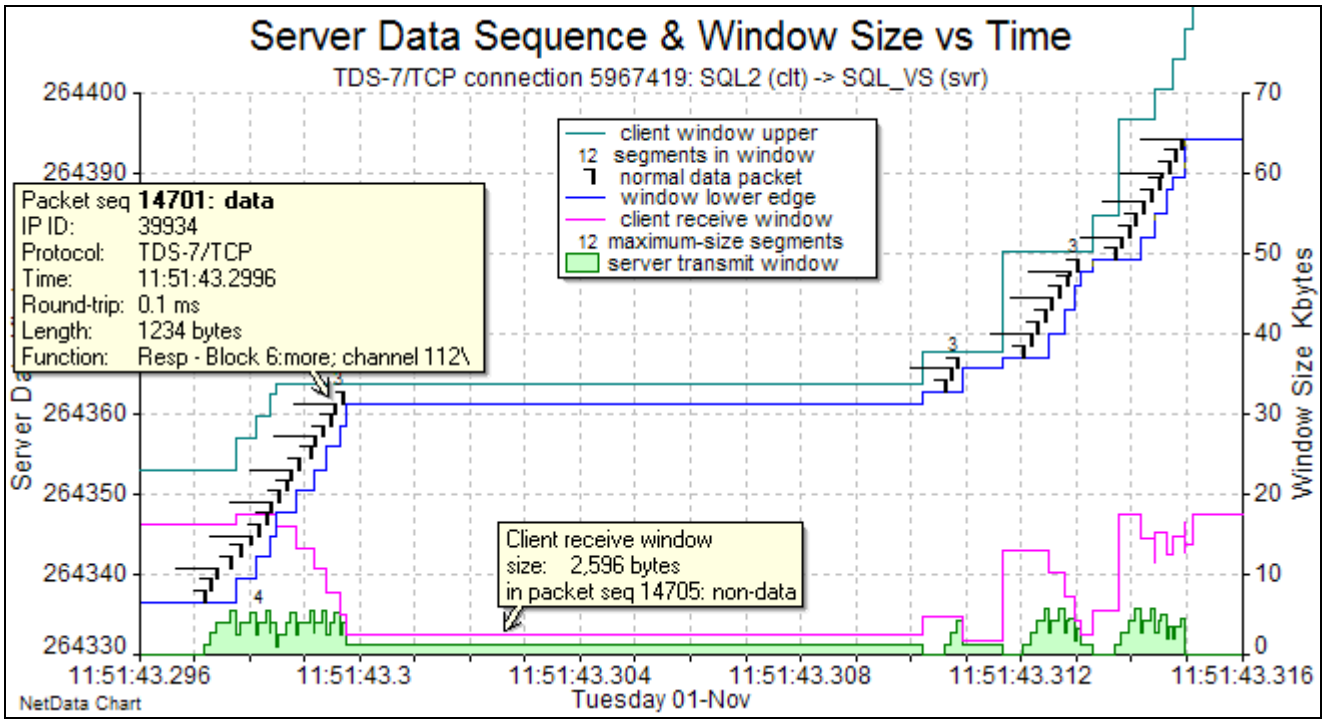


This tree of the structured statement has been expanded only partially to reveal two sub-queries and two sets of joined tables. It also shows a Case statement in the query's list of selected columns.

3.5.1 Progressive Responses

As SQL Server works through a sequence of SQL statements in the one command or procedure it releases progressively any relevant data-sets and in-procedure status blocks for transmission to the client. At the socket interface SQL Server assembles the output into send buffers referred to as 'packets', and only full packets are passed to the transport layer. When a database session is opened on a new connection its packet size is usually set to 4 KB.

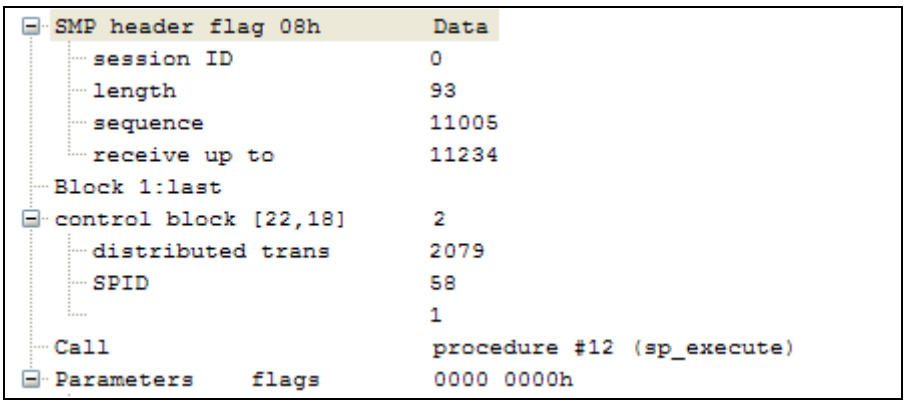
If there is a pause in the transmission of a response message there are several possible explanations: the client may have closed its receive window; there may have been a retransmission timeout after a packet was lost; or the server may have been busy preparing the next part of the message. The data-flow chart should identify the cause. If the delay was in the server it should appear only after the transmission of a complete application block or TDS packet.



In the response message of the profiled query there was a pause of only 10 ms and it didn't occur at the end of a block (indicated by a long horizontal tick at the top of a packet strip). This pause had two contributors. The client was slow in removing data from its receive buffer and consequently slow in opening its receive window. However, the primary cause of the pause was a delayed ack. The receive window had closed to less than two segments with one packet remaining unacknowledged, and no more data could be sent until the client issued its delayed ack. A better TCP driver would not delay an ack if the window was effectively closed as in this situation.

3.5.2 Session Multiplex Protocol

SQL Server has a Multiple Active Result Set (MARS) feature that supports a protocol (Session Multiplex Protocol, SMP) for multiplexing many concurrent TDS transactions on the one connection. Between the TCP and TDS headers it inserts a 16-byte header that includes a session ID, a message sequence number, and another sequence number that controls data flow in the reverse direction. In other words, this protocol uses a message-oriented, sliding-window flow-control system. NetData decodes the SMP header and derives a secondary connection ID from the session ID. Descriptions of SMP headers are recorded in the Contents column of the packets table, with a TDS message description if relevant:

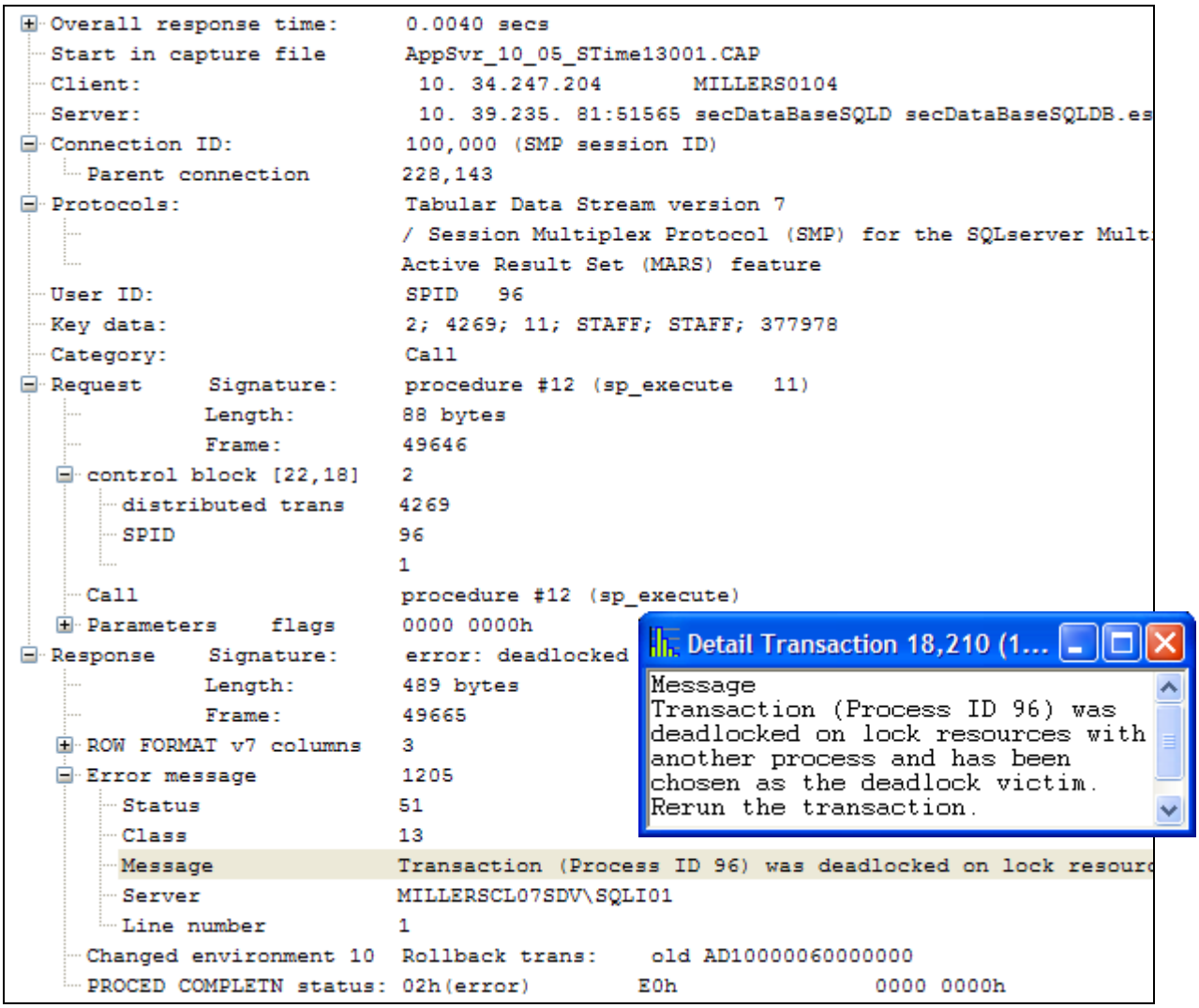


The TDS/SMP decoder allows a mixture of TCP connections through the one database port—some connections using SMP and some not. Multiplexed transactions and the records of their multiplexed sessions (described by NetData as secondary connections) have the type 'TDS-7/SMP'. The connection ID assigned to a secondary connection is the SMP session ID offset by 100,000.

A multiplexed TCP connection will have two or more connection records: one for the TCP connection itself (type SMP/TCP), to which is associated the initial TDS authentication and session establishment transactions; and one for each SMP session (type TDS-7/SMP), to which is associated all the TDS transactions handled by that multiplexed session. Even if all types of connection records are loaded into the charting module, a graph of concurrent connections counts only the TCP (primary) connections, not the secondary connections. As with any multiplexing protocol, the timing chart allows multiplexed transactions and their packets to be split into separate bands representing the secondary connections

3.5.3 Error Messages

If SQL Server encounters a problem when parsing or executing a SQL statement it returns one or more error messages in the transaction's response message. The following transaction description includes a common error message.



When displayed in a tree structure the text of this error message is too long for the display panel, but double-clicking on the item pops up a separate window that is able to display the item's full contents.

A deadlock arises when two transactions both need to acquire the same two or more database locks but acquire them in a different order: one transaction holds lock A and waits for lock B to be released, while the other transaction holds lock B and waits for A. The database manager detects the deadlock and aborts one of the transactions. A deadlock is not serious because the client is normally programmed to repeat the request, but it can introduce a significant delay. Deadlocks would be avoided if all procedures were programmed to acquire locks in a strict sequence.

3.5.4 Cancelled Transactions

There are many reasons why a client might cancel a database transaction, including a timeout while waiting for a response. A SQL Server client cancels a request by sending a special request message called a Non-Expedited Attention Request, or *Attention* for short. NetData terminates the transaction in progress, recording it with a response signature that indicates its cancellation.

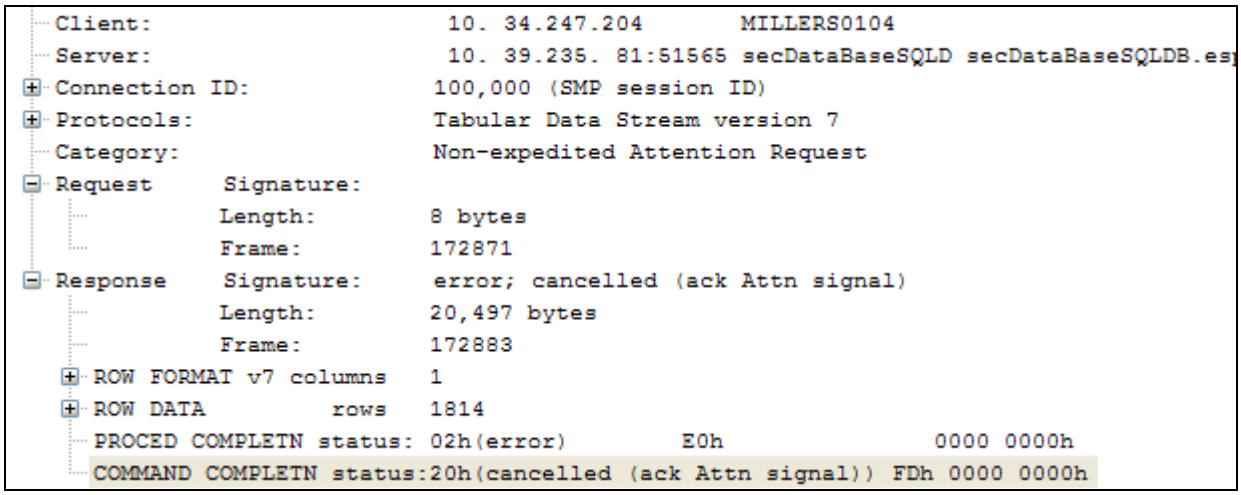
The client is obliged to continue receiving packets until it receives an attention acknowledgement. NetData similarly catches any responses to the cancelled transaction, recording them with the Attention transaction, until it records the attention acknowledgement.

	Trn Key	Request Strt	Description	End Rsp	Data
•	56627	13:20:10.0902	Call procedure #13 (sp_prepexec) SELECT: SELECT se_market_...	0.0030	2; 2105; 0; @
◆	56628	13:20:10.0932	Non-expedited Attention Request: --error; cancelled (ack Attn signal)	0.0010	
•	56630	13:20:10.0962	Dist Trans Control: rollback transaction--OK	0.0010	2; 2105

In this example the client cancelled a query almost immediately on receiving the start of the query's response message, and followed the cancellation with a rollback request for the distributed transaction that began with the cancelled query.

The response signature in the following description of the cancelled transaction indicates that the server had begun the response message before it was cancelled.

```
⊞ Overall response time:     0.0030 secs
  Start in capture file      AppSvr_10_05_STime13001.CAP
  Client:                     10. 34.247.204       MILLERS0104
  Server:                     10. 39.235. 81:51565 secDataBaseSQLD secDataBaseSQLDB.es
⊞ Connection ID:             100,000 (SMP session ID)
⊞ Protocols:                 Tabular Data Stream version 7
  Key data:                  2; 2105; 0; @P1 varchar(8),@P2 datetime; ASX; 04/10/10
  Category:                  Call procedure #13 (sp_prepexec) SELECT
⊟ Request     Signature:     SELECT se_market_history.market_date  FROM se_market_hist
            Length:          467 bytes
            Frame:           172854
  ⊟ control block [22,18]    2
      distributed trans      2105
      SPID                   59
                             1
    Call                     procedure #13 (sp_prepexec)
  ⊞ Parameters     flags     0000 0001h
  Response     Signature:    > incomp; cancelled
```

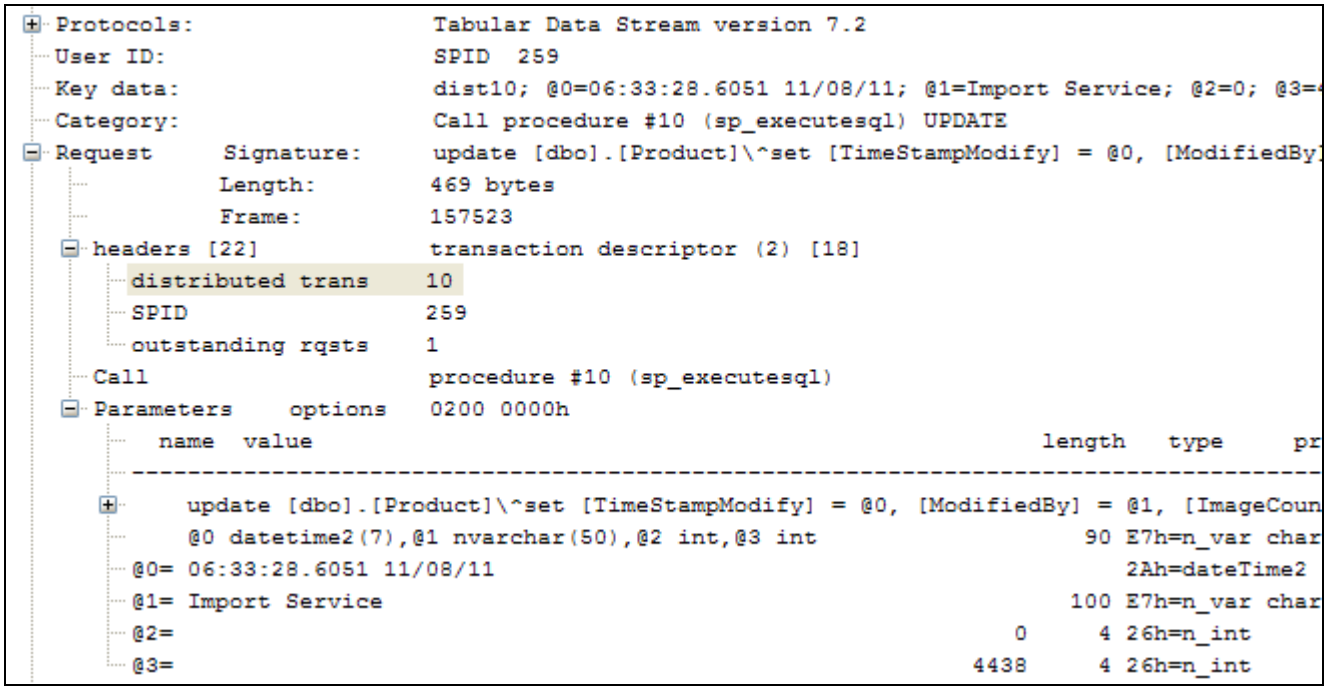


NetData's record of the Attention (cancellation) transaction collected the partial response to the cancelled query; a procedure-completion error indication that terminated the cancelled response; and the attention acknowledgement.

3.5.5 Distributed Transactions

Transactions that involve multiple databases are coordinated by a Windows service called the Distributed Transaction Controller (DTC), and TDS reserves message-type 14 for coordination requests sent to the transaction manager associated with a database. NetData not only records the individual round-trips sent to the transaction manager but also characterises complete distributed transactions that start with a begin-transaction request and end with an abort, roll-back or commit request sent to the transaction manager. NetData classes these transactions as *group* transactions and refers to them as distributed transactions even if they involve only one database.

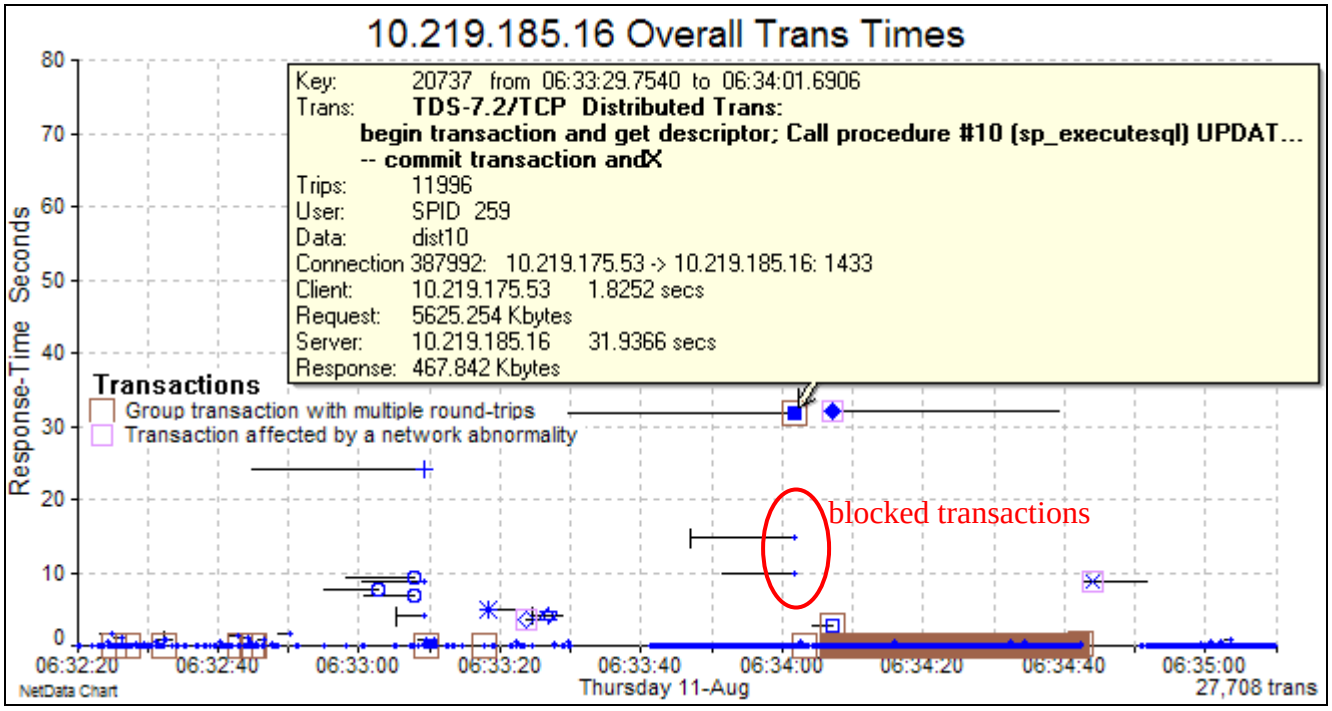
If a SQL command or procedure call forms part of a distributed transaction its transaction descriptor will be specified in an 18-byte header preceding the command:



This SQL Update command was conveyed as the first parameter to the standard stored procedure number 10 which executes any given command. The second parameter defines the types of command variables, and subsequent parameters convey values for the variables.

This request header specified the descriptor for a distributed transaction. The descriptor is normally in two parts: a transaction serial number (10) and the session's process identifier (SPID = 259). Because NetData always records the serial number with the prefix 'dist' as the first item in the transaction's list of key data, components of distributed transactions can be readily found and sorted in the transaction table.

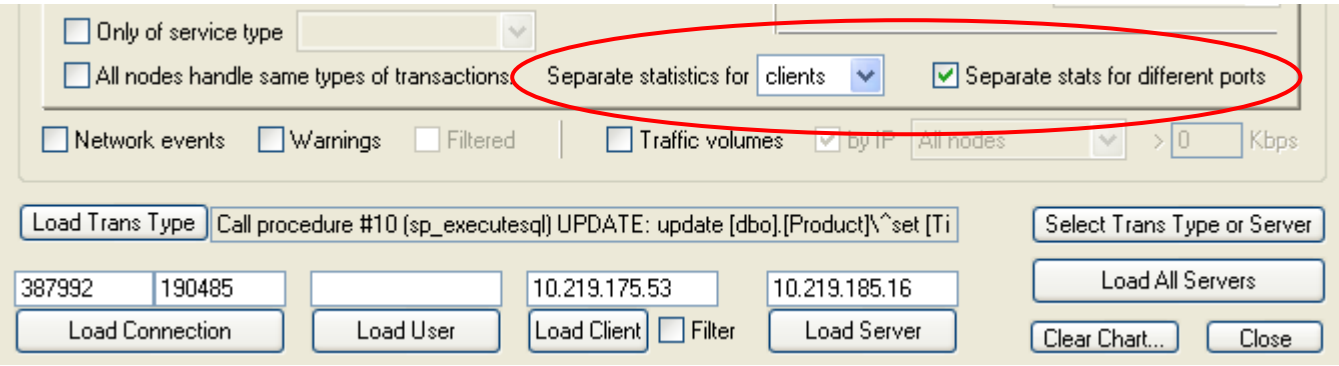
The response-time markers for distributed transactions, as for all group transactions, are enclosed in brown squares on the performance chart:

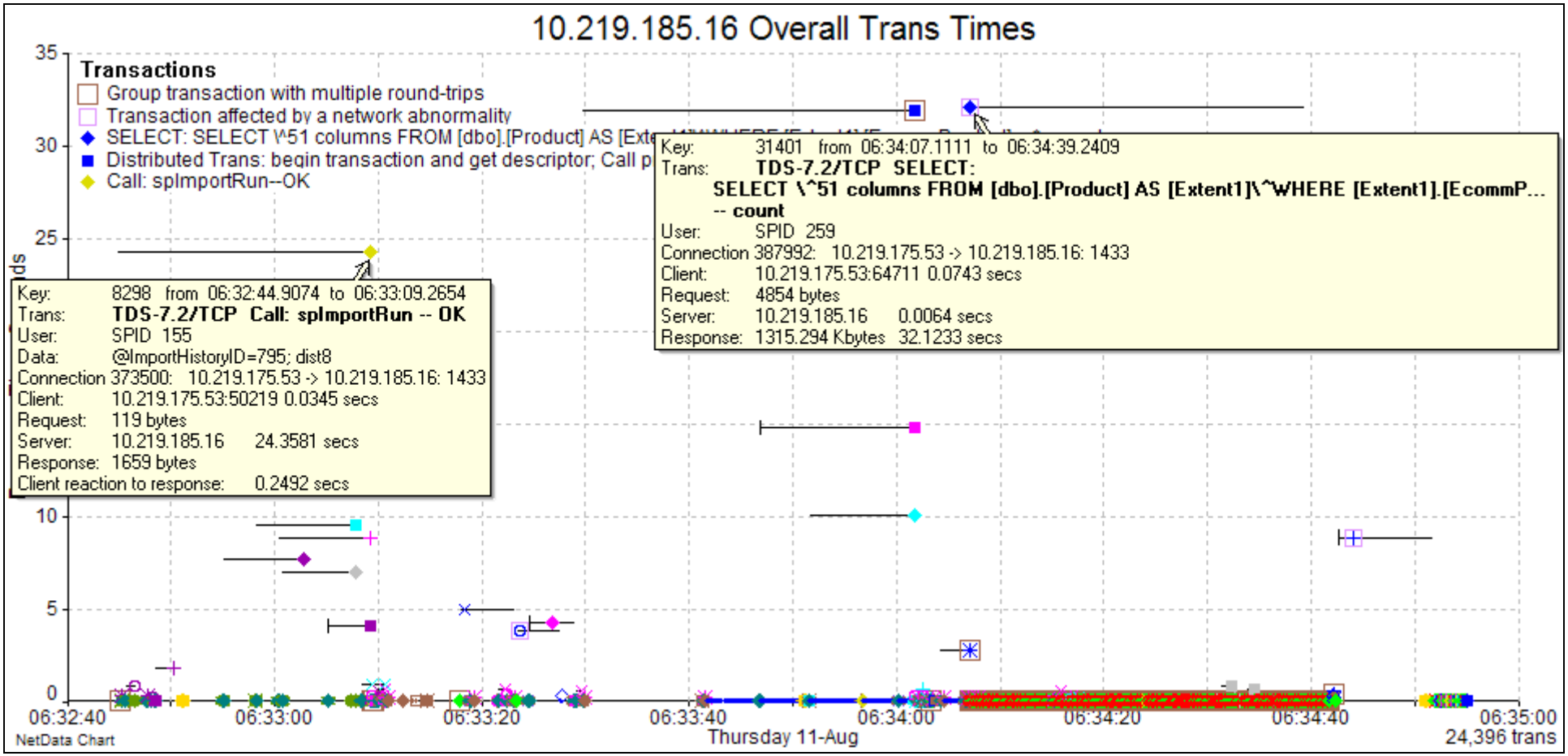


The pop-up box on this chart describes a distributed transaction that involved 11,996 round-trips, two trips to begin and eventually commit the transaction, and 11,994 intervening Update commands. It locked a database table for 32 seconds and explains why two queries were blocked until the distributed transaction finished.

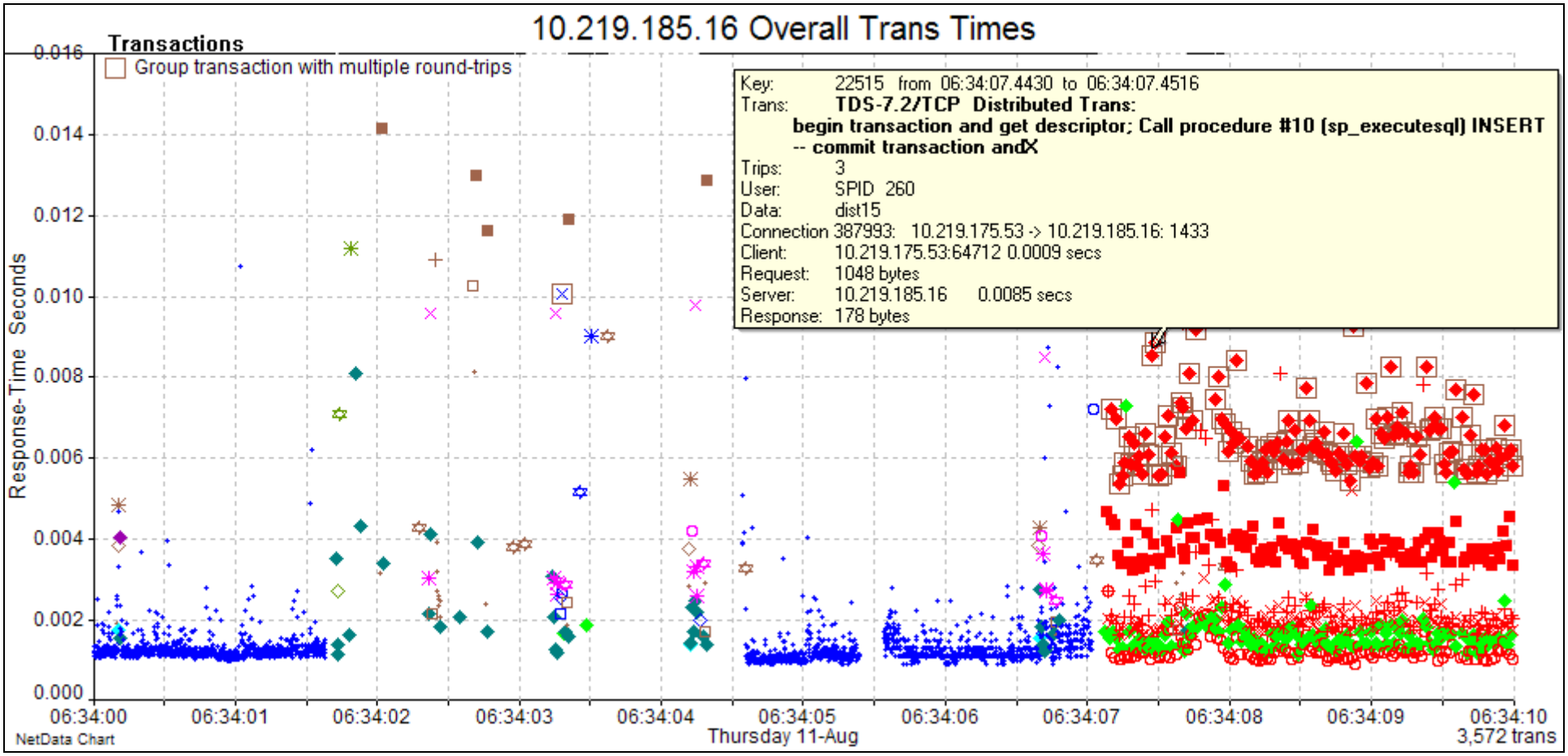
Response-time markers for the component Update round-trips sit on the bottom axis, leaving unexplained a pause of 11 seconds before the updating work began. This 32-second distributed transaction was followed by a long burst of distributed transactions each comprising a single Insert command, and their markers also sit on the bottom axis in brown squares. By showing complete distributed transactions and individual round-trips this chart characterises in fine detail the operation of a major database updating process and explains its interference with other database operations. Further detail appears in pop-up boxes as the cursor is rested on individual markers; the sequence of operations is presented also in the transaction table; and further detail can be provided in separate windows that describe individual transactions.

The next chart uses a technique to identify the activity of different threads in the client machine. The transaction records of this server are reloaded after requesting different pools of statistics for clients rather than servers, and split further by client port number:


Only of service type
All nodes handle same types of transactions
Separate statistics for
clients
Separate stats for different ports
Network events
Warnings
Filtered
Traffic volumes
by IP
All nodes
0
Kbps
Load Trans Type
Call procedure #10 (sp_executesql) UPDATE: update [dbo].[Product]\^set [Ti
Select Trans Type or Server
387992
190485
Load Connection
Load User
10.219.175.53
Load Client
Filter
10.219.185.16
Load Server
Load All Servers
Clear Chart...
Close



The different colour markers identify different client port numbers and thus the activity of different SQL sessions and their client threads. The client's database updating process is recognised by its use of distributed transactions and associated queries; it involved four threads identified here by markers coloured blue, red, green and lemon. The 24-second call to spImportRun (lemon diamond) delayed at least five queries that started during the call, and was followed by the 32-second group transaction that encompassed 11,994 Update commands. Updating was followed by a group transaction with 1552 Delete commands and then a 32-second query that returned 1.3 MBytes. The transfer of that response message was frequently interrupted by a closed receive window, and the window was closed while the client processed the query result progressively, conducting queries and distributed Insert transactions on two separate threads (green and red).



This chart focuses on a critical part of the updating process. The two bursts of blue markers relate to Update and Delete commands within two long-running distributed transactions. The green and red threads performed queries and Insert commands while processing the results of the 32-second query, in a loop described by the following portion of the transaction table:

	Trn Key	Request Strt	Resp End	Type	Class	Description	End Rsp	ConnID	Client	Data
◆	31161	06:34:38.3729	06:34:38.3744	TDS-7.2/TCP	Server	SELECT: SELECT Count(*) FROM sys.databases WH...	0.0015	387965	10.219.175.53:64684	
■	31162	06:34:38.3813	06:34:38.3847	TDS-7.2/TCP	Server	Call procedure #10 (sp_executesql) SELECT: SELECT ...	0.0034	387993	10.219.175.53:64712	1; @p__linq__0=182
◇	31163	06:34:38.3859	06:34:38.3869	TDS-7.2/TCP	Server	Dist Trans Mgr Request: begin transaction and get des...	0.0010	387993	10.219.175.53:64712	dist1411
◆	31166	06:34:38.3859	06:34:38.392	TDS-7.2/TCP	Group	Distributed Trans: begin transaction and get descriptor; ...	0.0062	387993	10.219.175.53:64712	dist1411
⚙	31164	06:34:38.3877	06:34:38.3895	TDS-7.2/TCP	Server	Call procedure #10 (sp_executesql) INSERT: insert [db...	0.0018	387993	10.219.175.53:64712	dist1411; @0=9564;
○	31165	06:34:38.3897	06:34:38.392	TDS-7.2/TCP	Server	Dist Trans Mgr Request: commit transaction andX--OK	0.0023	387993	10.219.175.53:64712	dist1411
◆	31167	06:34:38.3931	06:34:38.3956	TDS-7.2/TCP	Server	SELECT: SELECT Count(*) FROM sys.databases WH...	0.0025	387965	10.219.175.53:64684	
■	31168	06:34:38.402	06:34:38.4057	TDS-7.2/TCP	Server	Call procedure #10 (sp_executesql) SELECT: SELECT ...	0.0037	387993	10.219.175.53:64712	1; @p__linq__0=182
◇	31169	06:34:38.4065	06:34:38.4075	TDS-7.2/TCP	Server	Dist Trans Mgr Request: begin transaction and get des...	0.0010	387993	10.219.175.53:64712	dist1412
◆	31172	06:34:38.4065	06:34:38.4122	TDS-7.2/TCP	Group	Distributed Trans: begin transaction and get descriptor; ...	0.0057	387993	10.219.175.53:64712	dist1412
⚙	31170	06:34:38.4083	06:34:38.4103	TDS-7.2/TCP	Server	Call procedure #10 (sp_executesql) INSERT: insert [db...	0.0020	387993	10.219.175.53:64712	dist1412; @0=9565;
○	31171	06:34:38.4104	06:34:38.4121	TDS-7.2/TCP	Server	Dist Trans Mgr Request: commit transaction andX--OK	0.0017	387993	10.219.175.53:64712	dist1412

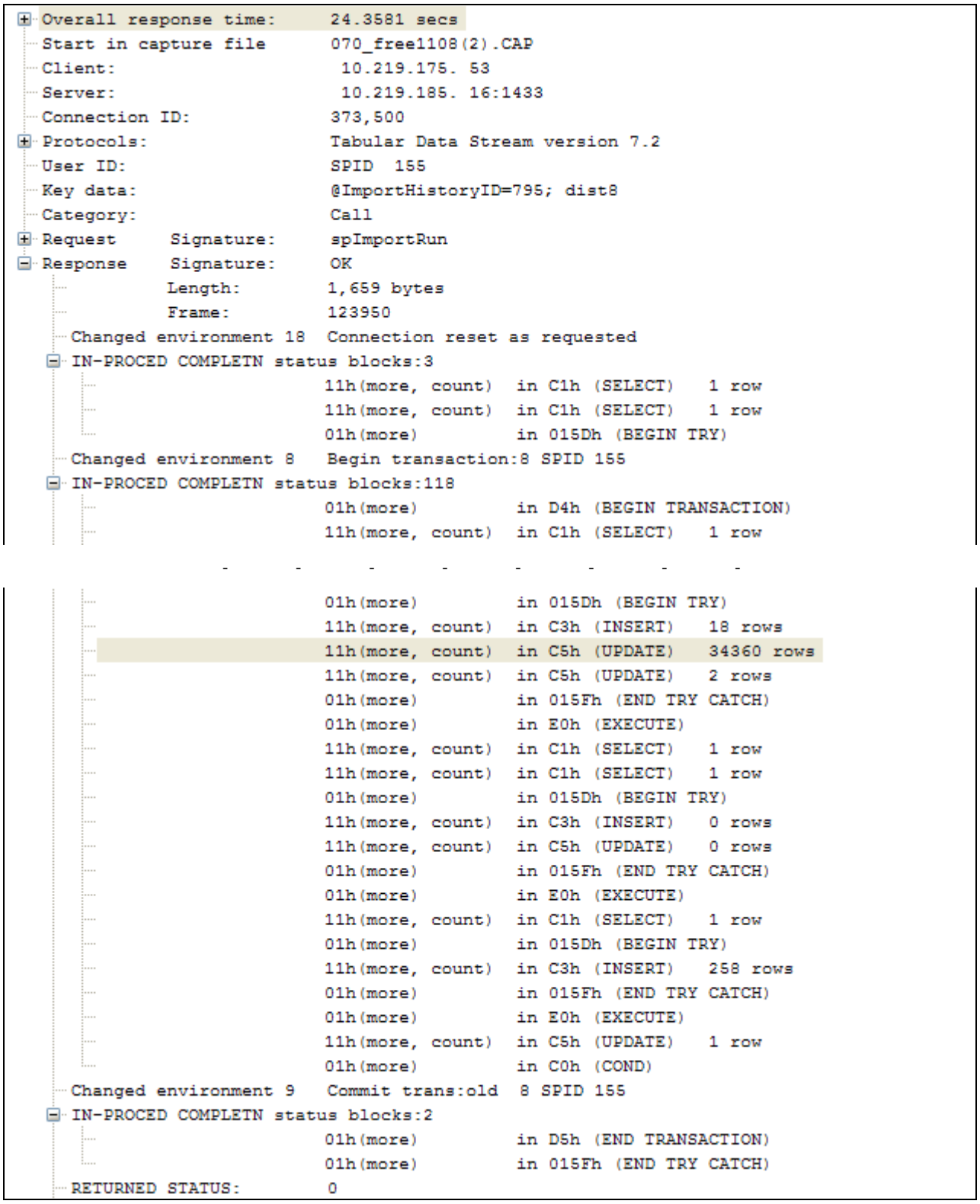
This table covers two iterations of the processing loop, each comprising first a query by the green thread and then a distributed Insert transaction (here with serial numbers 1411 and 1412) by the red thread. The Class column of this table indicates which rows characterise a complete distributed (*Group*) transaction and which rows characterise single (*Server*) round-trips, including the requests sent to the distributed transaction manager.

The updating process was believed to affect overall database performance and this investigation explains why: tables are locked while tens of thousands of rows are updated, and the solution is to break the large distributed transactions into smaller chunks. Analysis of this traffic was able to rule out the possibility of a network problem or an overloaded server.

3.5.6 In-Procedure Status Blocks

Every command in a stored procedure produces an in-procedure status block that appears in the response message, and every status block combines a set of flags with a two-byte ‘current command’ token and a row count. NetData displays the value of the row count only if a status flag indicates that it is valid.

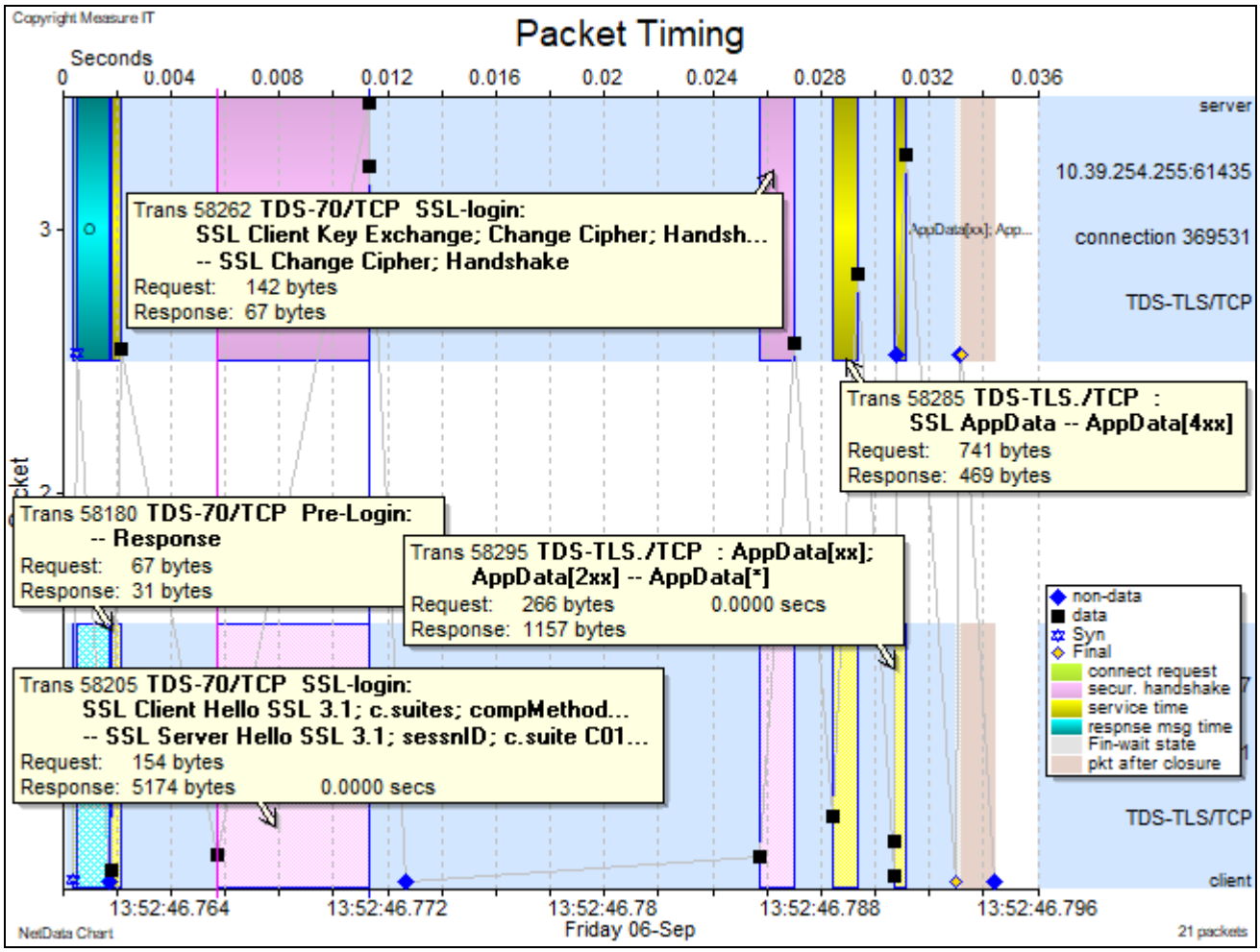
The command token is not interpreted by either TDS or the client and is conveyed only for diagnostic purposes. Although the meanings of the tokens are in principle subject to change by Microsoft the acronyms displayed by NetData for over 400 different tokens give useful clues to the nature of the commands. The following description for a call to a stored procedure illustrates how the status blocks can characterise serious database blockages and other problems:



After two queries this procedure started an error catcher (BEGIN TRY), and a distributed transaction (BEGIN TRANSACTION) that was reported by a changed-environment message. The database operation that took most of the 24 seconds—and blocked co-incidental queries—was an Update operation involving 34,360 rows. The procedure ended with a Commit command (END TRANSACTION) also reported by a changed-environment message, and with closing of the overall error catcher (END TRY CATCH). The error-catch operations always appear in pairs and may be nested; they report any SQL Server errors with severity greater than 10.

3.5.7 SQL Server SSL Login

When the traffic of a Microsoft SQL server is encrypted, the TCP payload of all packets conveying queries and response are fully encrypted with TLS/SSL like HTTPS traffic, but the initial login transactions and SSL handshakes have standard TDS protocol (Token Data Stream) headers in clear. NetData's decoders for TDS, TDS-TLS, and the default TLS dialect tagged as SSL-TLS, recognise TDS logins at the beginning of new connections, decode the handshakes with their TDS headers, and characterise SSL data records as usual.

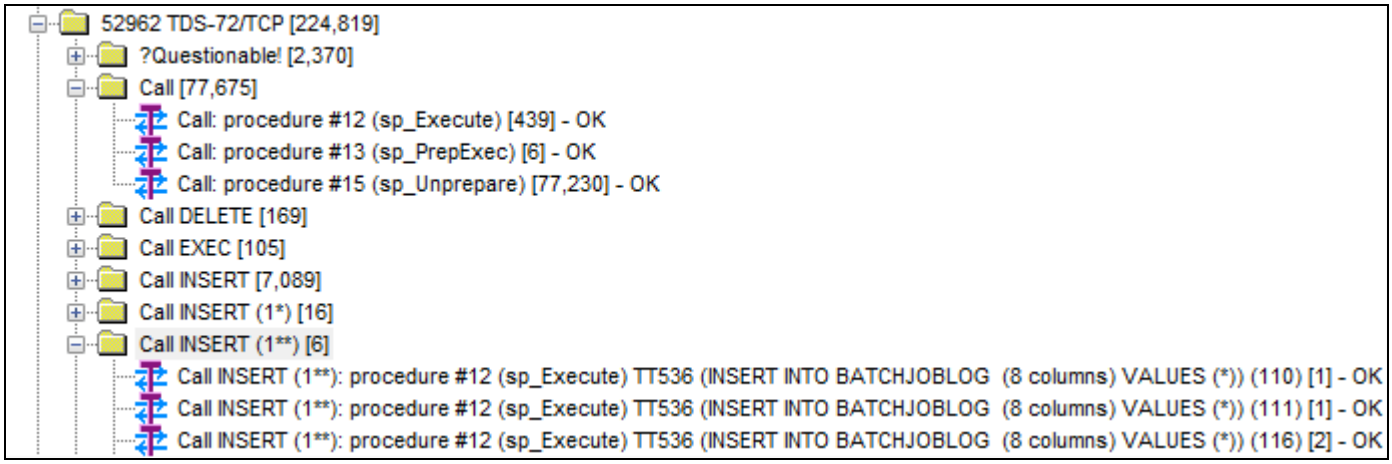


This timing chart describes all the packets and transactions of a short-lived database session that conducts only two data round-trips. The teo round-trips conveying SSL handshake records are coloured purple as for normal handshakes.

3.5.8 SQL Server Transact-SQL System Stored Procedures

Data manipulation operations in Microsoft's SQL Server database are often handled by calls to its system stored procedures such as sp_cursor, sp_cursoropen, sp_cursorprepare, sp_cursorexecute, sp_cursorprepexec, sp_cursorunprepare, sp_cursorfetch, sp_cursoroption, sp_cursorclose, sp_prepare, sp_execute, sp_prepexec and sp_unprepare. Because they are used so frequently, messages making these calls identify the required procedure not by name but by a numeric identifier in the range 1 to 15. NetData has long recognised these identifiers and inserts the corresponding procedure names in transaction descriptions.

The various procedures open or close a cursor (a table of dataset rows extracted from a database), prepare an execution plan for a given Transact-SQL statement, and execute a statement with a given set of values for the bound variables that might be referenced in the statement. Database efficiency can be achieved by executing a previously prepared statement many times, with multiple calls to sp_execute. During analysis NetData extracts cursor IDs and statement handles from call and response messages, and maintains such cursor-state information as statement signatures and column formats. When describing a call to sp_execute it includes the signature – a generalised description – of the executed statement, as in the following segment of a transaction-class tree:



This part of the tree tells that the project characterised 224,819 transactions with a SQL Server database at port 52962, using the TDS 7.2 protocol. There were 77,230 calls to sp_unprepare (to free an execution plan), 169 calls to sp_execute to execute Delete statements, 7,089 calls to sp_execute to execute a single Insert statement, and (on the last row) 2 round-trips that each made 116 calls to sp_execute to execute an Insert statement.

A code such as 'TT536' in a transaction description refers to NetData's identifier for the transaction type of the original database command from which the execution plan was constructed. It allows NetData to describe a database transaction with two windows – one to describe the execution command, and another to describe the original command that specified the SQL statement, even if it appeared on the network hours earlier. The first window will display the set of values for the bound variables, and the second window might be used to explore the structure and details of the relevant query, particularly if it is large and complex. The second window is requested with a right-click on the first window and by choosing 'Describe Original Command' from the context menu. The same facility is also available for exploring DB2 database transactions that open and execute previously defined queries.

As with DB2 and Oracle database transactions, NetData assigns a TDS transaction's cursor ID or statement handle (if present) to its secondary connection ID. By sorting the connection ID column of the transaction table it is possible to find all the activity of a particular cursor.

3.5.9 Describing Transact-SQL Statements

In transaction and packet message descriptions NetData reveals the structure of SQL statements by displaying the statements in tree views, with each clause starting in a separate row. NetData recognises some relatively new clauses allowed in Microsoft Transact-SQL statements, such as Into in Select statements and Output in Delete statements. It also recognises new datatypes in TDS token streams.

One new datatype, signalled by the token F3h, is a user-defined table type or Table Valued Parameter (TVP) that serves as an input parameter and is found only in client packets. It encapsulates an entire table with up to 1024 columns and an arbitrary number of rows. In the example below the table called *UserProfileIds* has only one row of one column containing a 46-character ID. The table is accessed in a subquery whose results populate an IN subclause of a Where clause.in each of four Select statements submitted to the stored procedure number 10 (sp_ExecuteSQL).

```
SMP header flag 08h          Data
Block 1:last
headers [22] type 2          transaction descriptor [18]
Call                         procedure #10 (sp_ExecuteSQL)
      options                2h  no metadata to be returned
Parameters
   name            value                                                          length   type     pr
-------------------------------------------------------------------------------------------------------
statement       \^SELECT * FROM VW_EffectiveUserProfile WHERE UserProfileId IN (SELECT Value FROM
   SELECT *
   FROM VW_EffectiveUserProfile
   WHERE UserProfileId IN (
   SELECT *
   FROM EffectiveUserRoleMapping
   WHERE UserProfileId IN (
   SELECT *
   FROM EffectiveUserLoginRights
   WHERE UserProfileId IN (
      SELECT Value
      FROM @UserProfileIds)\^
   SELECT *
   FROM EffectiveUserGroupMembership
   WHERE UserProfileId IN (
      SELECT Value
      FROM @UserProfileIds)
param types     @UserProfileIds [VarChar256TableType] READONLY                  92 E7h=n_var char
@UserProfileIds VarChar256TableType (type) columns=1                               F3h=table
   col 1        user_type=0 nullable                                           -1 E7h=n_var char
   row 1        S-1-5-21-1731099354-1922606306-2941623471-5104
```

The raw packet contents, even when displayed in NetData's Auto mode, are harder to interpret:

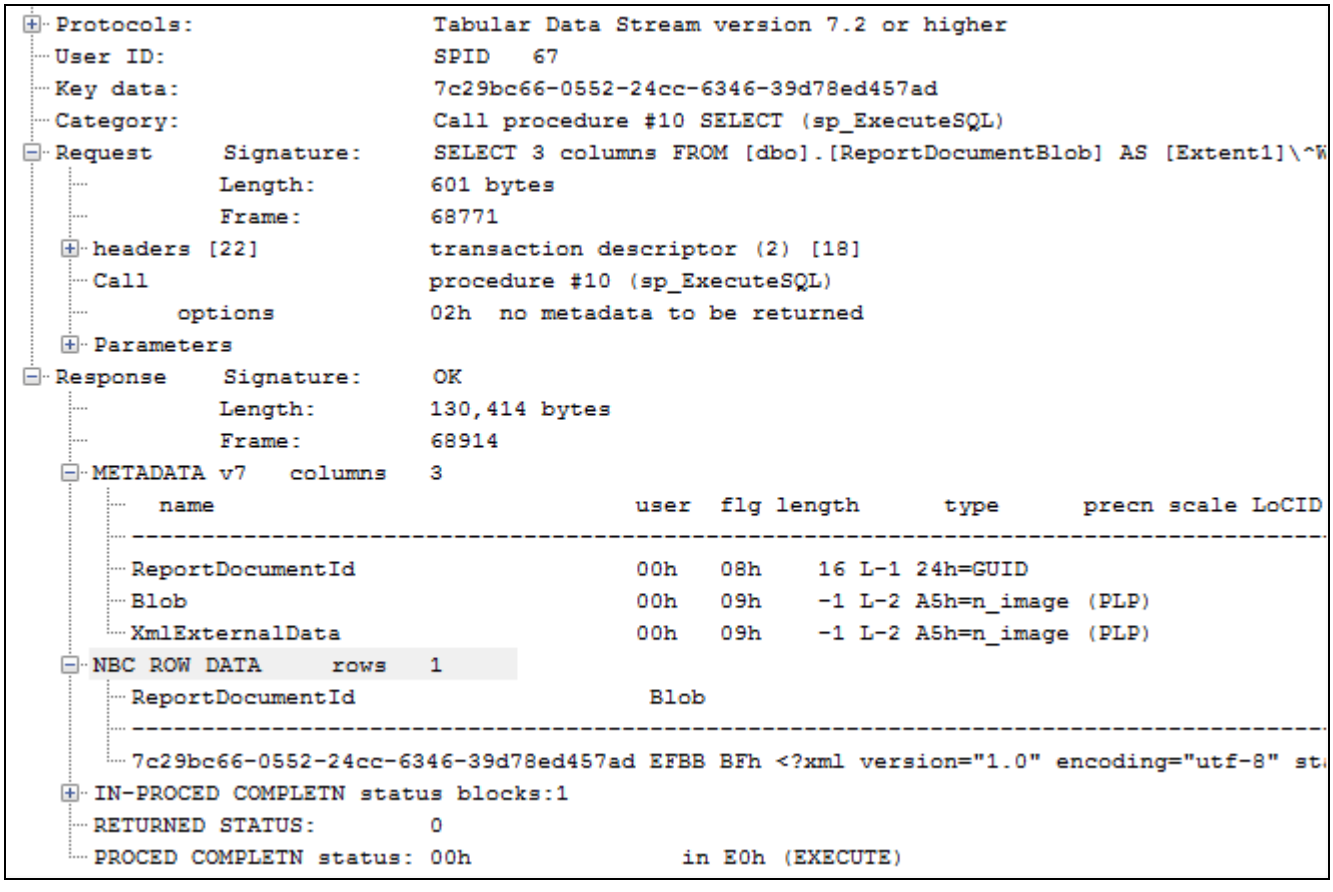
```
5308 0100 9004 0000  3107 0000 3407 0000  0301 0480 0000 0100  1600 0000 1200 0000
 0200 5002 0000 5F01  0000 0100 0000 FFFF  0A00 0200 0000 E71E  0309 04D0 0034
1E03h \z^z Uni[SELECT * FROM VW_EffectiveUserProfile WHERE UserProfileId IN
(SELECT Value FROM @UserProfileIds)\^SELECT * FROM EffectiveUserRoleMapping WHERE
UserProfileId IN (SELECT Value FROM @UserProfileIds)\^SELECT * FROM
EffectiveUserLoginRights WHERE UserProfileId IN (SELECT Value FROM
@UserProfileIds)\^SELECT * FROM EffectiveUserGroupMembership WHERE UserProfileId
IN (SELECT Value FROM @UserProfileIds)z]`\z~ 04D0h z4 Uni[\@UserProfileIds
[VarChar256TableType] READONLY]*@z Uni[UserProfileIds]z F300 0013h Uni
[VarChar256TableType] 0100 0000 0000 0100  E7FF FF09 04D0 0034  0000 015C 00[*6]
005Ch zzz Uni[S-1-5-21-1731099354-1922606306-2941623471-5104zz]z
```

3.5.10 Tabular Data Stream Data Formatting

Tabular Data Stream (TDS) encodes the responses to queries and other commands with a stream of data blocks of various types preceded by identifying single-byte tokens. A query response normally contains two prominent blocks, one of metadata defining columns in the result set, and the other conveying the rows of data in the result set. The full protocol is described in the Microsoft document [MS-TDS] Tabular Data Stream Protocol.

NetData's TDS decoder handles two data-formatting schemes. One scheme, known as Null Bitmap Compression (NBC), saves space by indicating the presence of null fields in a data row by setting bits in a bitmap for the columns of the data set.

The other scheme, known as Partially Length-Prefixed (PLP) bytes, handles variable-length strings of text or binary data that are longer than the usual TDS limit of about 8,000 bytes. Use of this scheme is indicated in the block of column metadata and precedes the relevant data values with an 8-byte length indicator. This indicator has special negative values to indicate whether the value is null or its length is unknown. Otherwise it provides a hint regarding the size of buffer that the receiver will need, and this length value need not be correct. To indicate the correct length, the data is assembled as a sequence of chunks, each preceded by a 4-byte length indicator, and the sequence is terminated with a zero-length chunk.



3.6 Microsoft SQL Monitor, Port 1434

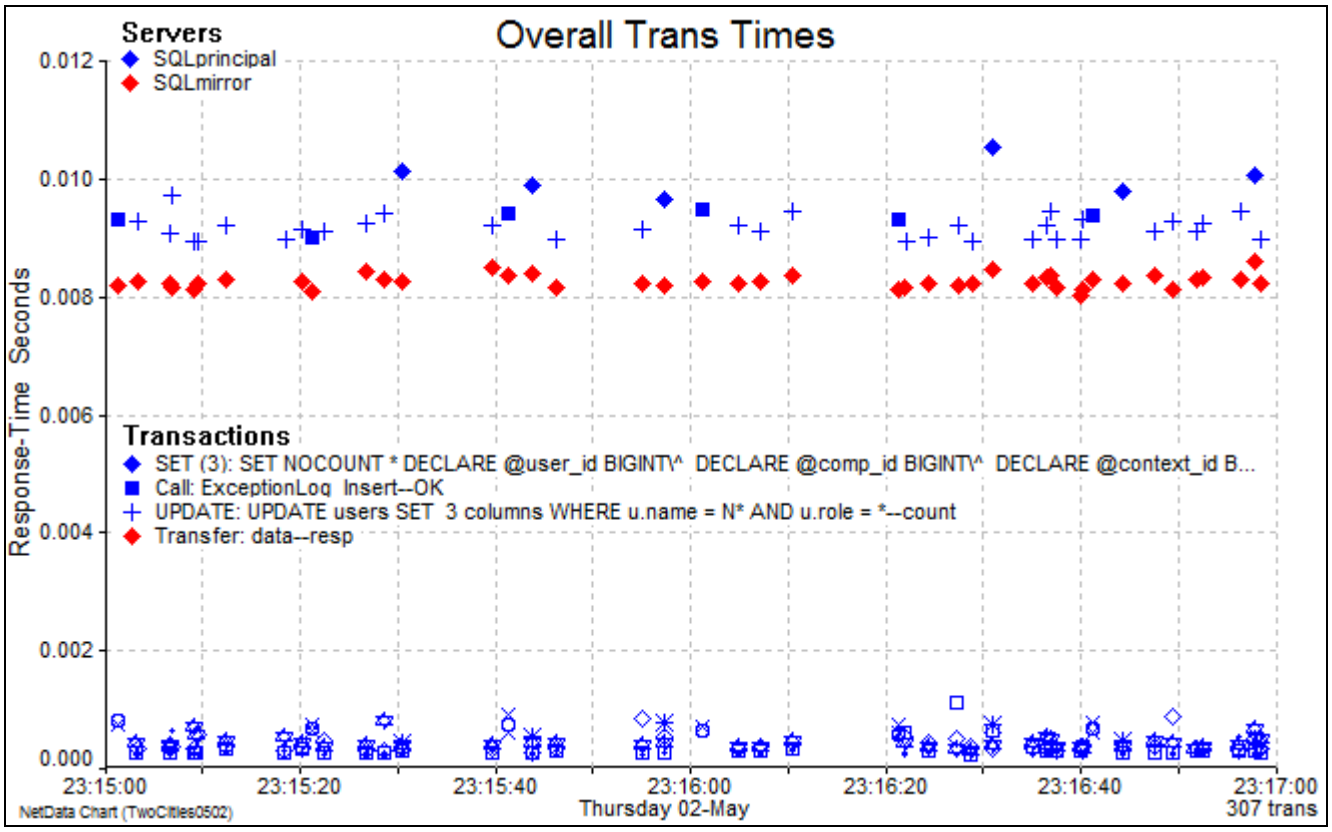
The SQL Monitor decoder recognises another type of message sent to port 1434. If it sees an abnormally long message with a relevant signature it identifies the Slammer worm that tries to exploit weaknesses by sending abnormally long messages for the monitor service and placing a new jump instruction on a corrupted stack.

3.7 Microsoft SQL Server Database Mirroring

NetData now detects pairs of TCP connections between a SQLserver principal server and its mirror server. Each server accepts a connection on port 5022. Because message contents are compressed and encrypted NetData characterises transactions only in terms of message lengths.

NetData assumes that messages with a length of 192 bytes are supervisory messages that are independent of other messages and are not part of any round-trip transaction.

The mirroring process conveys database logs from the principal that can be applied to the mirror database at the physical level. Large mirroring response times would be a concern if mirroring was conducted in the *high-safety mode*, because delays in the mirror would delay responses from the principal database to its clients.



This chart was drawn from traffic captured at the switch port of a principal SQLserver database server. It must have been operating in the high-safety mode because all three types of transaction that changed the database had response times about one millisecond greater than the response time (red marker) of a corresponding transfer to a mirror server in another city.

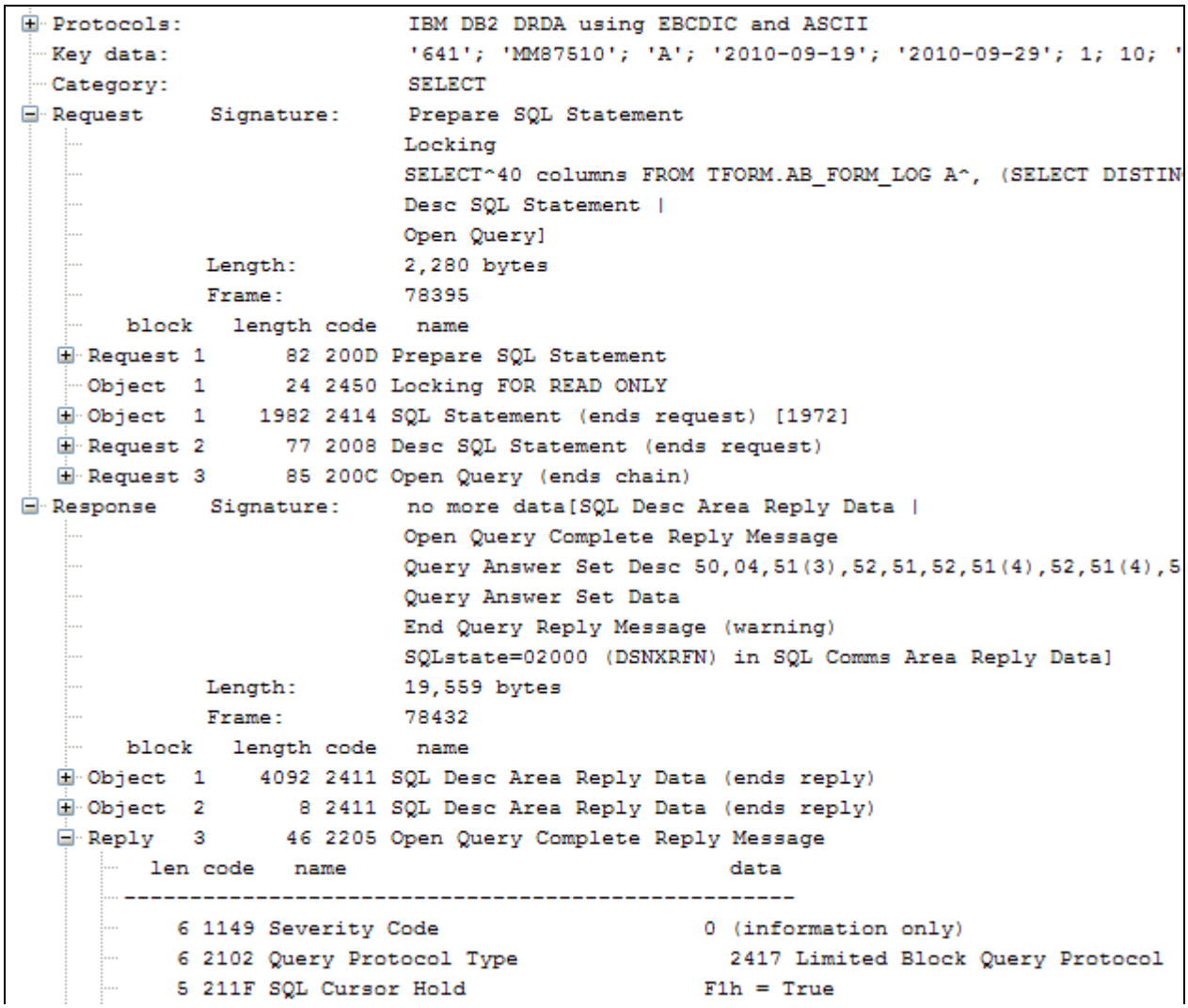
3.8 IBM DB2 DRDA and DDM

Communication between an application and an IBM DB2 database is normally controlled by protocols of the Distributed Relational Database Architecture (DRDA) which comprises the Formatted Data Object Content Architecture (FD:OCA) and the Distributed Data Management (DDM) architecture. DDM defines the commands, parameters, objects and messages of DDM data streams that exchange data between clients and servers. Although first developed by IBM, DRDA is now maintained as a technical standard of The Open Group.

What NetData regards as request and response messages, DDM describes as Request and Reply data streams. A Request data stream is a sequence of Request messages and objects, and a Reply stream is a sequence of Reply messages and objects. NetData regards each message and object as a data block.

Each block has three parts, a two-byte length indicator, a two-byte codepoint, and the block's data. The codepoint, normally expressed in hexadecimal notation, is an alias for the name or type of the block.

The data of a message is usually a sequence of objects, each with its own length, codepoint and data. NetData's description of a data stream respects the structure of objects within messages, as illustrated by the expansion of Reply 3 of the following transaction:



```
Object  1    4092 2411 SQL Desc Area Reply Data (ends reply)
Object  2       8 2411 SQL Desc Area Reply Data (ends reply)
Reply   3      46 2205 Open Query Complete Reply Message
   len code   name                                 data
  ------------------------------------------------
     6 1149 Severity Code                        0 (information only)
     6 2102 Query Protocol Type                    2417 Limited Block Query Protocol
     5 211F SQL Cursor Hold                      F1h = True
     5 2150 Query Attribute Updatability         1 (cursor is read only)
    12 215B Query Instance Identifier            0000 0240 AC15 6FC8h
     8 215F Query Blocking Factor                0 (no blocking limit imposed by targe
Object  3     372 241A Query Answer Set Desc
Object  3   14886 241B Query Answer Set Data (length 8008h)  rows 26
Reply   3      32 220B End Query Reply Message
Object  3      81 2408 SQL Comms Area Reply Data (ends chain):
```

The above structured transaction description displays only a small part of the complete tree, but enough to show all the first-level objects and messages in the Request and Reply data streams. Any part of the tree can be expanded to display the full contents of an object.

The numbers (1, 2, 3...) associated with each block are correlation IDs assigned by the communications manager to help the client associate replies with requests. Each block in a data stream is preceded by a two-byte correlation ID and an extra byte conveying the block type and three flags. One flag indicates the end of a request or reply, and another flag indicates the end of a chain of requests or replies.

In the above query description one of the Request objects, codepoint 2414, contains the complete SQL statement which can be displayed fully or partially in its structured form:

```
Object  1    1982 2414 SQL Statement (ends request) [1972]
SELECT^              columns  40
FROM                 tables   3
   table             alias
  ------------------------
  TFORM.AB_FORM_LOG A^,
(subquery)           B^,
(subquery)           C^
  SELECT               columns  2
  FROM                 TFORM.AB_FORM_LOG D^
  WHERE D.CTRY_NO =                  '641'^
    AND D.SEQ_NO =                   (
    SELECT             CASE MAX(E.SEQ_NO) ^     ~~~~~~  ~~
      WHEN 1
        THEN 1
        ELSE MAX(E.SEQ_NO) - 1
      END
    FROM               TFORM.AB_FORM_LOG E^     ~~~~~~~~
    WHERE E.CTRY_NO =                  D.CTRY_NO^     ~~~~~~~
  AND SUBSTR(CHAR(D.LAST_UPDT_TIME), 1, 10) BETWEEN '2010-09-19' AND '2010-
WHERE A.CTRY_NO =                  '641'^~
ORDER BY
```

This query accessed one table and two sub-queries. The second sub-query had a third sub-query in its Where clause.

3.8.1 Transaction Signatures

The preceding panel shows how the transaction's request and response signatures summarise their respective data streams by listing the names of the first-level objects and messages. An

object with a SQL statement, however, is represented by the generalised version of the statement:

```
SELECT^40 columns FROM TFORM.AB_FORM_LOG A^, (SELECT DISTINCT 7 columns FROM TFORM.CIS_R
```

The name of an object with a data-set description is extended with a list of the type codes (attribute IDs) of the fields in the data row. It helps to characterise and distinguish different queries if the relevant SQL statement was not captured in the network traffic:

```
Query Answer Set Desc 50,04,51(3),52,51,52,51(4),52,51(4),53,54,55,56,51(3),50,51,52,57,
```

If a command produces a warning or error, the response signature extends the Reply-message name with an indication of the error's severity, and prefixes the name of the SQL Comms Area Reply Data (SQLCARD) with both the returned SQLstate code and the name of the program in which the error occurred:

```
End Query Reply Message (warning)
SQLstate=02000 (DSNXRFN) in SQL Comms Area Reply Data ]
```

The warning or error indication is also prefixed to the complete signature to give it prominence in the transaction-class tree.

A signature includes special characters that indicate the position of flags signifying the end of a request or reply (with a vertical bar |) or the end of a chain (with a closing bracket]).

3.8.2 Data-Row Format Specifications

The Query Answer Set Description object defines the format of a data row in a recursive manner, identifying subgroups and their subgroups down to lists of individual data types according to rules and codes of the content architecture FD:OCA. Every element in the hierarchy is defined by an attribute triplet of three bytes. The first byte is an ID defining the attribute or data type, and the next two bytes form a parameter whose meaning depends on the attribute—it might, for example, specify the length of a character string. As the following example illustrates, the object description begins with the lowest-order attributes and ends with an attribute for the whole data row:

```
Object  2      67 241A Query Answer Set Desc
 76  D0       nullable group of single-row fields 15
    ID  param description
    ------------------------------------------------
    30h    20 fixed character SBCS
    04h     2 2-byte integer
    30h     6 fixed character SBCS
    30h     6 fixed character SBCS
    20h    10 date
    30h    20 fixed character SBCS
    30h    30 fixed character SBCS
    30h     4 fixed character SBCS
    02h     4 4-byte integer
    20h    10 date
    30h     6 fixed character SBCS
```

```
    22h     8 time
    30h     8 fixed character SBCS
    30h     8 fixed character SBCS
    04h     2 2-byte integer
71  E0      RowLayOut header E0
  ID  param description
  ------------------------------------------------
  54h     1 SQL CommsArea
  D0h     1 user data
71  F0      SQLData As Reply Data
  ID  param description
  ------------------------------------------------
  E0h     0
```

This description ends with the ID (F0h) for the Reply Data which is here composed of only one attribute, E0h. Working backwards through the description, that attribute, for the row-layout header, comprises a SQL Comms Area (54h) and user data (D0h). The user data is a nullable group of 15 single-row fields, listed at the beginning of the description.

The following description for a different data-set begins with metadata defining application-specific data types with codes 50h to 5Ah:

```
Object  3     372 241A Query Answer Set Desc
  78  00        meta data:   5  1(data)   1  DRDA type:3Eh
  70  50        simple type: 11000003A701017FFF   type:3Eh variable character mixed
  78  00        meta data:   5  1(data)   1  DRDA type:3Ch
  70  51        simple type: 10000003A701007FFF   type:3Ch fixed character mixed
  78  00        meta data:   5  1(data)   1  DRDA type:32h
  70  52        simple type: 110000034401017FFF   type:32h variable character SBCS
  78  00        meta data:   5  1(data)   1  DRDA type:24h
  70  53        simple type: 10000003440100001A   type:24h timestamp
  78  00        meta data:   5  1(data)   1  DRDA type:24h
  70  54        simple type: 10000003440100001A   type:24h timestamp
  78  00        meta data:   5  1(data)   1  DRDA type:24h
  70  55        simple type: 10000003440100001A   type:24h timestamp
  78  00        meta data:   5  1(data)   1  DRDA type:33h
  70  56        simple type: 910000034401017FFF   type:33h nullable variable charact
  78  00        meta data:   5  1(data)   1  DRDA type:24h
  70  57        simple type: 10000003440100001A   type:24h timestamp
  78  00        meta data:   5  1(data)   1  DRDA type:24h
  70  58        simple type: 10000003440100001A   type:24h timestamp
  78  00        meta data:   5  1(data)   1  DRDA type:24h
  70  59        simple type: 10000003440100001A   type:24h timestamp
  78  00        meta data:   5  1(data)   1  DRDA type:24h
  70  5A        simple type: 10000003440100001A   type:24h timestamp
  78  00        meta data:   5  2(group)  1  DRDA type:D0h
  76  D0        nullable group of single-row fields 40
    ID  param description
    ------------------------------------------------
    50h    23 variable character mixed
    04h     2 2-byte integer
    51h    30 fixed character mixed
    51h    20 fixed character mixed
```

Some of the attributes listed for the user-data group (D0h) are application-specific data types (such as 50h and 51h) and the others are system data types such as a 2-byte integer (04h). This description ended in the conventional way, with simple specifications for the row-layout header, E0h, and for the Reply Data as a whole, F0h.

NetData walks through the hierarchy of attributes as it scans each row of data in the Reply Data object, to separate and read the data values into their respective columns and thus reconstruct the complete data table:

```
Object  3   14886 241B Query Answer Set Data (length 8008h)  rows 26
    CommsA LOG                      SEQ_NO WORK_CAT
    ------------------------------------------------------------------
    Null   TRA8232099090416-0            2 TRAVEL 823C CONTRACT PREP  RE
    Null   TRA8232099090418-0            2 TRAVEL 823C CONTRACT PREP  RE
    Null   TRA8232099090420-0            2 TRAVEL 823C CONTRACT PREP  RE
    Null   TRA8232099090421-0            2 TRAVEL 823C CONTRACT PREP  RE
    Null   TRA8232099090422-0            2 TRAVEL 823C CONTRACT PREP  RE
    Null   TRA8232099090423-0            2 TRAVEL 823C CONTRACT PREP  RE
    Null   TRA4353209091418-0            2 TRAVEL 435 LOADING REQUEST
    Null   TRA4353209091428-0            2 TRAVEL 435 LOADING REQUEST
```

All the rows in this table began with a nulled Comms Area. For the column headings in this data table NetData extracted the column names or aliases from the Select statement:

```
SELECT^              columns  40
    expression                                          alias
    ----------------------------------------------------------
    A.LOG_NO || '-' || CHAR(A.SUB_LOG_NO)    LOG,
    A.SEQ_NO,
    A.WORK_CAT,
  CASE A.STS_CODE^                           STATUS,^
    WHEN 'N'
      THEN^~CASE A.REJCT_IND ^
        WHEN 'Y'
          THEN 'NEW(R)'
          ELSE 'NEW'
        END
    WHEN 'S'
      THEN 'SAVE'^
    WHEN 'C'
      THEN 'COMPLETE'^
    WHEN 'R'
      THEN 'REJECT'^
    END
```

The alias for a column value produced by a case statement is specified after the statement but, as this example shows, NetData promotes the alias (STATUS) to the top of the statement's structured description to make it visible even when the description is not expanded.

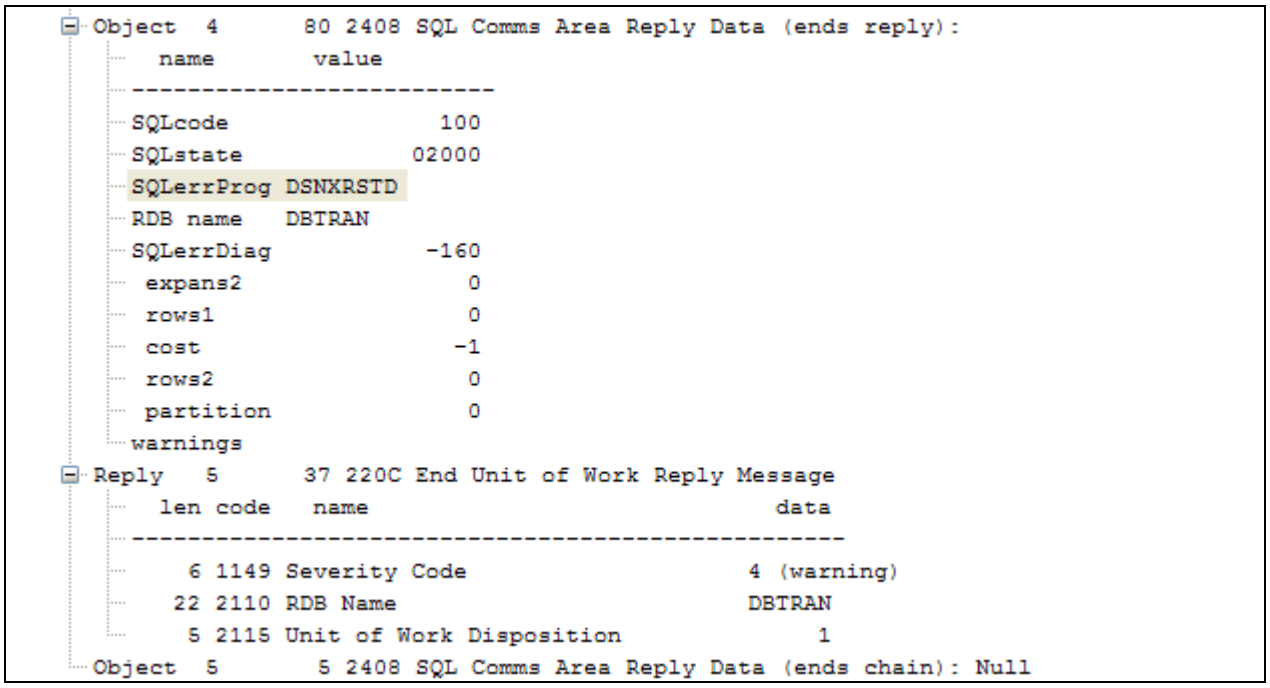
3.8.3 Error Indications

When the following Insert transaction failed the response was a single reply message that described a syntax error associated with external data.

```
Category:                  INSERT
Request     Signature:     Prepare SQL Statement
                           insert into ARMTEST."ABC_FORMEX"  (19 columns) ^values (*)
                           Desc SQL Statement |
                           Exec SQL Statement
                           SQL Program Variable Data 41,05(2),41(2),05,CDh incomplete
            Length:        95,789 bytes
            Frame:         24
    block   length code   name
  Request 1       77 200D Prepare SQL Statement
  Object  1      551 2414 SQL Statement (ends request) [543]
    insert into           ARMTEST."ABC_FORMEX"
      columns             19 (
      values (
  Request 2       77 2008 Desc SQL Statement (ends request)
  Request 3       77 200B Exec SQL Statement
  Object  3       77 2412 SQL Program Variable Data
Response    Signature:     error[Syntax Error Reply Message] (error)
            Length:        139 bytes
            Frame:         53
    block   length code   name
  Reply   -1     133 124C Syntax Error Reply Message (ends chain)
     len code   name                                  data
    ------------------------------------------------------
       6 1149 Severity Code                        8 (error)
      90 1153 Server Diagnostic
       name      value
      ------------------------
                              0
      RDB name DBTRAN
                              0
                           4102
      server    DB2 DDF Server
      program   DSNLCMRL0026
                51DCh
      22 2110 RDB Name                            DBTRAN
       5 114A Syntax Error Code                        0
       6 000C Code Point                          146C   External Data
```

NetData's request signature indicates that the reply message was issued before the request chain was complete—Object 3 was the last complete block before the reply appeared, but that block had not flagged the end of the message or the end of the chain (the signature doesn't end with a closing bracket). The packet timing chart revealed that the client was still transmitting an object with external data when the reply appeared.

As is often the case, this transaction error was preceded by many warnings in preceding transactions, from this client and another client that was accessing similar records. This Insert command was preceded by a Delete command that generated the following warning:



For a Fetch, Update or Delete command a SQLcode of 100 means that a row was not found, and a SQLstate of 02000 means that the last row of a query data-set has been returned. The label SQLerrProg given by NetData to the third parameter (DSNXRSTD) is normally abbreviated to SQLERRP; it is the name of the SQL procedure or program CSECT (control section) that detected the error. Further diagnostic information is likely to be found by searching the web or database documentation for references to the program name with the given values of SQLstate and SQLcode.

The six parameters beginning with SQLerrDiag (normally abbreviated to SQLERRD) provide additional diagnostic information. The meanings of the parameters depend on the error and don't always relate to the labels attached by NetData.

The warnings field is a string of 11 characters acting as flags with individual meanings that also depend on the error type.

A zero SQLcode indicates no error or exception condition. A negative value indicates a hard error (execution was unsuccessful), and a positive value indicates an exceptional but valid condition (a warning).

3.8.4 Data Blocks and Minimising Round Trips

An open-query object usually sets a query-block size of 32,767 but it can have a maximum value of 10,485,760 (10 MB).

```
Request 2      94 200C Open Query (ends chain)
   len code   name                                  data
  ------------------------------------------------------
    68 2113 Package Name, Consistency Token, Section N
     name         value
     --------------------------
     RDB name     DBTRAN
     collection   NULLID
     package      SYSSN200
     consistency SYSLVL01
     section                   5
      8 2114 Query Block Size                    32767 = 7FFFh
      6 2141 Max Blocks Extra                    -1 (unlimited)
      8 1900 Monitor                             8000 0000h (elapsed time)
```

The more important parameter is the number of extra blocks allowed in reply data. The default value for this optional parameter is zero, which means that the reply data can return only one block. The above request, however, allows an unlimited number of blocks and could receive a complete data-set in one trip. While this parameter indicates what the client can receive, it does not compel the server to send the specified number of blocks. The DDM architecture (*DRDA Volume 3*, The Open Group) states that

> The number of extra blocks actually returned is dependent on the capabilities of the target SQLAM (Application Manager) and the dynamic state of the target SQLAM at the time it executes the command. This parameter is only meaningful when the target SQLAM selects the limited block protocol.

The application manager indicates the chosen query protocol in its reply message:

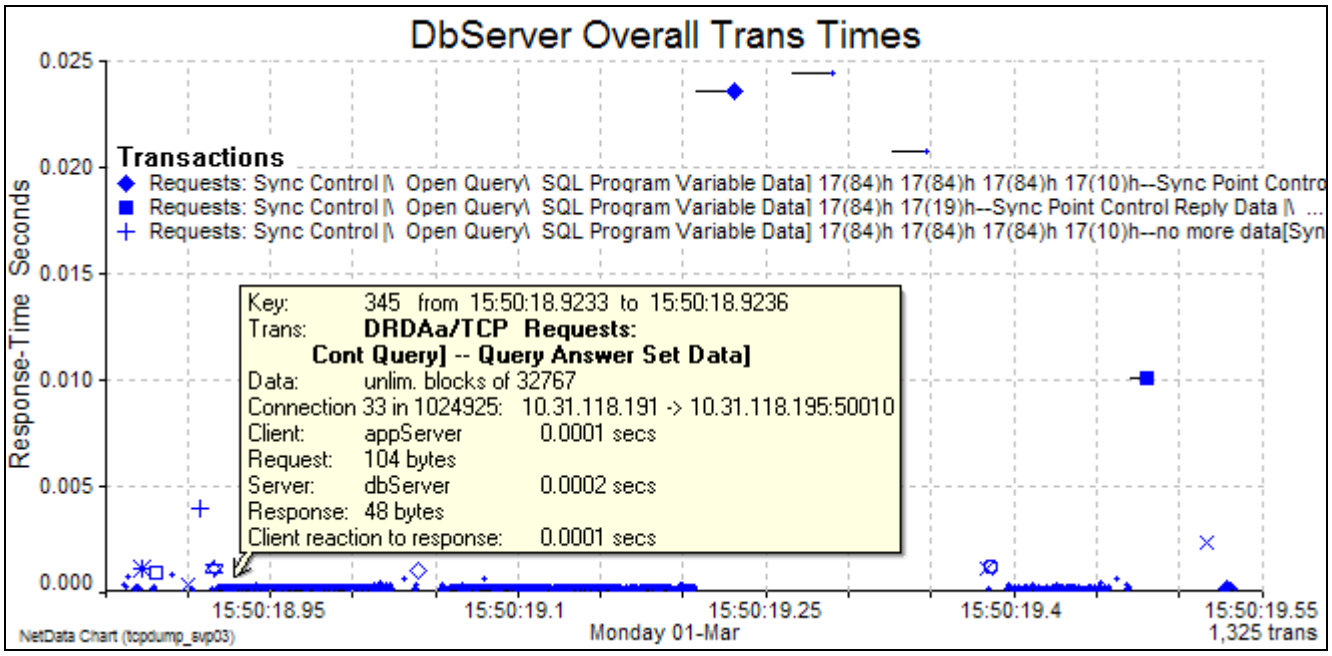
```
Response    Signature:      no more data[SQL Desc Area Reply Data
                            Monitor Reply Data |
                            Open Query Complete Reply Message
                            Query Answer Set Desc 30,04,30(2),20,30(3),02,20,30,22,30(
                            End Query Reply Message (warning)
                            SQLstate=02000 (DSNXRFF) in SQL Comms Area Reply Data
                            Monitor Reply Data]
            Length:         987 bytes
            Frame:          58
     block   length code   name
  Object  1      713 2411 SQL Desc Area Reply Data
  Object  1       16 1C00 Monitor Reply Data (ends reply)
  Reply   2       21 2205 Open Query Complete Reply Message
      len code   name                                  data
     ------------------------------------------------------
        6 1149 Severity Code                      0 (information only)
        6 2102 Query Protocol Type                  2417 Limited Block Query Protocol
        5 2150 Query Attribute Updatability     1 (cursor is read only)
  Object  2       67 241A Query Answer Set Desc
  Reply   2       32 220B End Query Reply Message
  Object  2       80 2408 SQL Comms Area Reply Data:
  Object  2       16 1C00 Monitor Reply Data (ends chain)
```

A query data block is composed with either *exact* or *flexible* blocking. With exact blocking the size of every block but the last is exactly the specified block size, and may start and end with

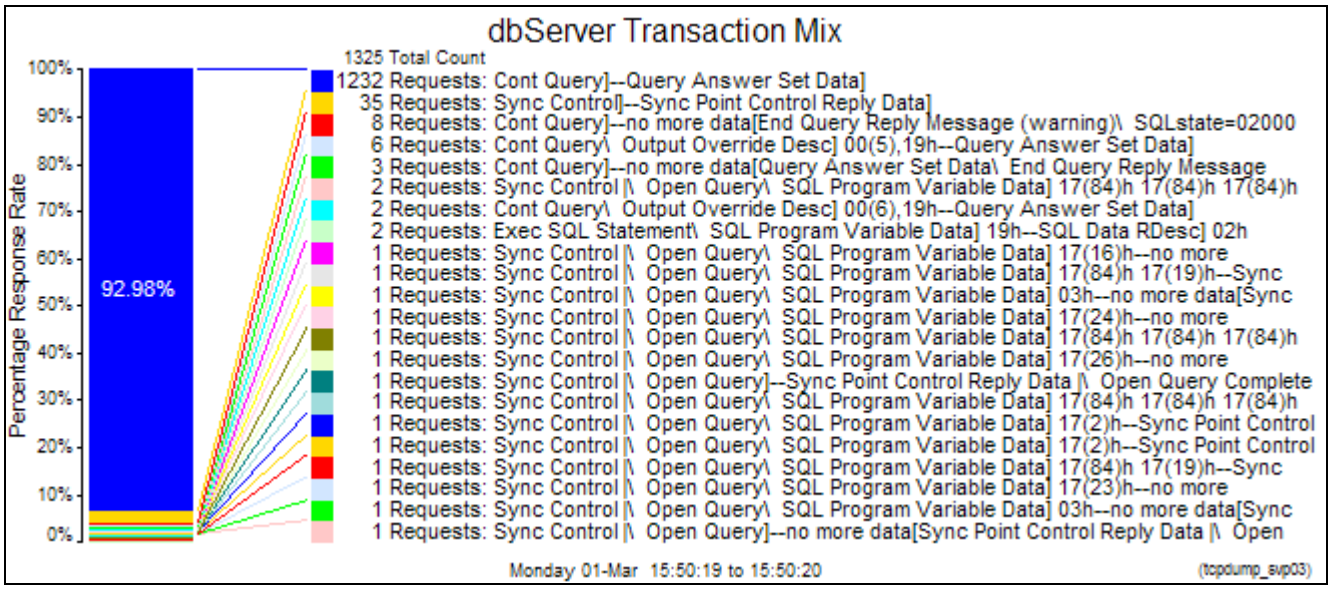
part of a data row. With flexible blocking a block contains only complete data rows and its size may be increased beyond the specified size in order to complete a row. In both cases NetData constructs a structured table to display the contents of each block with complete data rows aligned in the proper columns. With exact blocking the last row in the NetData table may be incomplete, and that partial row is carried over to be displayed in its complete form with the content of the next block.

The application manager may indicate its choice of block type with a Codepoint 2133 object in a reply message. The default type is exact blocking.

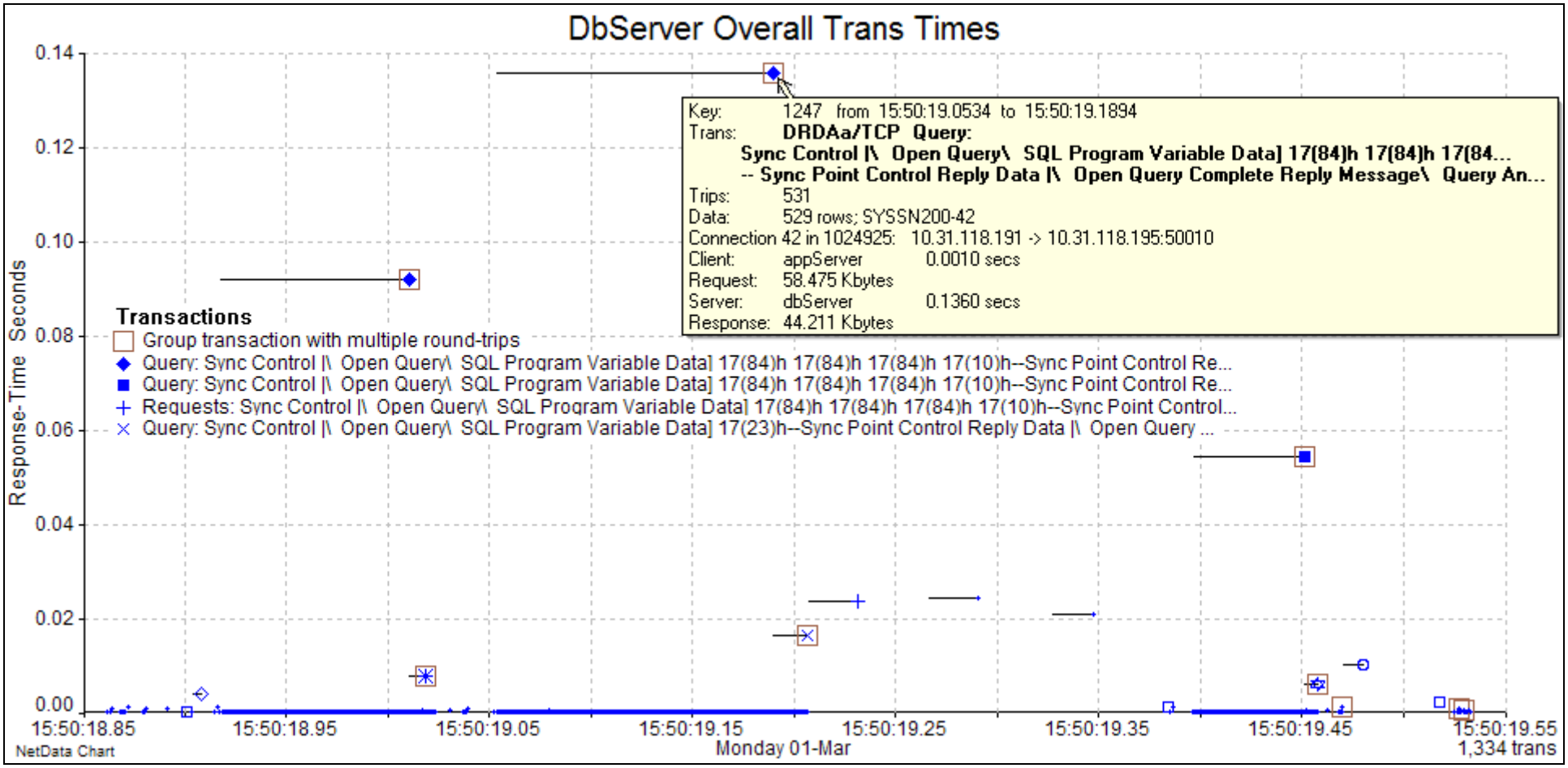
The chart below shows database activity for what was probably a single user transaction that took only 0.7 seconds but involved 1325 round-trips to the database.



The long bands of identical markers suggest an inefficiency that is confirmed by the transaction mix:



93% of the transactions were Cont Query requests to fetch more data from a result set.



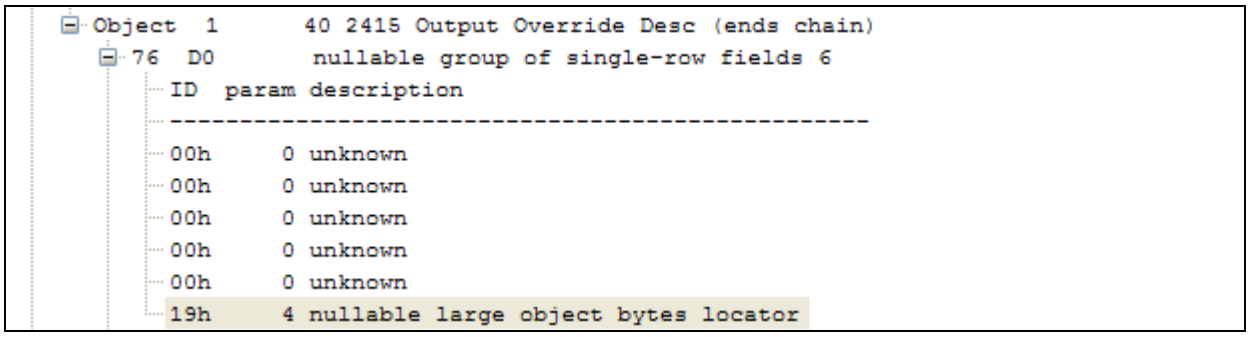
The large numbers of round-trips needed to complete some queries is confirmed by loading the group transactions. Each group transaction measures the elapsed time across all the trips involved in a query, and on the chart the markers of group transactions are enclosed in brown squares. The longest-running query involved 531 round-trips and fetched 529 rows.

```
Request     Signature:     Cont Query]
            Length:        104 bytes
            Frame:         2423
    block   length code    name
Request 1       98 2006 Cont Query (ends chain)
      len code   name                                  data
    ------------------------------------------------------
       68 2113 Package Name, Consistency Token, Section N
        8 2114 Query Block Size                  32767 = 7FFFh
        6 2141 Max Blocks Extra                  -1 (unlimited)
       12 215B Query Instance Identifier             0
Response    Signature:     Query Answer Set Data]
            Length:        48 bytes
            Frame:         2424
    block   length code    name
Object  1       42 241B Query Answer Set Data (ends chain)  rows 1
```

Every Cont Query request indicated that the client could receive any number of data blocks, but why was only one row returned? The reply to the Open Query request indicated that the server was using the fixed-row query protocol, and the last field in each row was a large object (LOB).

```
    block   length code    name
Request 1       25 1055 Sync Control (ends request)
Request 2       91 200C Open Query
      len code   name                                  data
    ------------------------------------------------------
       68 2113 Package Name, Consistency Token, Section N
        8 2114 Query Block Size                  32767 = 7FFFh
        6 2141 Max Blocks Extra                  -1 (unlimited)
        5 215D Query Close Implicit                   1
Object  2    3175 2412 SQL Program Variable Data (ends chain)
Response    Signature:     Sync Point Control Reply Data |
                           Open Query Complete Reply Message
                           Query Answer Set Desc] 16,3E,02(2),29,C9h
            Length:        111 bytes
            Frame:         2420
    block   length code    name
Object  1       12 1248 Sync Point Control Reply Data (ends reply)
Reply   2       41 2205 Open Query Complete Reply Message
      len code   name                                  data
    ------------------------------------------------------
        6 1149 Severity Code                     0 (information only)
        6 2102 Query Protocol Type                 2418 Fixed Row Query Protocol
        5 2150 Query Attribute Updatability      1 (cursor is read only)
       12 215B Query Instance Identifier             0
        8 215F Query Blocking Factor             max 4,294,967,295 rows per block
Object  2       40 241A Query Answer Set Desc (ends chain)
   76  D0        nullable group of single-row fields 6
    ID  param description
    ------------------------------------------------
    16h     8 8-byte integer
    3Eh   255 variable character mixed
    02h     4 4-byte integer
    02h     4 4-byte integer
    29h   255 nullable variable bytes
    C9h 32772 nullable large object bytes
```

The first Cont Query request included an Output Override object that requested each row provide an LOB locator rather than the LOB itself:



The inclusion of a large object, or a setting in the stored procedure, may dictate the fixed-row protocol and require that only one row can be returned with each request.

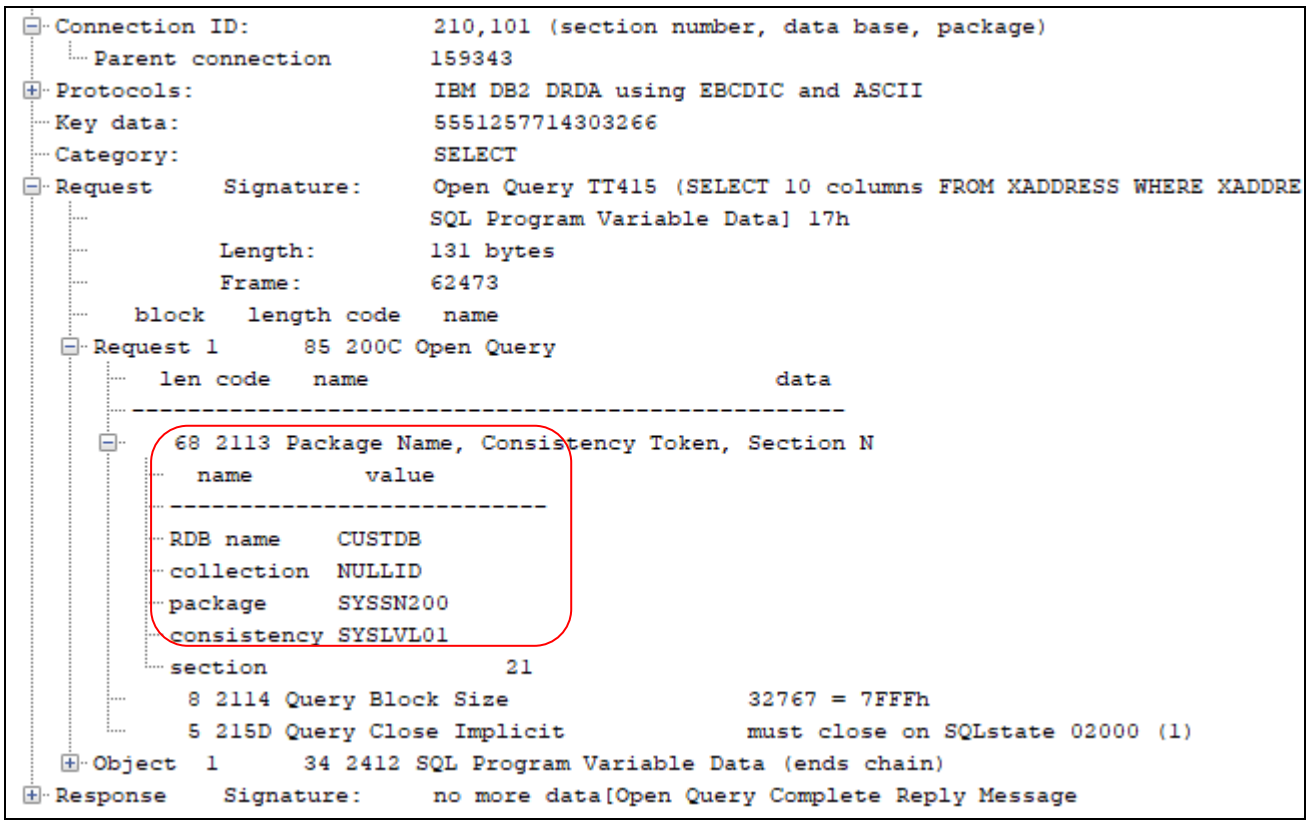
3.8.5 DB2 Queries Re-opened or Data Manipulation Commands Re-executed

DB2 database clients assign each new SQL statement to a package *section* that has a function similar to that of an Oracle cursor. NetData records the generalised version of the new SQL statement with other section-state information such as column headings for result data. If the SQL statement is executed later, with different values for its bound variables, NetData includes the generalised statement in the new transaction's signature, and assigns the statement verb (such as SELECT or UPDATE) to the transaction's category.

The more verbose signatures help in understanding the function of queries, but it is not so easy to distinguish transactions with new types of queries. Transactions that convey a new SQL statement usually start with a 'Prepare SQL statement' block, whereas re-opened queries start with an 'Open Query' block. Requests to re-execute a data-manipulation command (Insert, Update or Delete) begin with an Execute SQL block.

3.8.6 Unique IDs for Sections in DB2 Databases and Packages

When a SQL statement and its related state information is retained in a DB2 database for a query that involves multiple round-trips, or for repeated queries with different search targets, its unique identification requires the names of its database and package, and the number of its section within the package. This identifying information is specified in the first request block of relevant transactions:



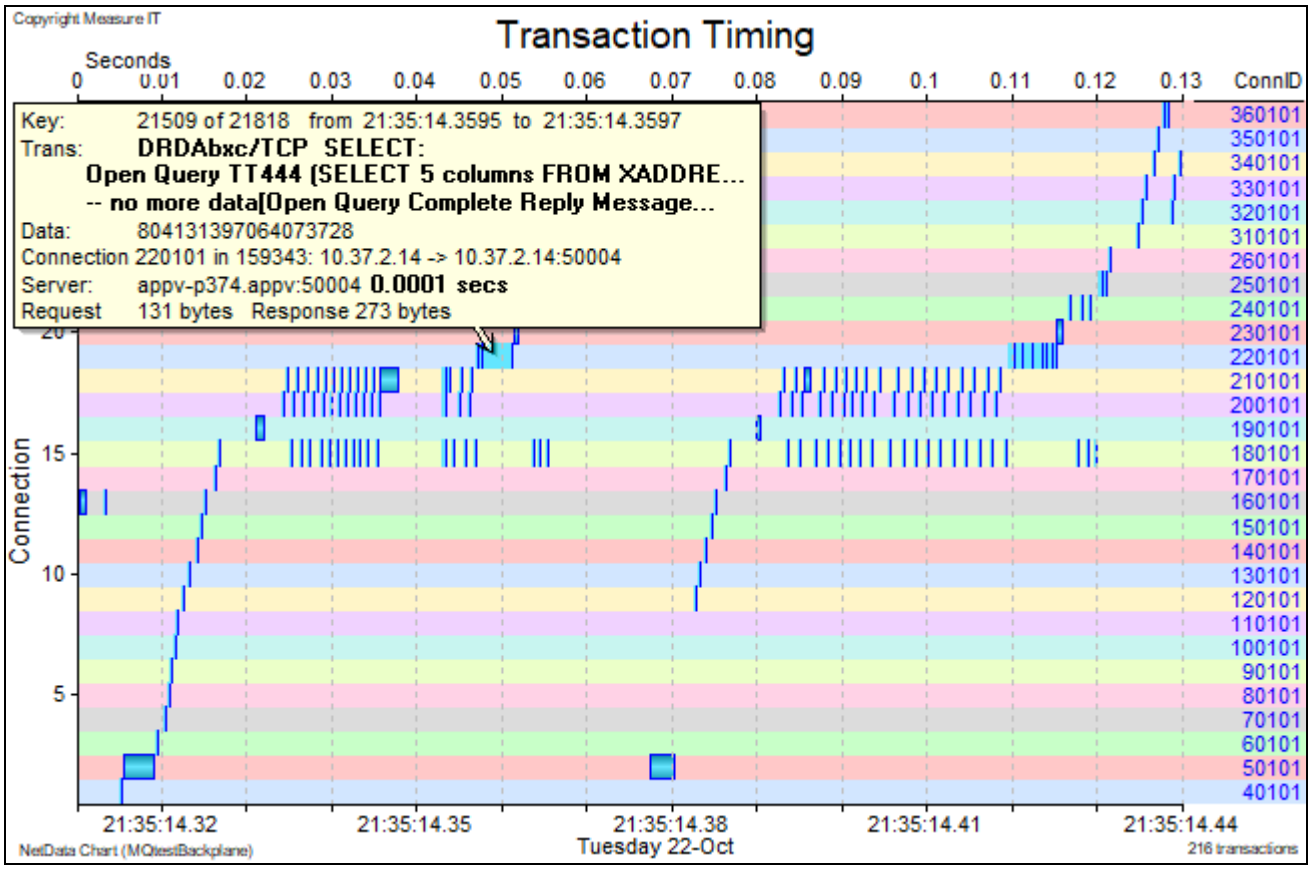
To find all the transactions related to a particular SQL statement, NetData now assigns to transactions a secondary connection ID that uniquely identifies its combination of database name, package name, and section number. NetData gives the database and package names identifying index numbers, and the secondary connection ID is calculated as follows:

<section number> x 10000 + <database index> x 100 + <package index>

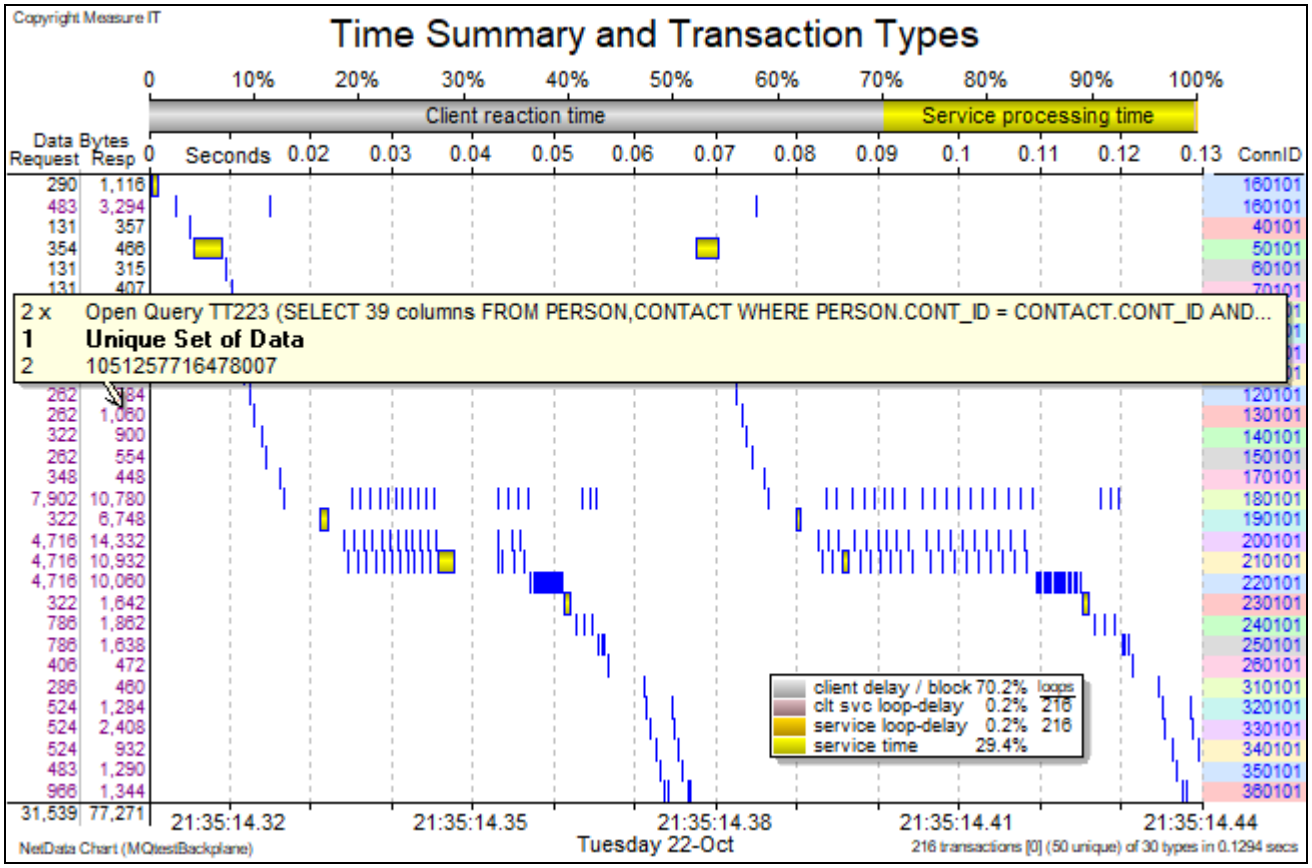
An ID of 350204 can be read as section 35 of the fourth package of the second database, making it possible to distinguish statements that belong to different databases or different packages.

A single user request may generate a large number of database transactions (forming what NetData calls a *family*) and they are all likely to be conducted sequentially in a single connection held by the application thread that handles the user request. By choosing to 'Plot Related Secomdary Connections' for that connection on the transaction timing chart it is possible to compare the activity of all the different sections in the DB2 database. If there is a large number of secondary connections their details may be better viewed by scrolling across them vertically. Vertical scrolling is enabled by the 'VrtScroll' button.

If the connection ID displayed on the end of a row appears in blue, it is its composite section number, as in the following timing chart:



A more thorough view of repeated transactions and the occurrence of identical queries is provided by the waterfall chart which dedicates each row to a different transaction type:



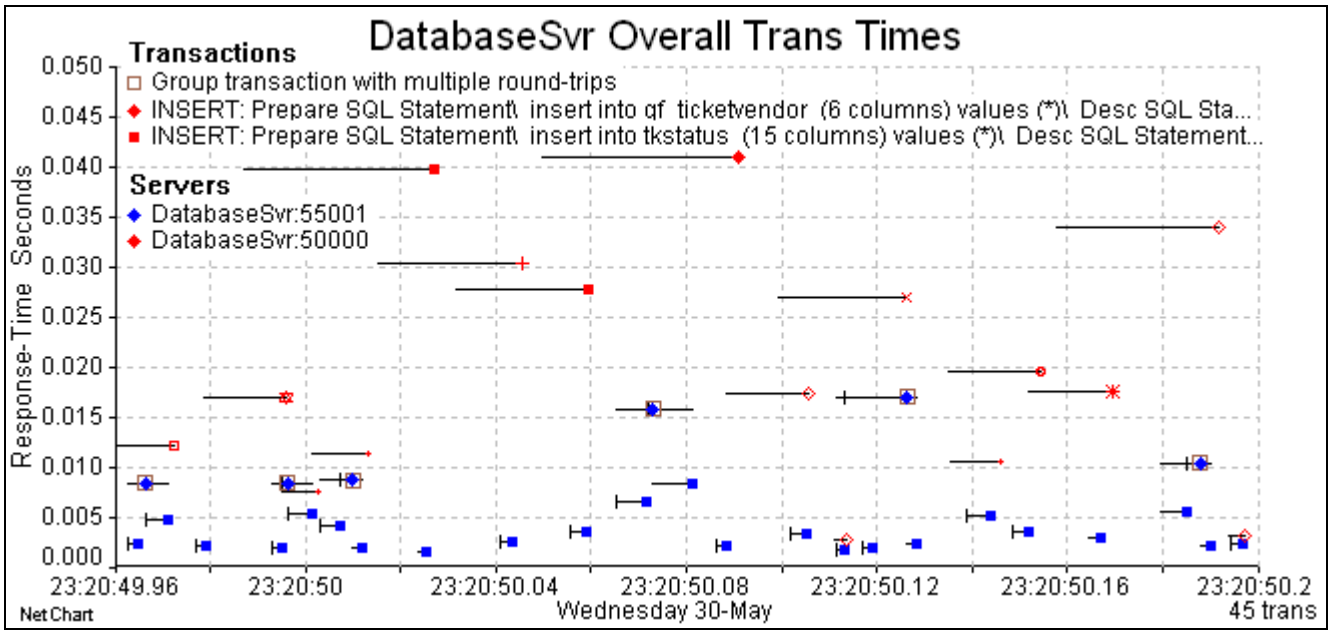
3.8.7 DB2 Log Shipping for High Availability Disaster Recovery

DB2's High Availability Disaster Recovery (HADR) feature ships log buffers directly from the primary database to an HADR standby database, and the standby replicates the primary by replaying those log records. HADR log-shipping traffic normally uses TCP port 55001, and all log buffers are output in one or more blocks with a fixed size that is normally 4096 bytes.

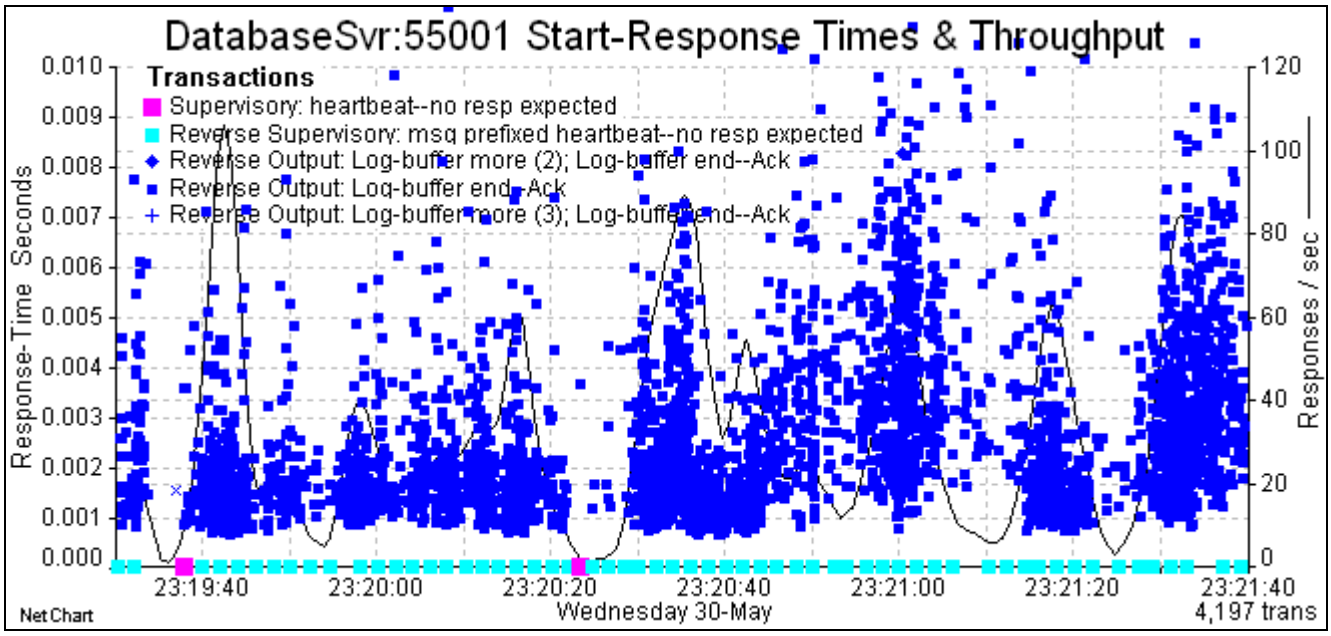
NetData decodes the headers of message blocks and reconstructs round-trips as *server* transactions. Some database changes generate several log-shipping round-trips and NetData characterises each set of related round-trips as a *group* transaction.

If the database behind port 55001 is the primary database then the connection's server initiates the round-trips and NetData describes them as 'Reverse Output' or 'Reverse Group' transactions.

NetData's transaction performance chart is able to show the direct relationship of HADR transactions with Insert, Update, Delete and other transactions that change the database (as defined in IBM Redbook *sg247363.pdf*). A connection between partner databases can be in any one of three log-shipping modes – Sync, NearSync and Async – and in the first two modes the primary will delay Commit transactions until it has received an HADR Ack signal from the standby. A performance chart can indicate whether HADR has affected database performance.



The Insert transactions marked in red on this chart are accompanied by a log-buffer shipment and Ack (in blue) that ends just before the start of the Insert response. Some shipments required two or three consecutive round-trips, and the markers indicating groups of trips are enclosed by brown squares. The standby database had exhibited some stress and on this chart the longest time to complete a group of log shipments was 17 ms.

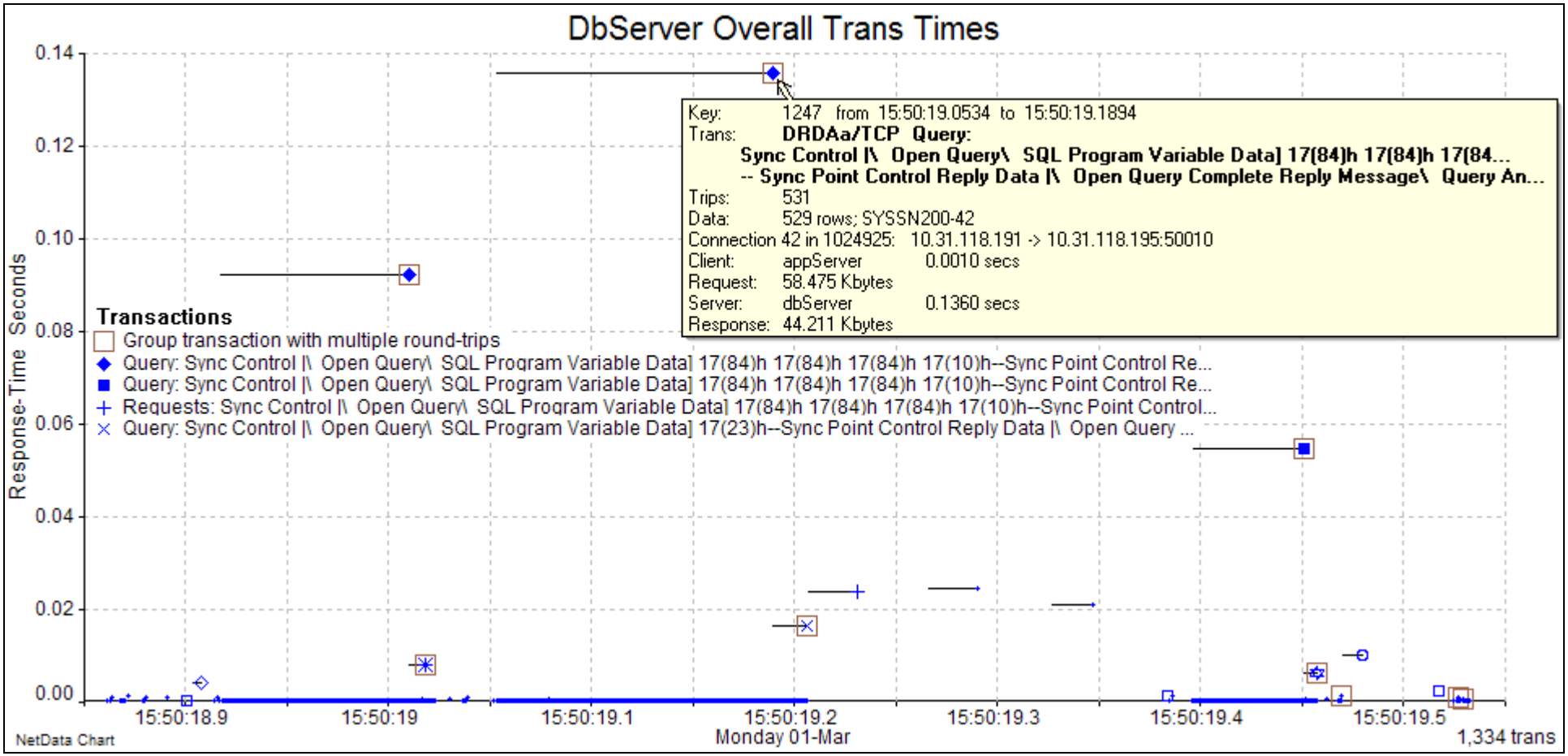


Over this two-minute period the bottom of the band of markers for log-shipment round-trips occasionally rose by a millisecond, indicating a small degree of congestion and stress in the standby database, yet there was no apparent correlation between log-response times and the logging rate.

The markers on the x-axis indicate the occurrence of heartbeat messages. This primary database issued heartbeats at two-second intervals but the standby issued heartbeats sporadically because Ack messages also serve the purpose of heartbeats.

3.8.8 DB2 Group Transactions

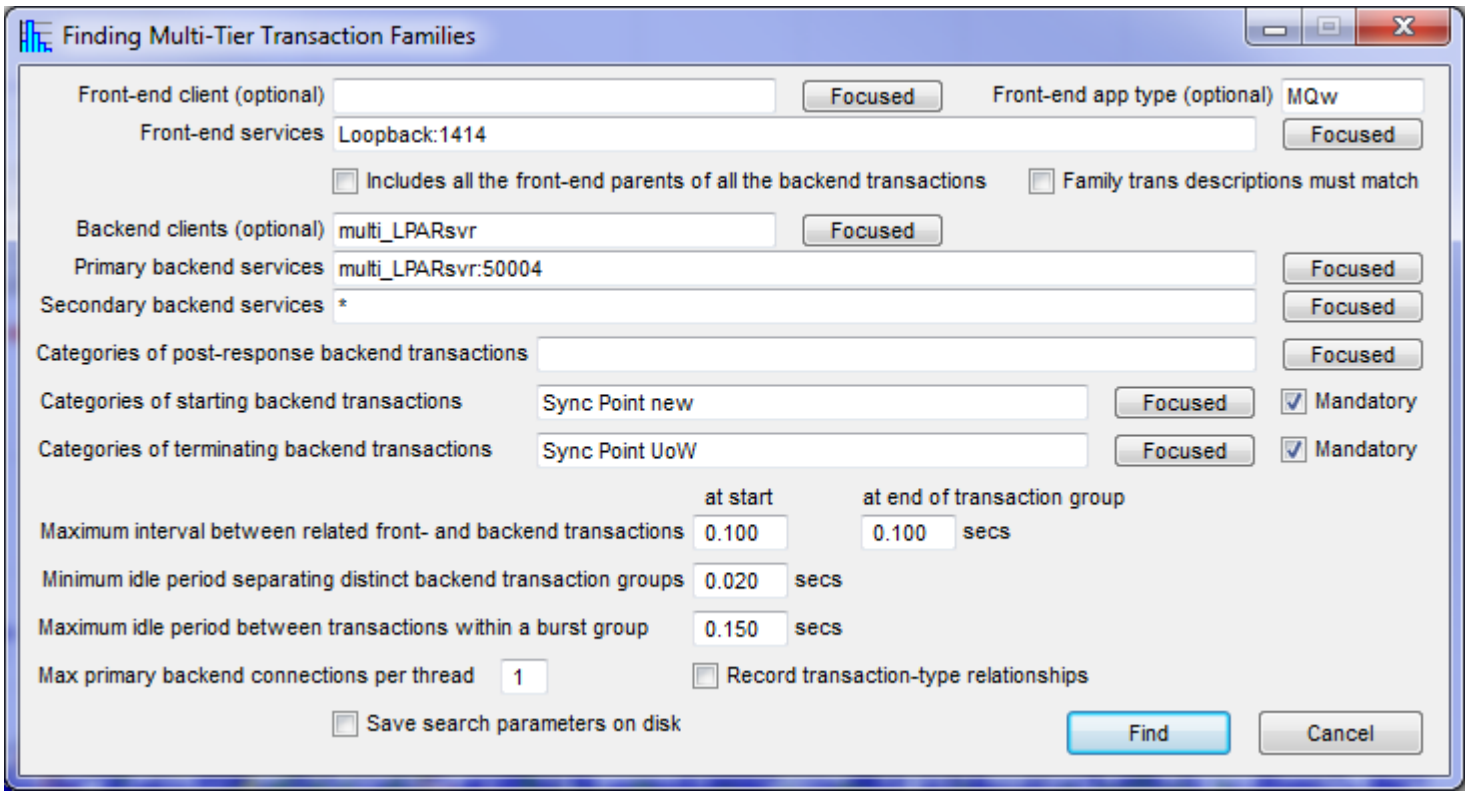
Some user transactions take a long time not because individual round-trips take a long time but because they involve large numbers of round-trips. Such behaviour is often seen in the traffic of Oracle and DB2 databases when the client fetches a large result data-set only one row at a time. NetData's DB2 decoder now characterises *group* transactions that measure the elapsed time across all the trips involved in an individual query. When plotted on the performance chart they give a more realistic view of performance problems and help to explain the purpose of long streams of round-trips.



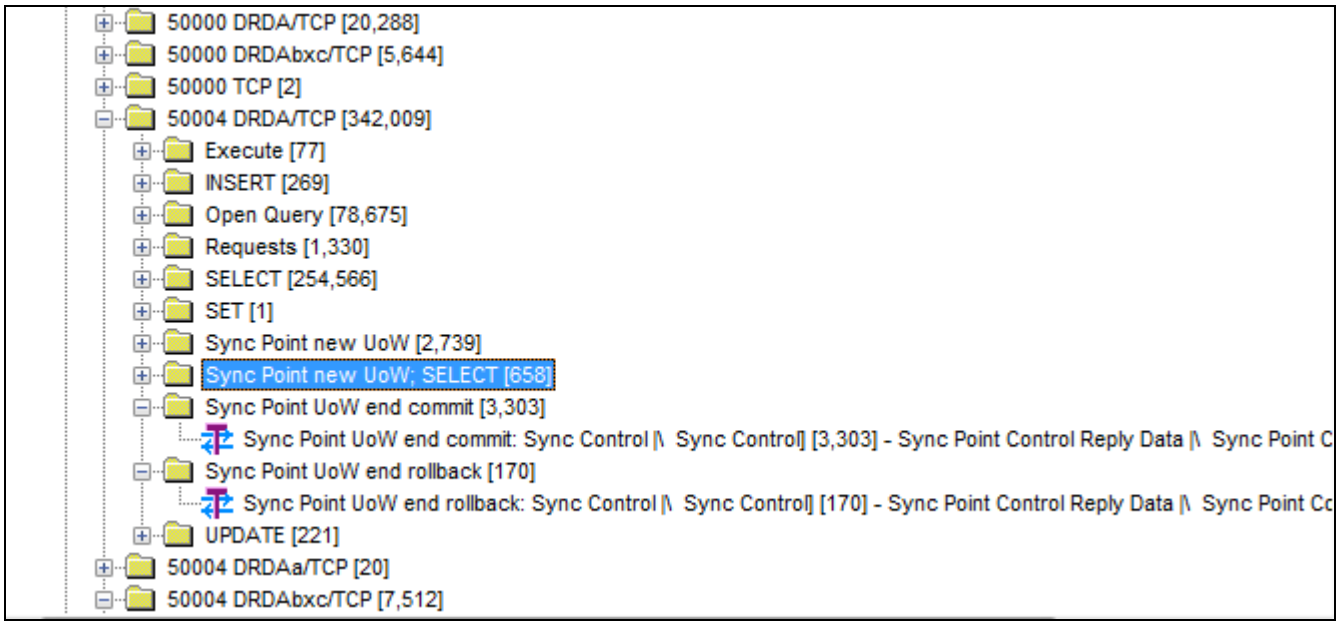
This one user transaction involved 1334 round-trips but only a few queries. The markers enclosed in brown squares show the response times of group transactions – complete queries. The longest-running query involved 531 round-trips and fetched 529 rows.

3.8.9 DB2 Sync Point Control

The DB2 decoder decodes sync-point control types and includes this information prominently in the transaction category. Because phrases such as 'Sync Point new UoW' and 'Sync Point UoW end commit' appear in transaction categories they can be specified in the controls for finding multi-tier transaction families. Some application threads will start the processing of a front-end transaction by issuing to the database a sync-point request specifying a new unit of work, and end the processing with sync-point requests that end the unit of work and require either a commit or a rollback.



The following extract from a transaction tree indicates the presence of sync-point control requests, including those signalling a new unit of work that are included in the same message as the first request for a database query:



3.9 IBM Business Monitor and DB2 Monitor

NetData recognises two application protocols believed to be involved in the management of DB2 databases. The name of the application tagged as 'DB2mon' is not known, but some of its transactions appear to enumerate the table spaces of a database manager as in the following fragment of a response message:

PROFILEOID	1463139617.1.7052
RESULT	db2inst2,JPM_AC,0,SYSCATSPACE
	db2inst2,JPM_AC,0,SYSTOOLSPACE
	db2inst2,JPM_AC,0,USERSPACE1
	db2inst2,JPM_AC,0,JPMAC_STREAM_TS32K
	db2inst2,JPM_AC,0,JPMAC_STREAM_IS4K
RESULT_COMMENTS	TaskName = EnumTableSpaces
TableSpaceType	ALL
TaskName	EnumTableSpaces
TbspNames_List	*

The request and response messages are lists of name-value pairs preceded by a length indicator and a transaction ID. The application wastes packets because the length indicator is not buffered with the message and is usually sent in its own packet, but this inefficiency is not significant if the transactions are confined to a server's loopback path. Names and values are ASCII text strings also preceded by length indicators, and NetData displays them in two columns as in the above message fragment.

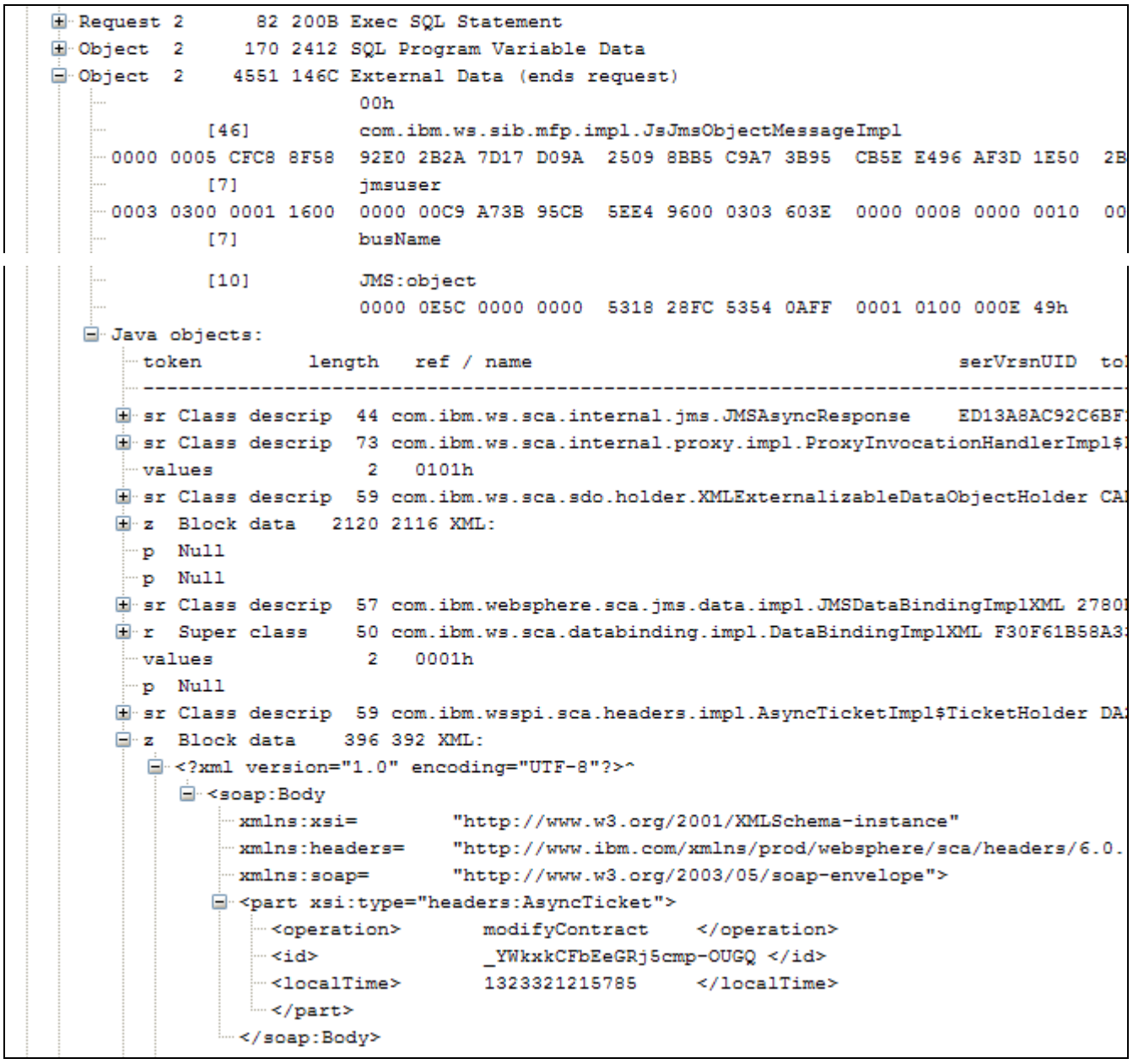
The traffic tagged as 'BusMon' is believed to belong to a version of IBM Business Monitor. It has been seen in a loopback connection using TCP port 2020, a possible default port number assigned by Business Monitor to the second in a set of cluster groups. Request and response messages consist of a single, serialised Java object such as `tivoli.monitoring.shared.proxy.RequestObject`. They have no length indicator or any other message-framing device, although the serialised object streams end with a Reset token (character y). The application is particularly inefficient in that it doesn't buffer its output, with the result that a typical message of 2834 bytes is spread across 119 packets, many with only one or two characters that are Java stream tokens.

3.10 Java Objects and XML in DB2 Transactions

Large strings of XML code and serialised Java objects are usually conveyed to and from a DB2 database in a DDM External Object (codepoint 146Ch). NetData searches such objects for strings of Java and XML code and decodes those strings with its normal Java and XML decoders.

The first benefit is that message content is more easily read and understood when displayed in a structured form as part of a transaction description. The second benefit is that NetData's XML decoding controls can request extraction of the values of named fields, to augment the transaction's signature, its user ID, and its key data. When front-end and backend database transactions are loaded into the charting module it is possible to search the data column of the transaction table for key identifiers such as requestID and customerID that relate backend transactions to their front-end transactions. By tracking the flow of transactions through multiple tiers it is possible to explain how slow backend systems delay particular user transactions.

The DB2 decoder extends these capabilities to XML strings found within Java objects within database fields as in the following example:



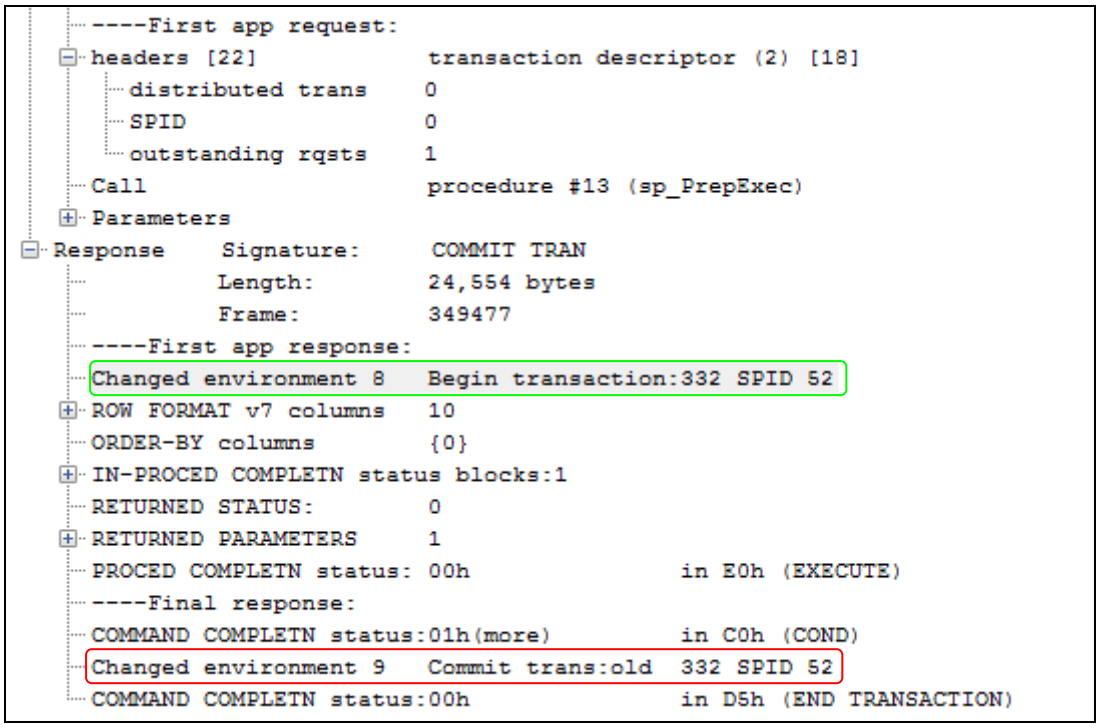
3.11 Distributed Transactions

A database distributed transaction is a single, logical operation that comprises a group of one or more data-manipulation operations in one or more database servers that as a whole has ACID properties – properties of Atomicity, Consistency, Isolation, and Durability. To achieve atomicity a transaction manager supervises all the operations and at their completion, or when an error occurs, issues commands to either rollback or commit all the changes.

NetData tracks Microsoft and IBM DB2 distributed transactions and creates a *group* transaction that records the overall time-span of the distributed transaction, its number of round-trips and the totals of the lengths of its request and response messages. It doesn't need to record all its constituent server transactions because after a right-click on a group transaction in any context NetData will offer to load all its server transactions and optionally display them in a waterfall chart.

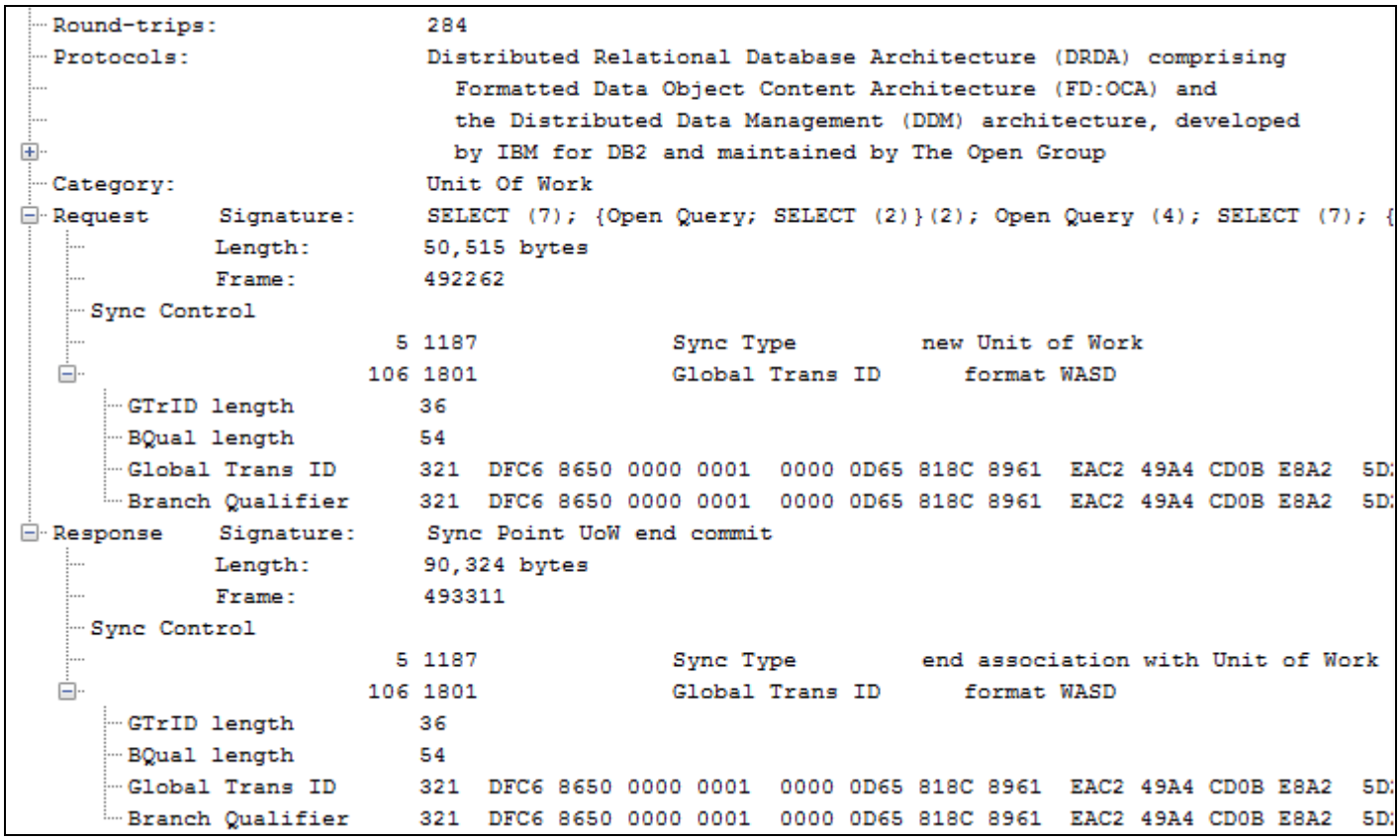
Because a distributed transaction is unlikely to be associated with more than one user transaction, and because it represents many round-trips, it helps in viewing overall system performance and in identifying families of backend transactions. Charts with only distributed transactions can give a broader view of system performance, covering a wider time span than is possible with server transactions.

A SQL Server distributed transaction is often started implicitly by the database manager, with details conveyed in a changed-environment message in a command response:

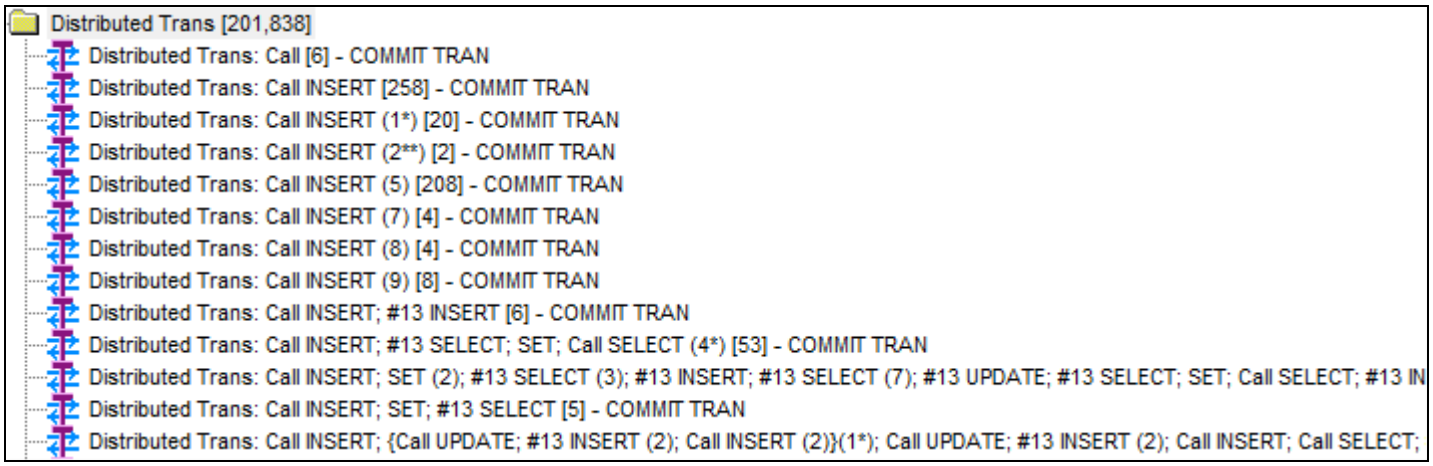


As illustrated by this panel NetData's description of a distributed transaction includes the request and response messages of its first round-trip and the response message of its last round-trip. This example concerns a distributed transaction with ID of 332 that is unique only to the particular connection (session). All but the first database request involved in this transaction specify its number in their header's transaction descriptor. The final request specifies either Rollback or Commit, and the action is confirmed by a changed-environment message in its response.

A DB2 database refers to distributed transactions as Units of Work (UoW), and a connection with the database can be associated with only one UoW at a time. A client can suspend a UoW on a connection by issuing a Sync Point control command to end the connection's association, and then introduce a new or another UoW, but most units of work end with a single round-trip conveying two Sync Point commands, one to end the connection's association and the other to request a rollback or commit. A Sync Point command that starts or ends a UoW conveys a global transaction ID that typically requires 36 bytes.



This description of a Unit of Work contains code points (message blocks) from the first and last round-trips and confirms that they refer to the same global transaction ID. It also tells there were 284 round-trips and ended with a commit. The request signature summarises the pattern of database manipulations by listing the SQL verbs, but compresses the list by giving numbers of repetitions in parentheses. NetData also notes repeated cycles of different verbs by enclosing the cycle in braces and giving a number of repetitions after the closing brace. In the example above the UoW began with 7 Select statements and then repeated a cycle twice; that cycle comprised an Open Query (to execute a previously defined Select statement) and two Select statements. The sequence continued with 4 Open Queries, 7 Select statements and another repeated cycle.



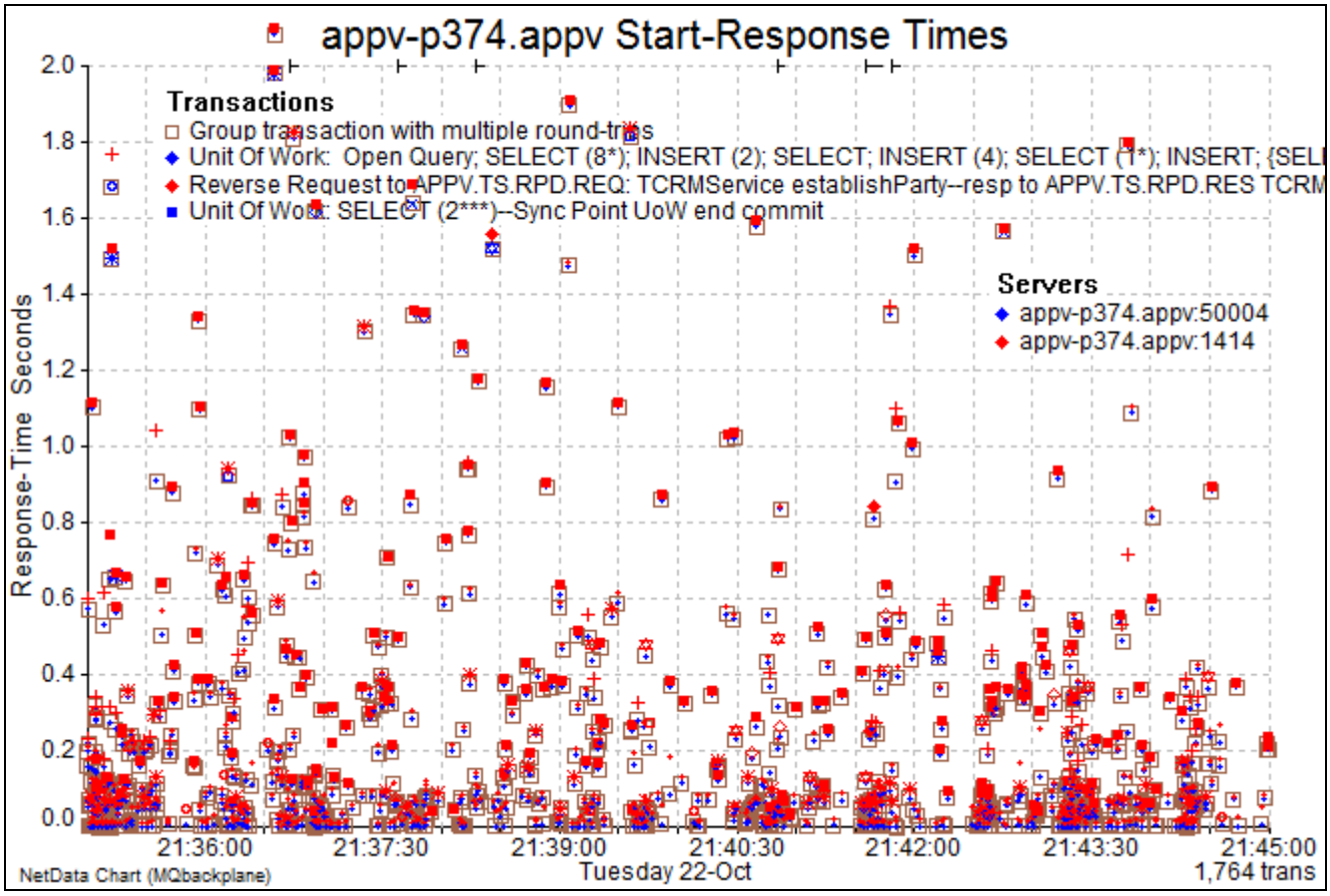
The panel above is part of a transaction-class tree that lists some different types of distributed transactions handled by SQL Server. Calls to system stored procedures identified by numbers are abbreviated to the number and the SQL verb of the relevant statement. An item such as '#13 SELECT (3)' signifies 3 calls to procedure 13 to prepare and execute a Select statement. A call with a SQL verb and no procedure number implies a call to sp_execute (procedure #12) to execute a previously

prepared statement. The last distributed transaction in the above list repeated a cycle that began with execution of a prepared Update statement, then prepared and executed 2 Insert statements and executed 2 prepared Insert statements.

{Call UPDATE; #13 INSERT (2); Call INSERT (2)}(1*)

The number of cycle repetitions was between 10 and 19. Calls to sp_unprepare are excluded from these descriptions. The numbers in square brackets are numbers of round-trips.

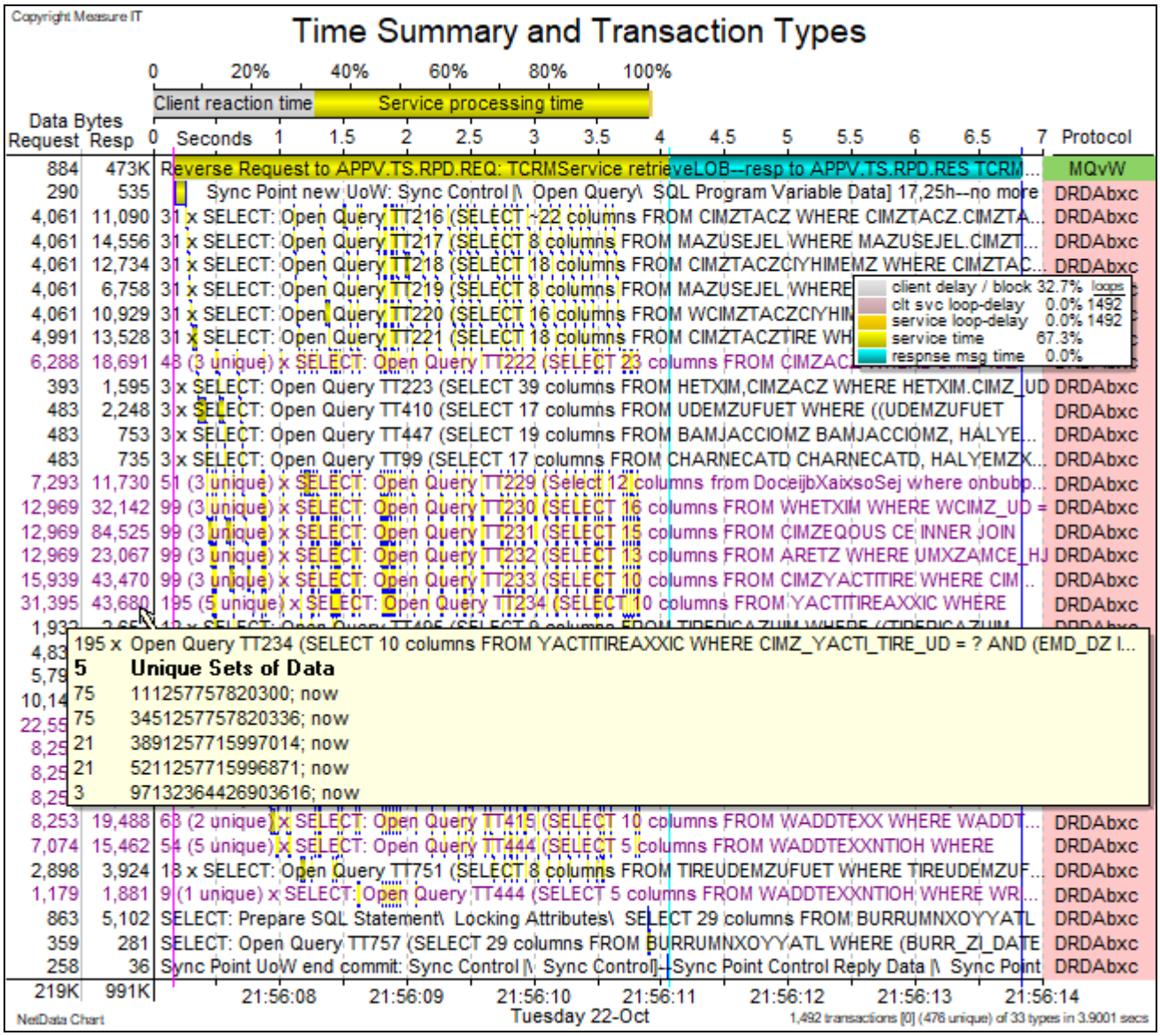
The following chart displays the time spans of Units of Work with blue markers, in grey boxes to indicate that they are group transactions with multiple round-trips. The red markers plot the server response times of WebSphere MQ application transactions, and their coincidence with the blue markers indicates that each application transaction became a single Unit of Work in the database. It illustrates how a plot of distributed transactions can often provide a reliable guide to the performance of user transactions without capturing those transactions. A problem investigation can begin with a waterfall chart of a long-running distributed transaction, to see whether most of the time is spent in the database server or its client, and to look for redundancies in the execution of queries and other operations.



3.12 Recognising Redundant Queries Dependant on Current Time

User transactions often generate a large number of database queries that cycle through a small number of query types, and sometimes the queries of a particular type repeat a small number of search targets. NetData is able to quantify the extent of such redundancy but common sets of search variables are more difficult to recognise if they include the time of the query. The current time may be required, for example, to find only those records with an expiry date in the future.

When displaying a table of search targets NetData now replaces timestamps that are closely related to the transaction's start time with a brief phrase such as 'now' or 'now-60min'.



In this example one type of query was executed 195 times. The search targets used only five different ID numbers but every query included a slightly different search time. Because the word ‘now’ was substituted for every time value NetData was able to report that there were only five unique queries. This user transaction generated 1482 database transactions of which only 476 were unique.

3.13 MySQL

NetData measures and characterises MySQL database transactions, extracting SQL statements and tables of query-result datasets. The decoder has been tested with MySQL versions 4.0, 4.1, 5,0 and 5.1. It can parse very large request and response messages; to display large objects (blobs) found in fields of dataset rows; and display the structure of XML blobs in datasets. It handles Show statements and Limit clauses in Select statements. It extracts constants from Where clauses and displays them in the data column of the transaction table.

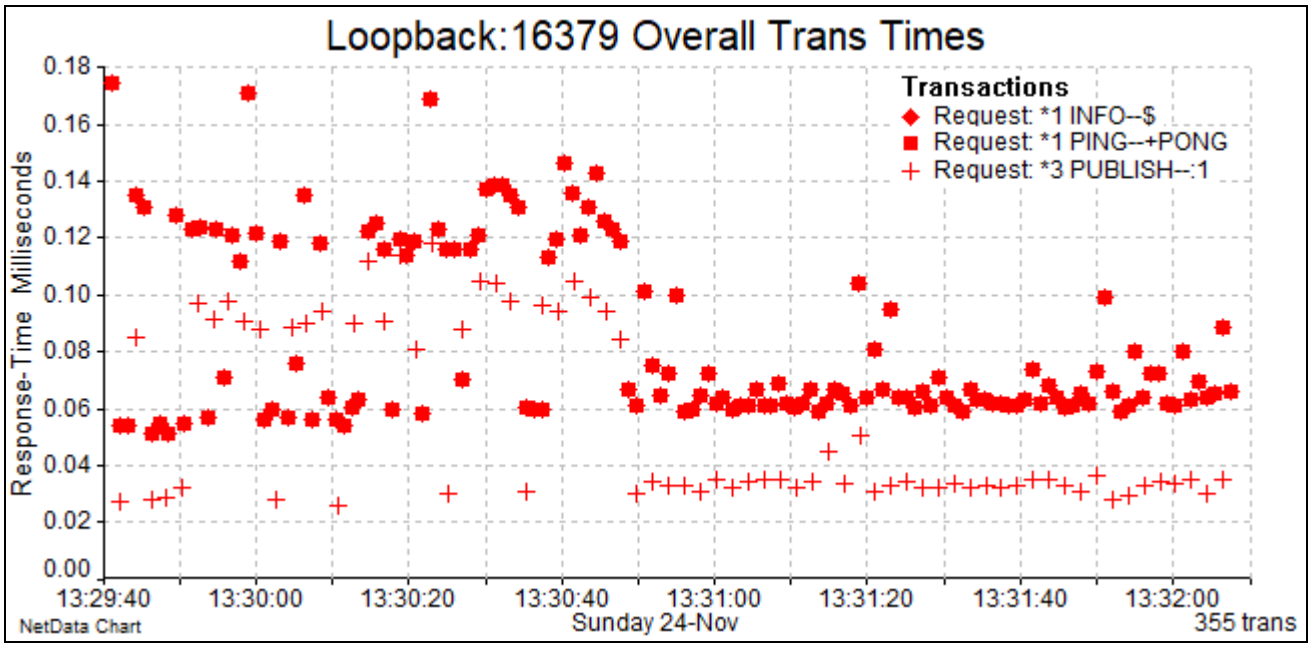
The decoder characterises a connection’s initial handshake that is initiated by the server, and handles the adoption of SSL (Secure Socket Layer) after a handshake.

Error messages sent by the server and referring, for example, to SQL syntax errors, non-unique keys and deadlocks, are recorded as application errors in NetData’s table of network events.

For MySQL, NetData's SQL parser now handles Show statements and Limit clauses in Select statements. The MySQL decoder also now extracts constants from Where clauses and displays them in the data column of the transaction table.

3.14 Redis Inter-Node Traffic

NetData decodes inter-node traffic within clusters of servers running a Redis in-memory key-value database or cache. Redis is an open-source project found in a variety of applications and normally uses port numbers 6379 (for client commands), 16379 (for the cluster bus) or 26379. For high performance Redis pipelines transactions between nodes and allows commands to be pipelined.



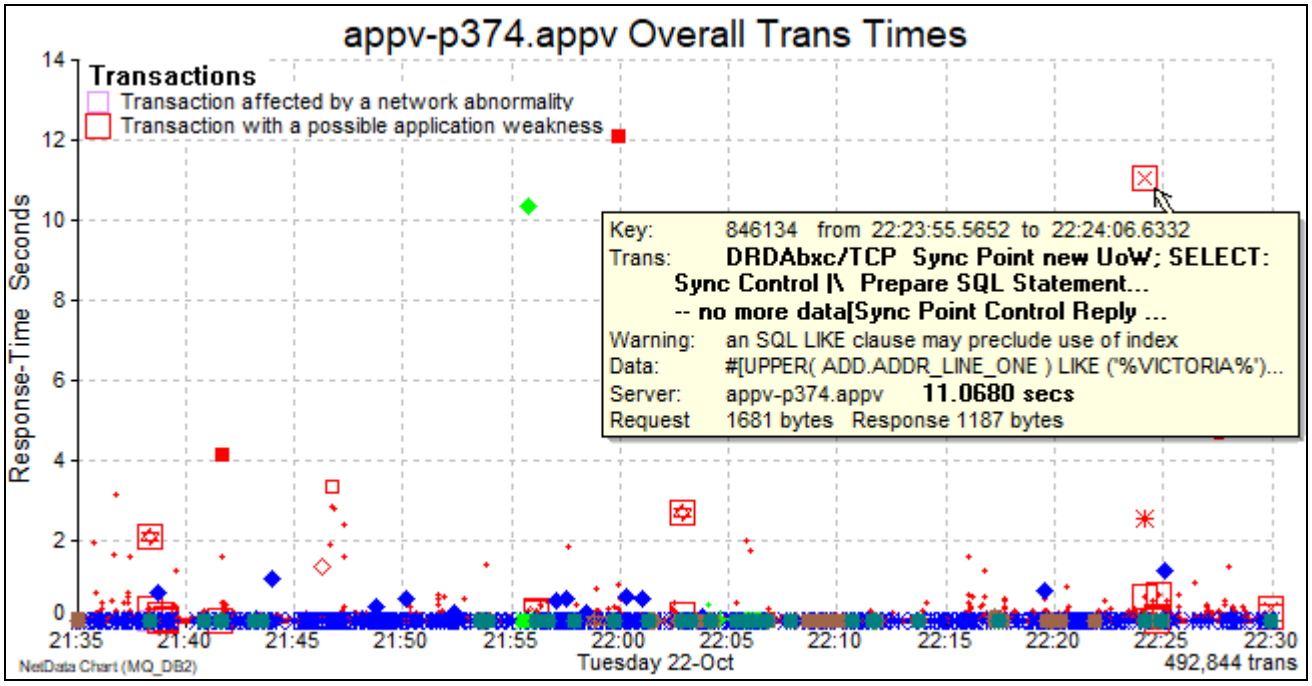

```
Protocols:                  Inter-node traffic of Redis in-memory
                              database or cache system
Category:                   Request
Request     Signature:      *3 PUBLISH
            Length:         143 bytes
            Frame:          5356
  data                      *3 PUBLISH
    $18
    __sentinel__:           hello
    $94
    127.0.0.1,26379,ab589a57796ab02504b7453c21005d0a9910d48a,0,quota-enforcement,127.0.0
Response    Signature:      :1
```

```
Request     Signature:      *1 INFO
Response    Signature:      $
            Length:         2,221 bytes
            Frame:          2556
  data [2214]
    # Server
    # Clients
    # Memory
    # Persistence
    # Stats
    # Replication
    # CPU
    # Cluster
```

3.15 SQL Like-Clause Performance Weaknesses

If an SQL Like clause refers to a search target that begins with a wildcard such as '%VICTORIA', the search engine can't make full use of an index and the resulting table scan may produce very large response times. If NetData sees such a Like clause it records the occurrence as a network event and flags the transaction so that when its response-time marker is plotted on the performance chart the marker will be enclosed in a red box.

Start	Category	Description
21:39:07.249806	Applicatn	SQL LIKE clause may preclude use of index in [UPPER(ADD.ADDR_LINE_ONE) LIKE ('%HIGH%')
21:41:37.15038	Applicatn	Error: SQLstate = 56098 (error during implicit rebind or prepare) (SQLNP012); SQLcode = -727 \ name ...
21:41:37.15038	Applicatn	Error: SQLstate = 56098 (error during implicit rebind or prepare) (SQLNP012); SQLcode = -727 \ name ...
21:41:37.15038	Applicatn	Error: SQLstate = 42601 (SQLNP012); SQLcode = -7 \ name value\ SQLcode -7\ SQLstate 42601\ S...
21:41:37.15038	Applicatn	open-query failure in CUSTDB
21:56:06.228923	Applicatn	SQL LIKE clause may preclude use of index in [UPPER(ADD.ADDR_LINE_ONE) LIKE ('%COOPER%')
22:02:49.91259	Applicatn	SQL LIKE clause may preclude use of index in [UPPER(ADD.ADDR_LINE_ONE) LIKE ('%TENNYSON...
22:23:55.565229	Applicatn	SQL LIKE clause may preclude use of index in [UPPER(ADD.ADDR_LINE_ONE) LIKE ('%VICTORIA%')



The search for `%VICTORIA` took 11 seconds by a database that completed 90% of its transactions within half a millisecond.

NetData also flags Like clauses whose search target is a bound variable. In such cases the search engine is likely to produce an execution plan to handle all possible values of the variable without knowing the position of any wildcard, and the query is unlikely to use an index.

The system designer probably can't avoid table scans when users submit search targets with an initial wildcard and NetData's warning serves only as a likely explanation for large response times. However, the designer should avoid bound variables in Like clauses because the very small saving in execution planning is likely to be heavily outweighed by long delays for all search targets.

NetData flags Where clauses that call uncommon procedures because they too may preclude the use of indexes.

3.16 Progress Software Database and Application Servers

NetData decodes and reconstructs transactions of two Progress Software protocols, one for the application server, and the other for the OpenEdge relational database management system (RDBMS). The application decoder reconstructs the tables of parameters and data in request and response messages and includes a called-procedure name and a list of returned data-set field names in the transaction's request signature.

The database protocol is particularly chatty and can take 10 round-trips to fetch each record. NetData is only able to separate text from the binary codes in database messages.

In both protocols some types of messages do not participate in round-trip transactions.

3.17 Teradata Unity and Database Traffic

The decoder for Teradata database transactions can characterise the messages and transactions of a different Teradata protocol that has been seen using TCP ports 5344, 5347 and 5348. This protocol is believed to be associated with Teradata Unity Data Mover servers.

NetData also recognises traffic of Apache ActiveMQ that provides a messaging service for Data Mover.

4 File Service

4.1 Server Message Block Version 2 (SMB2)

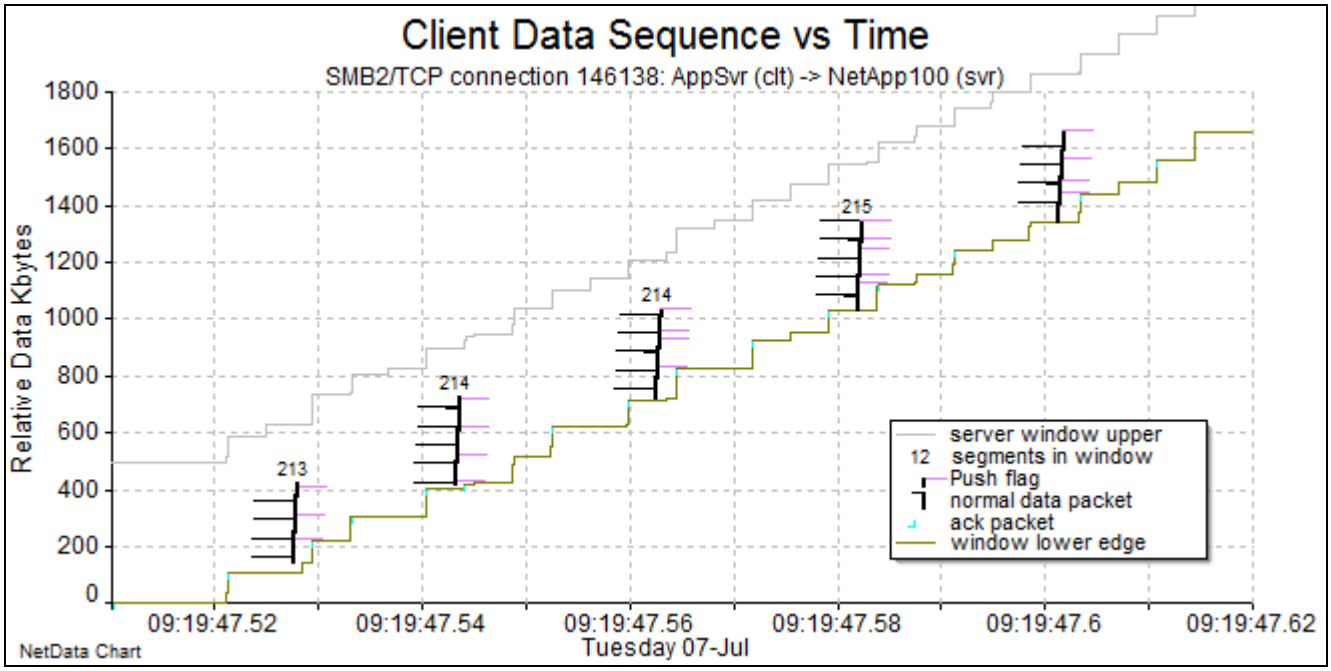
The SMB2 decoder fully parses most messages and reconstructs all transactions including OpLock break notifications, acknowledgements and responses. It also reconstructs RPC transactions conveyed in named pipes implemented with the pipe-transceive type of File-Service Control (FSCTL) command.

NetData also summarises file-IO activity with group transactions that span the life of an open file. The response times of these pseudo transactions start with their SMB2 command that opened the file and end either when the file was closed or when the file was last accessed before closure. The latter option produces better charts to indicate file activity because the workstation delays file closing in order to cache file handles for file-access later.

Like SMB, SMB2 is able to chain many commands or many responses in the one message block, all referring to the same file. Windows can open and close a file in one round-trip, to read the file's timestamps, attributes and size. It may also open a file and issue several Get Info commands in a single trip. When constructing file-IO group transactions NetData takes chaining into account when counting round-trips.

4.2 Signed SMB Messages

All versions of the SMB (Server Message Block) protocol allow commands and responses to be signed with a hash of the contents, to avoid tampering with data by a 'man in the middle'. The consequential heavy processing burden on clients and servers can seriously degrade throughput. There will be no sign of any network problem to explain the poor performance, other than indications that the send window often becomes empty, or the receiving node limits the flow by closing its receive window or delaying the issue of acknowledgements, as in this sliding-window chart of SMB Write commands received by a NetApp file server:



Although the server received very rapid bursts of up to 215 packets, at intervals around 20 ms, there was a significant delay before all the received data was acknowledged. These delays confirm that the transfer bottleneck was in the NetApp server, and disabling SMB signing raised throughput from less than 150 Mbps to more than 1000 Mbps. With signing enabled, ack packets were issued

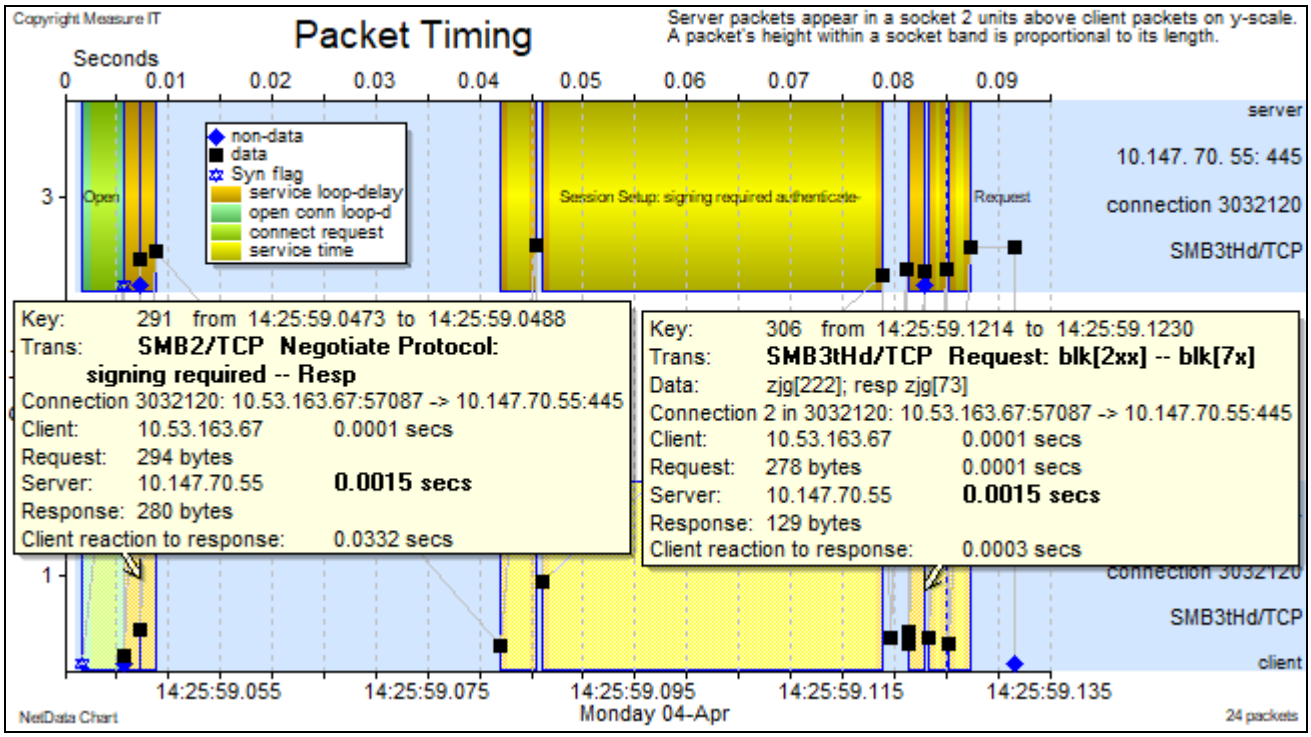
in small bursts at intervals of approximately 3.6 ms, and each burst acknowledged all the data received between TCP push flags (indicated by right-pointing purple ticks). The push flags did not correspond to packets with the end of a 65KB block of data in a Write command (left-pointing black ticks) – the packets that should trigger the CPU-intensive calculation and checking of a block signature.

NetData highlights the presence of signed messages by adding the word 'Signed' to the names of affected Read and Write commands, and inserts 'signing required' in the descriptions of Session Setup and Negotiate messages that flag the requirement for signed blocks.

4.3 SMB2_Transform_Header and SMB2_Compression_Transform_Header

The family of SMB version 3.x dialects supports a new header for encrypted messages. Known as the SMB2_TRANSFORM_HEADER, it follows the standard SMB length indicator and comprises a 4-byte protocol ID of FDh + 'SMB'; a 16-byte signature; a 16-byte nonce; the original message size (4 bytes); 2 null bytes; 2 bytes for flags or encryption indicator; and an 8-byte session ID.

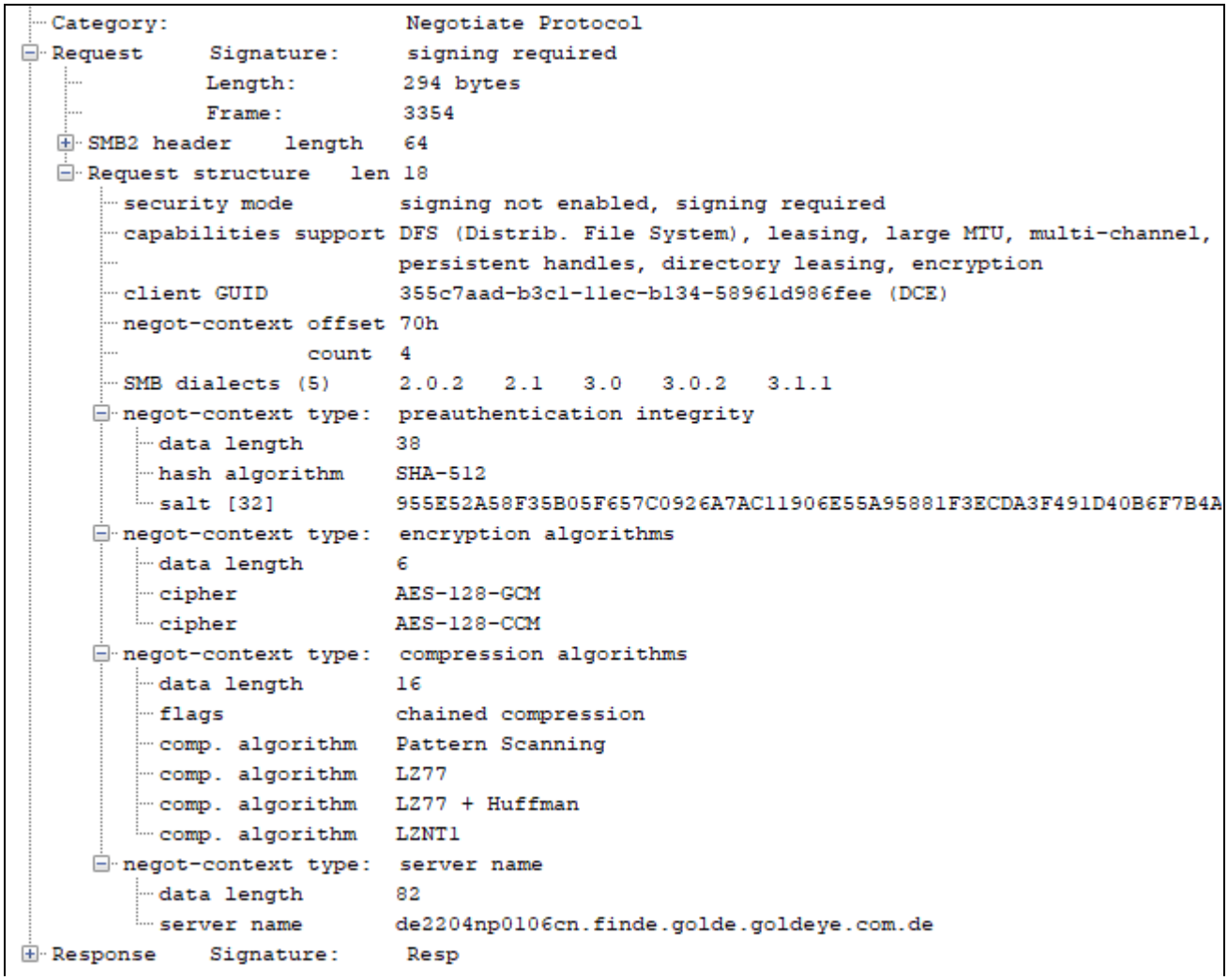
NetData decodes these headers and separates concurrent message exchanges into transactions according to their session IDs. Because all message contents are fully encrypted, including an optional compression header and the standard SMB2 header, the SMB message types are unknown and NetData characterises transactions according to the lengths of their request and response messages, much as it does for TLS transactions. For example, a request or response signature of **`blk[2xx]`** indicates a message whose payload length is in the range from 200 to 299 bytes.

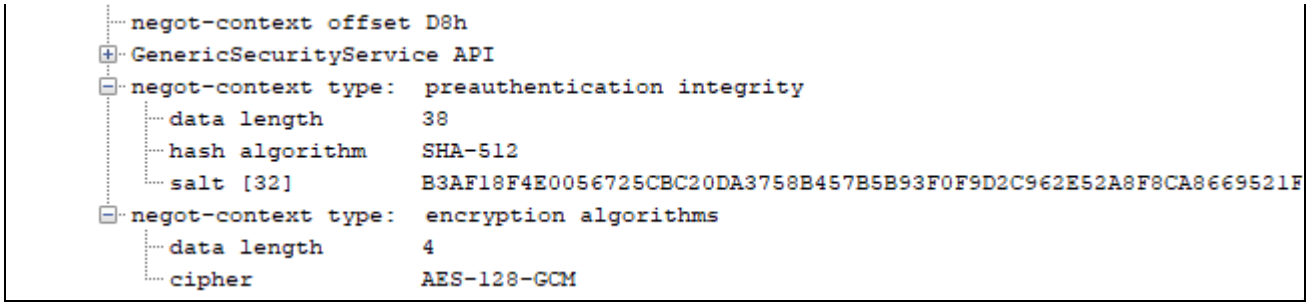


SMB version 3.1.1 also supports a compression header known as SMB2_Compression_Transform_Header_Unchained. It comprises a 4-byte protocol ID of FCh + 'SMB'; the original compressed

segment size (4 bytes); 2 bytes to indicate the compression algorithm; 2 bytes for flags; and 4 bytes for the offset from the end of the header to the start of the compressed data segment in the payload. NetData decodes this header and then characterises transactions as it does with transform headers.

The types of encryption and compression are established in a Negotiate Protocol transaction, and signing is requested by the security mode in a Session Setup transaction.



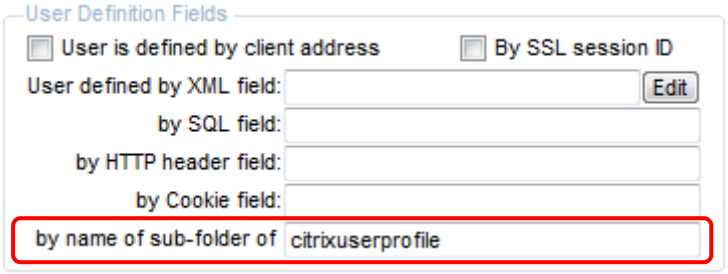


4.4 Forming User Transactions

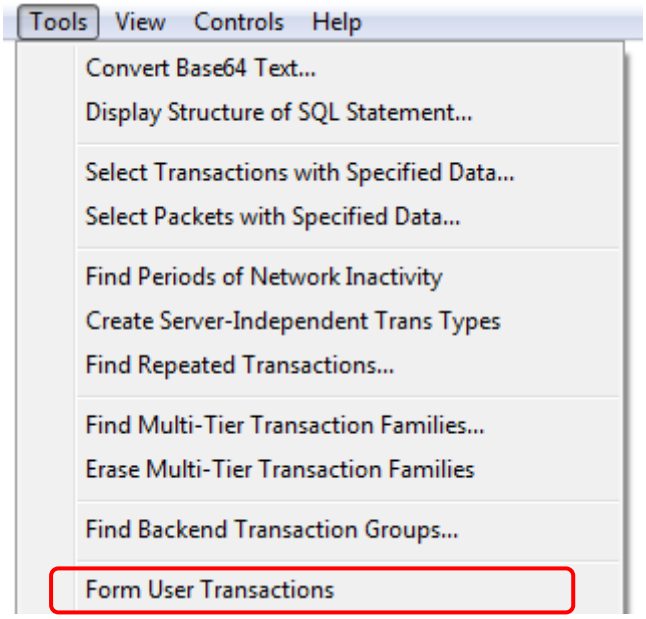
A control on the Decoding page allows NetData to associate file-IO group transactions with individual users by extracting user names from file path names. For example, a Citrix terminal server might allocate folders for user profiles as sub-folders within the folder `gg\citrixuserprofile` on a file server. During a Windows Login transaction, when the client reads registry contents from the file

`gg\citrixuserprofile\johndoe\Pending\UPM_Profile\NTuser.dat`

NetData extracts the name of the user's folder 'johndoe' and assigns it as a user name to the file-IO transaction, a *group* transaction that characterises all SMB transactions needed to open, access and close the file. It is not practical to identify users any other way because many users are able to read profiles from the file server at the same time, all from the same Citrix server and using various TCP connections.



Although NetData can identify a user's *group* transactions during analysis, in this case it can't form complete *user* transactions. A complete Login transaction might read hundreds of profile and data files, taking more than a minute, and to characterise such transactions NetData has a new tool that runs after analysis:



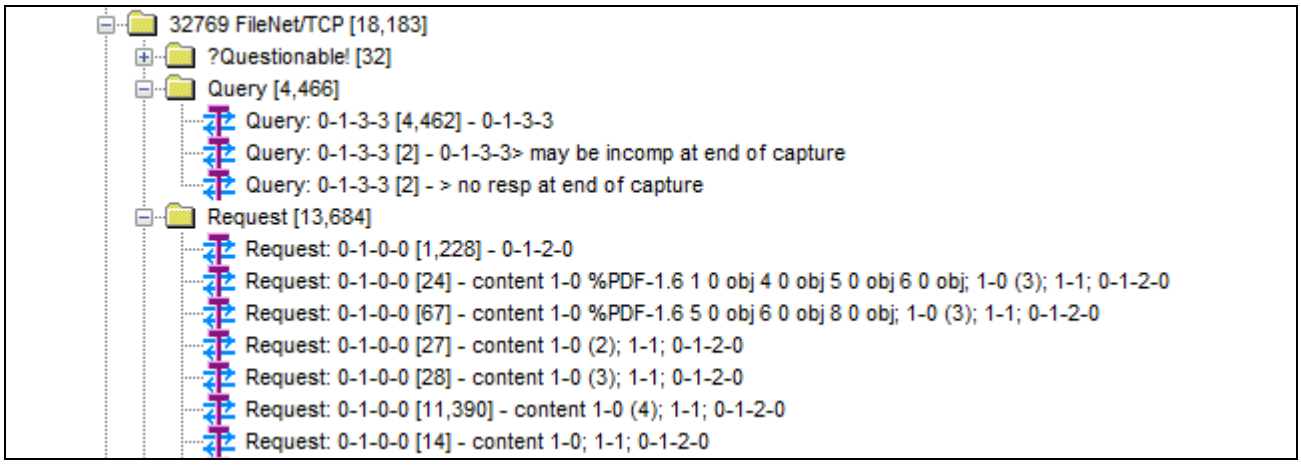
This function scans all the server and group transactions with a user name and forms user transactions that characterise each burst of such lower-level transactions. A user transaction is assumed to be complete after the user's workstation has been inactive for the minimum think time specified on the Charting page of controls.

4.5 FileNet Traffic Decoding

IBM's FileNet application is a content manager that specialises in serving images from a database. The images might, for example, portray customer accounts in a PDF format. NetData's FileNet decoder has been upgraded to distinguish different types of server round-trips.

FileNet request and response messages comprise one or more data blocks. Each block has a 4-byte header that starts with a 2-byte length indicator and has two bytes defining the block type. If the first byte is one, the block contains file content and its second byte signals (with 1 instead of 0) the last block in a message.

Non-file-content blocks begin with two double-byte integers that further define the block type. The header bytes and the block-type integers, separated by hyphens, are included in NetData's transaction request and response signatures, as in this extract from a transaction-class tree:



NetData recognises the first content block of a PDF file and displays the contents of the PDF objects found in the first thousand bytes of the file. Each object begins with a tag such as '5 0 obj' and these tags are also included in message signatures.

4.6 Tivoli Storage Productivity Centre Device Server

NetData decodes transactions with a Tivoli Storage Productivity Centre Device Server.

4.7 FileMaker Pro Host Contact

NetData detects traffic from FileMaker Pro attempting to contact a host server by broadcasting packets to UDP port 5003.

4.8 QVOD Player Search for SMV File Servers

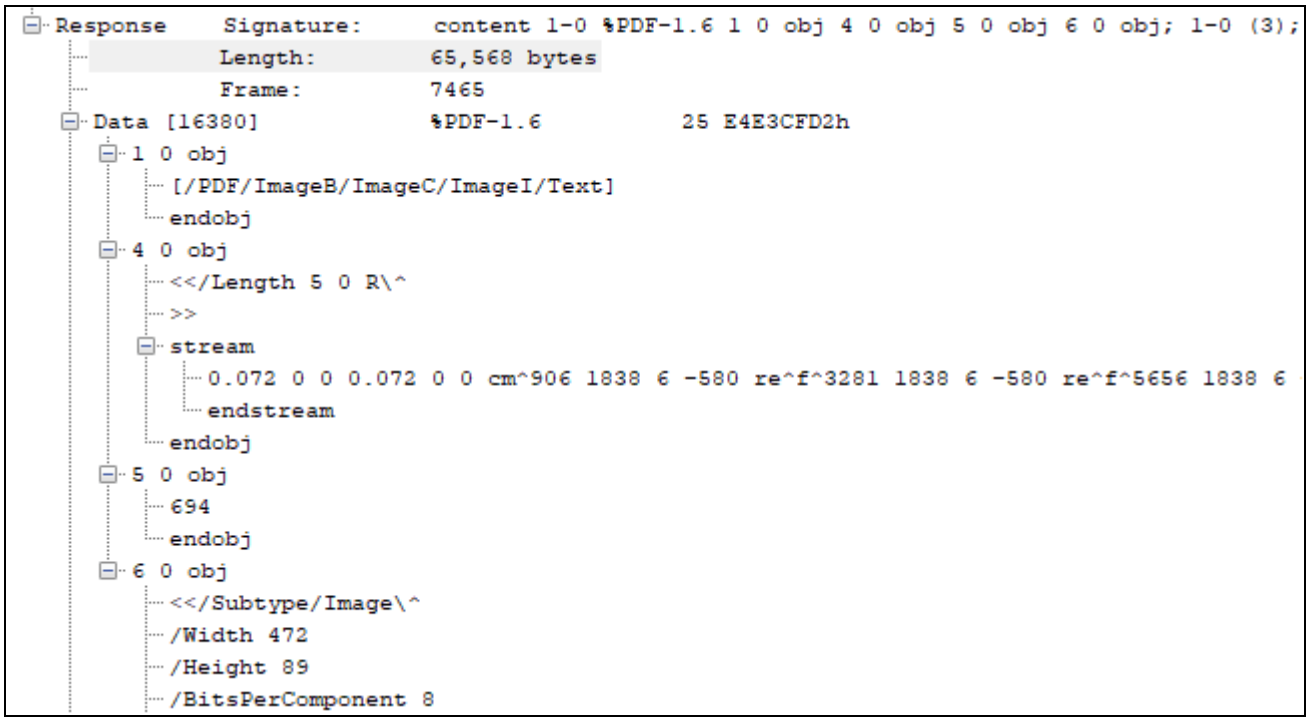
NetData detects traffic from QVOD players searching for hosts that serve SigmaTel Motion Video (SMV) files, by broadcasting to UDP port 20110.

4.9 Google Logging Service (glog)

NetData decodes transactions of logging services created with Google's C++ logging library (glog). It normally uses TCP port 4545. Messages may contain only a binary header, a binary header preceding an XML document, or an XML document on its own, as in the request and response messages of these transactions:

```
Request      Signature:     GLOG Req
             Length:        75 bytes
             Frame:         195559
  XML                       <Trans>
    <Req>
      <GLOG>
        <JobId>                      35786 </JobId>
        <Typ>                ALL           </Typ>
        </GLOG>
      </Req>
    </Trans>
Response     Signature:     GLOG Ack
             Length:        72 bytes
             Frame:         202343
  XML                       <Trans>
    <Ack>
      <GLOG>
        <JobId>                      35786 </JobId>
        <Tot>                            0 </Tot>
        </GLOG>
      </Ack>
    </Trans>
```

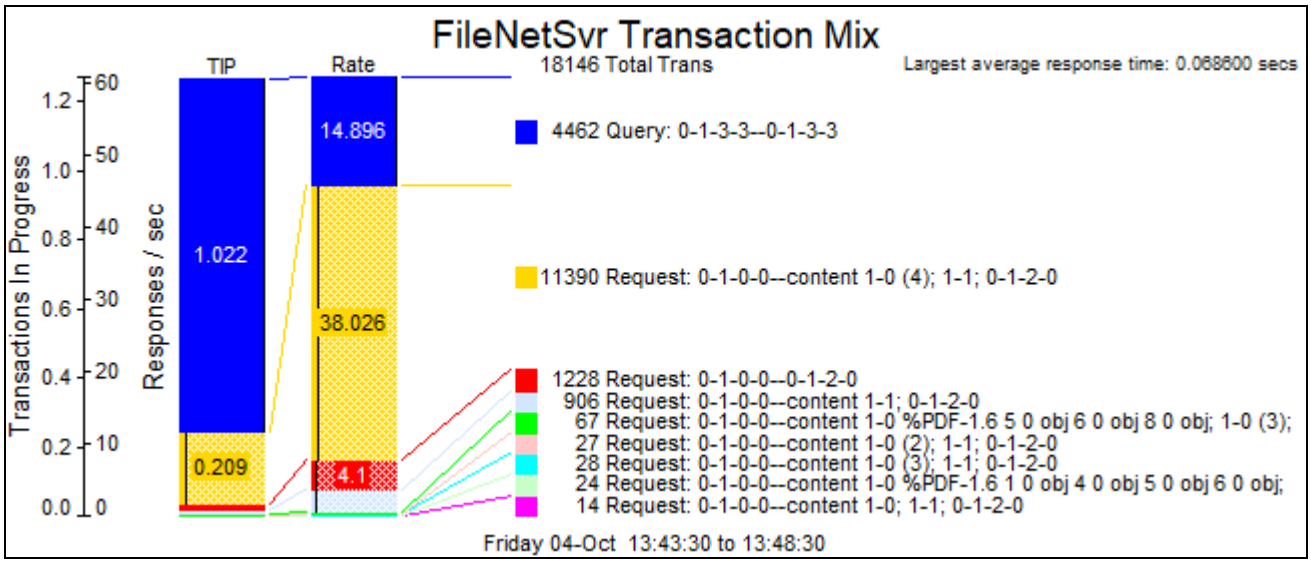
```
Request      Signature:     WASUP Req
             Length:        43 bytes
             Frame:         260910
  XML                       <Trans>
    <Req>
      <WASUP>                                 </WASUP>
      </Req>
    </Trans>
Response     Signature:     WASUP Ack
             Length:        690 bytes
             Frame:         271499
  XML                       <Trans>
    <Ack>
      <WASUP>
        <Ip>                 10.'  .2.15        </Ip>
        <Mac>                000B2806B027       </Mac>
        <MoSer>              AC2014090057       </MoSer>
        <EqSer>              9A162227           </EqSer>
        <EqId>                              12 </EqId>
        <Loc>
          <SSID>               dim                 </SSID>
          <CurMAC>             00351AB144EE        </CurMAC>
          <AP n="0">
            <MAC>                00351AB144EE        </MAC>
            <Sig>                                 -64 </Sig>
            <Noi>                                 -96 </Noi>
            <Rate>                                  0 </Rate>
            </AP>
          </Loc>
        <Vrsns>
        </WASUP>
      </Ack>
    </Trans>
```



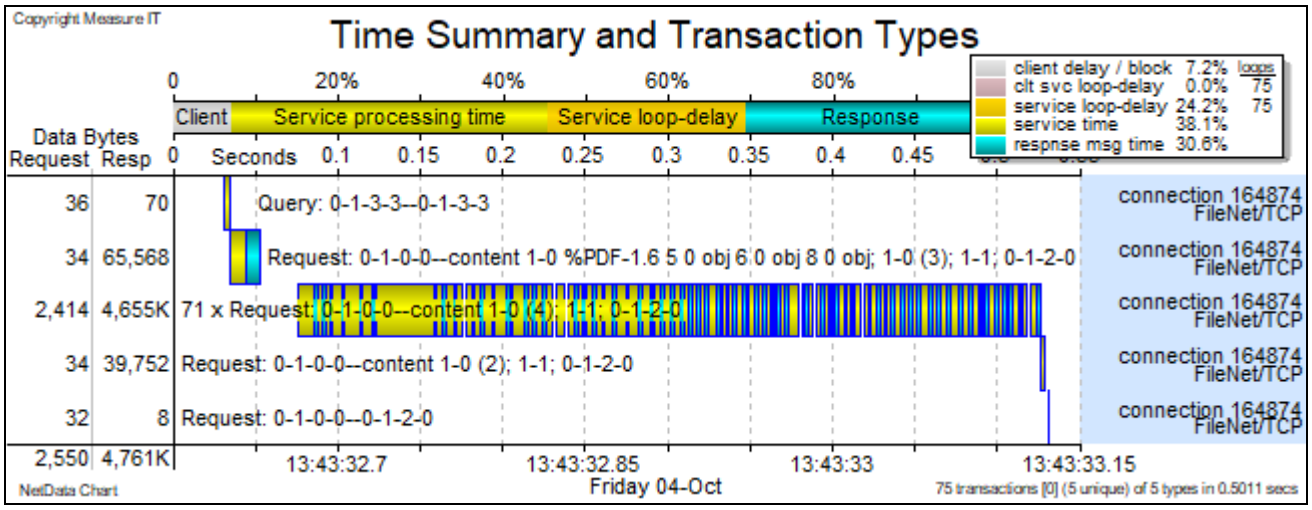
The first round-trip on a connection has the signature 0-1-3-3 for both request and response messages. These transactions are generally the slowest, probably because they serve as queries for requested content. When decoding all non-content messages NetData identifies all strings of text that contain words, numbers or dates (in mm/dd/yyyy format). Significant text strings are recorded with their transaction's key data, and transaction desriptions show the interleaved strings of text and binary data:

```
Key data:                820884/ 07/04/2019/ 10/04/2019; 820884/ 2784569048/ 2784869459
Category:                Request
Request     Signature:   0-1-3-3
            Length:      194 bytes
            Frame:       329641
  data [186]: [118]      0000 0000 0800 3402  0001 000C 4E7F 5C7C  0000 0010 00[*6] 000
         [6]             820884
  [2]                    0032h
         [2]             03
  [2]                    0032h
         [1]             2
  [2]                    0032h
         [3]             736
  [3]                    0000 32h
         [10]            07/04/2019
  [8]                    0046 0000 46A1 0032h
         [10]            10/04/2019
  [6]                    0046 0000 46FDh
Response    Signature:   0-1-3-3
            Length:      3,638 bytes
            Frame:       385899
  data [3630]: [15]      0002 0000 0010 0000  0000 00DC 0016 00h
         [1]             G
  [5]                    A5FD B4C5 00h
         [2]             E
  [77]                   0100 0300 3800 0046  A200 0500 3800 0050  9F00 0700 4300 3100
         [6]             820884
  [4]                    0022 0032h
         [2]             ai
  [4]                    0023 0032h
         [5]             07564
```

The following transaction-mix chart illustrates the dominant role of query transactions (0-1-3-3; coloured blue) in consuming server resources:



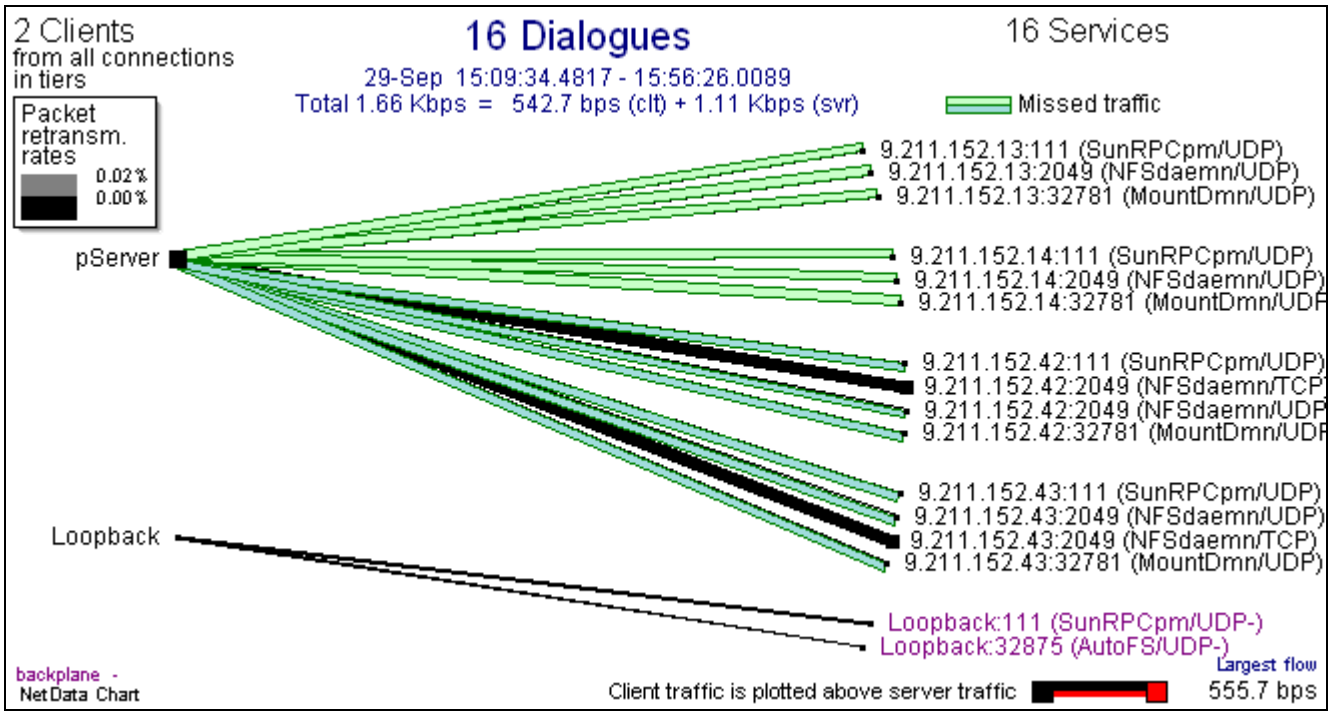
The most common transaction type (coloured cream in the mix chart) returns five content blocks totalling 65 Kbytes of content and ending with a small supervisory block. A large number of these transactions is sometimes needed to fetch the complete content of a large PDF file as illustrated by the following chart that summarise all the activity of a server connection.



4.10 Sun RPC/UDP Call-Reply Matching

Remote Procedure Calls (RPCs) using the Sun standard format and UDP, to a port mapper (at port 111), an NFS (Network File System) daemon (at port 2049), and other services such as the Mount daemon and the Auto FS daemon, are sometimes sent to one IP address but receive replies from a second address. Swapping addresses in this way doesn't break any rule but makes it difficult to match replies with their calls because the respective packets are associated with different connections. NetData searches more widely to match RPC replies with calls. Not only are transactions characterised in these circumstances, but the wider search also allows NetData to properly tag reply connections and correctly determine their client-server orientation, thus removing spurious dialogues from dialogue charts.

This chart illustrates UDP dialogues that convey packets in only one direction. Calls were sent *to* 9.211.152.42 and 9.211.152.43; replies were sent *from* 9.211.152.13 and 9.211.152.14:



NetData assigns distinguishing tags to traffic of the NFS Mount and AutoFS daemons.

4.11 Working With Huge File-Server Transaction Trees

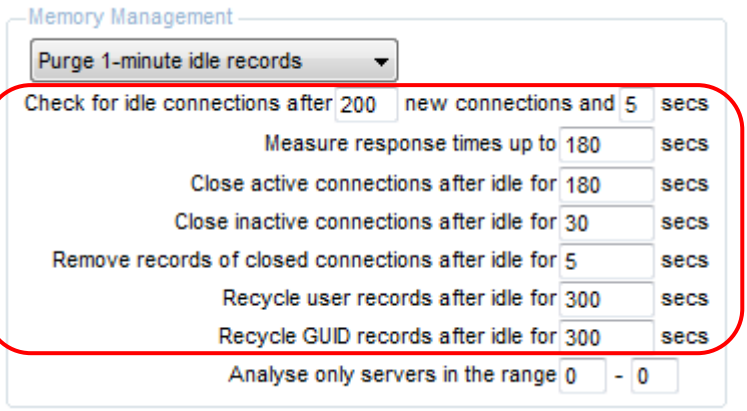
Some traffic captures contain large numbers of both concurrent connections and transaction types. For example, a central file server, storing the workstation files of thousands of roaming users, will carry huge loads during a morning peak hour when most users log on with a roaming profile, and users with large numbers of files and cookies may need tens of minutes just to complete the logon process (see *Roaming user profile* in Wikipedia). Windows fetches metadata of all the remote files using SMB GetInfo commands, and copies some of a user's files to the local disk.

When decoding file-server traffic NetData will record fewer than 300 types of SMB transactions but, because NetData includes file names in the signatures of file-IO *group* transactions, it might record hundreds of thousands of such types. The resulting transaction tree is useful because it identifies all the accessed files of every user, presented in a structure that reflects the directory structure of each user's virtual disk. Every folder on a disk is matched by a transaction group that appears in the tree, and, like folders, any group may contain sub-groups and transaction types. One file-server analysis by NetData recorded 2,500 clients and 250,000 transaction types over a period of 12 minutes during a morning peak hour.

NetData is unable to fully expand a tree with a hundred thousand transaction types but it can display any portion of the tree. However, it prepares the full tree automatically on completion of an analysis. The first display of a tree is delayed only by the need to build two database indexes.

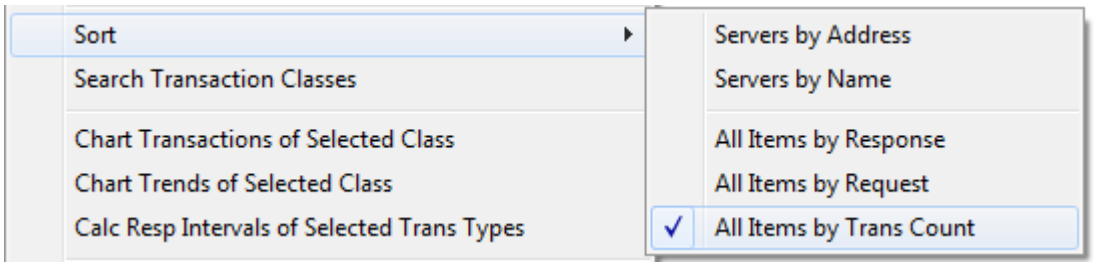
Although NetData can't fully expand a huge tree it can expand the relevant parts of a tree to display all the transactions whose signatures contain a particular text. Tree searching is controlled through a separate window that is displayed by a Search command in the Tree menu. One of the search controls will select all the matching transaction types ready for the Chart button to load and plot the response times of all the associated transactions.

Analysis of file-server traffic with thousands of concurrent connections requires NetData to manage its use of memory aggressively, as with the following controls set explicitly:



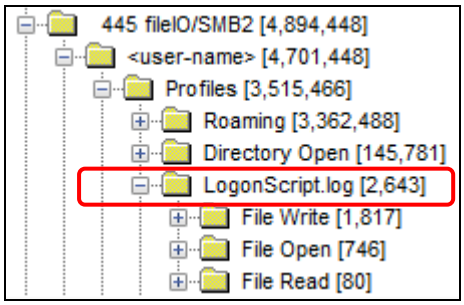
To make the best use of memory when analysing a sequence of 20 Gbytes or more of this traffic, it will also help if NetData is allowed to rebuild its data structures occasionally, by setting a terminate-and-restore memory threshold on the Input page of controls:

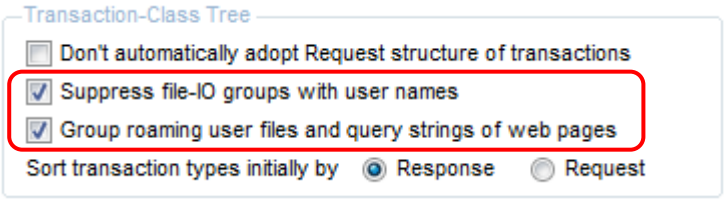
In the transaction tree NetData displays the number of transactions recorded for each transaction type, transaction group, and server. An option in the tree menu sorts tree items in descending order of their transaction counts, to clearly identify the busiest servers, groups and types.



Transaction counts are particularly useful in a tree that displays the file-IO transactions of a file server with roaming users. Because the path names of all files on the server begin with the user's initial letter and full name – as in 'F\First_Surname\' – a normal tree sorted by transaction count identifies the busiest users, and identifies which files of each user are read, written, or opened only to fetch attributes.

If the load on a file server seems too high it is possible to identify the applications and other categories of files that generate the most transactions during logins and at other times. A new checkbox in the transaction-tree group on the Tuning page of controls will suppress user names when aggregating transactions into groups. Another box will create transaction groups for individual file names whose path names start with a user name, to count the number of accesses to files (such as *LogonScript.log*, below) across all users, and to list file users.



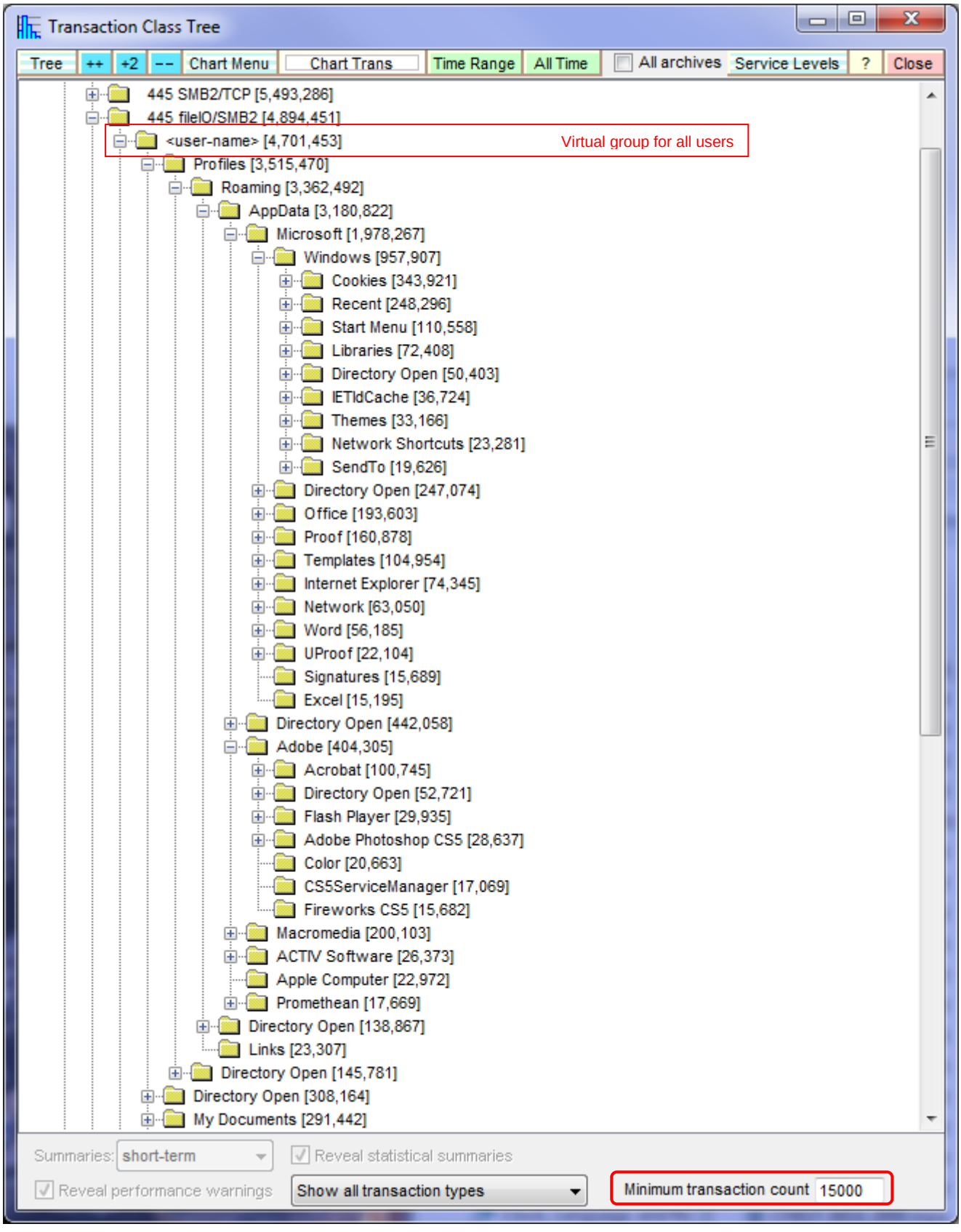


These boxes should be checked before traffic is analysed because they will take effect when groups are identified at the end of analysis. However, after analysis, groups can be recreated at any time with the 'Reconstruct Transaction Tree' command in the Maintenance sub-menu of the Project menu.

The following view of a transaction tree displays the busiest transaction groups of a file server. To display all the busiest groups on a single page, groups with fewer than 15,000 transactions were hidden by setting a minimum number of transactions in a control in the window's footer.

When user names are suppressed, all the transactions that access a file whose path name begins with a user's name are collected in the group labelled '<user-name>'. The sample tree view shows – with 4,701,453 transactions for the user-name group – that most of the file activity concerned particular users; the other transactions were Directory Open group transactions that fetched the attributes of higher-level directories.

The transaction counts for file-IO types of transactions refer not to the number of file-IO group transactions – i.e. the number of times files or directories are opened – but to all the SMB-command round-trips associated with the file accesses. The sample view shows that file-IO activity accounted for only 4,894,451 transactions out of a total of 5,493,286 SMB2 commands; the remainder were largely session-establishment and IO-control commands.



In this capture most of the traffic generated by user logins concerned the AppData folders of roaming users, and most of that traffic concerned the attributes of cookies in the Microsoft Windows folder. The second busiest AppData folder was the Adobe folder, and its busiest program was Acrobat.

4.12 Novell NCP File Service

The decoder for the NCP File service parses the messages of a new family of functions that specify lengths of path names with two bytes rather than one. File time and date stamps are decoded. File IDs, path names, file sizes and search patterns are displayed in the Data column of the transaction table to give a clearer view of the behaviour of NetWare clients.

4.13 IBM GPFS (General Parallel File System)

NetData recognises GPFS traffic and measures the response times of round-trips to or from the GPFS daemon.

4.14 EMC Documentum Content Server

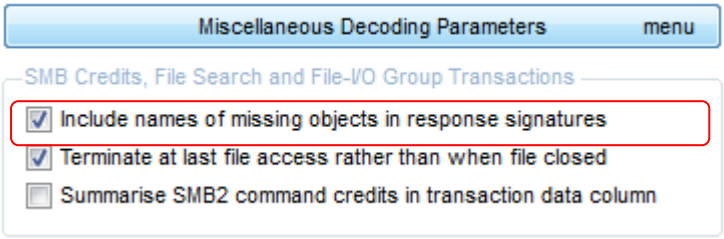
NetData decodes transactions of EMC Documentum's content server and connection broker. Messages are encoded with the Basic Encoding Rules of ASN.1 (Abstract Syntax Notation). Documentum has its own database manager and NetData extracts SQL statements from command messages, generalises them for transaction signatures, and displays them in their structured form (as it does for SQL statements in other protocols including Oracle SQLnet, Microsoft TDS, Sybase, Interbase, Teradata, IPFX, SAS, DST Systems, MySQL, Dharma, and IBM DB2).

5 File Transfer

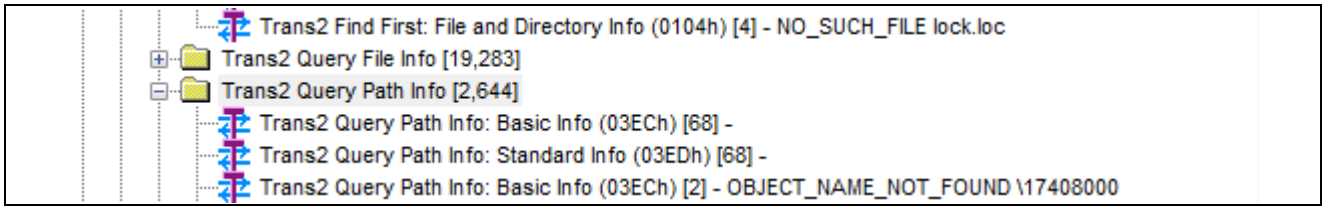
5.1 Missing Files Reported by SMB Commands

Many SMB (Server Message Block) commands return error codes such as BUFFER_TOO_SMALL or MORE_PROCESSING_REQUIRED that don't signify any fault but indicate a normal state transition within the execution of a multi-command function. Such error codes are reported in transaction descriptions but otherwise ignored, whereas all other errors are recorded also as network events.

A common but generally significant error concerns a failure to find a requested file or object path. A new option in the miscellaneous decoding controls, enabled by default, extends an SMB transaction's response signature to include the name of the missing file or object:



Failed requests for individual files can now be found in the transaction-class tree, together with the number of failed requests (in square brackets). It is now an easy matter to plot the occurrence of particular failed requests.



5.2 Aspera FASP

NetData detects and decodes parts of messages that use Aspera FASP (Fast And Secure Protocol) to transfer files. Aspera has developed a file-transfer protocol on UDP that is generally faster and more efficient than FTP/TCP because it avoids a sliding-window scheme and separates the two functions of congestion control and data recovery. It adapts flow to the capability of the path by measuring round-trip times and reacting to changes in those times rather than indications of packet loss.

NetData doesn't characterise any transactions associated with Aspera file transfers but does track data-packet sequence numbers and on a packet-timing chart will mark any sequence gaps, retransmissions, and requests for retransmission (displayed as selective acks).

5.3 DST Systems AWD (Automated Work Distributor)

NetData's decoder for AWD Views is supplemented with decoders for the AWD XML Service protocol and the AWD file transfer protocol. The latter is fashioned on FTP, using separate connections for control and data, and using a passive mode in which the server is requested to identify a dynamic port for data transfer.

5.4 AOL Instant Messaging and OSCAR File Transfer

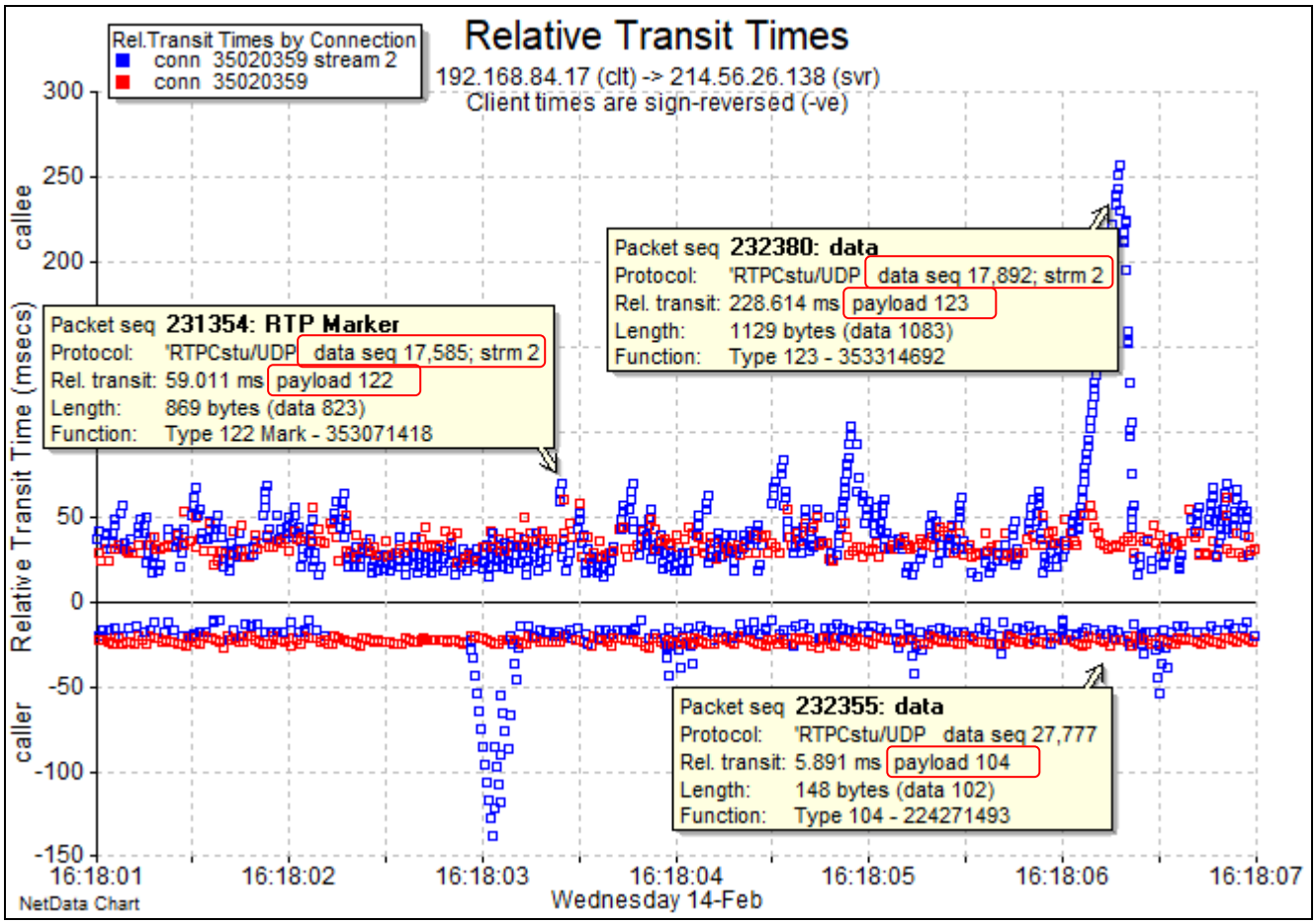
NetData's decoder for AOL Instant Messaging (AIM) Open System for Communication in Realtime (OSCAR) handles the OSCAR File Transfer (OFT2) protocol. It measures the response times for acknowledgements to client messages, and for all phases of file transfers.

6 VoIP and Media Transfer

6.1 Multiple RTP Streams in a Single Connection

In conventional VoIP sessions it is rare for a single UDP connection to convey more than one RTP stream in each direction, but in calls that combine video and audio streams, particularly when traversing network gateways that translate addresses, it is likely that all the RTP streams, RTCP packets and STUN transactions will share the one UDP connection.

NetData now handles multiple RTP streams in a single connection, tracking the independent sequences of timestamps and packet numbers to identify sequence gaps and calculate relative transit times. If transit-time markers are coloured according to connection ID, separate colours are assigned to the different RTP streams as well as to the different connections:



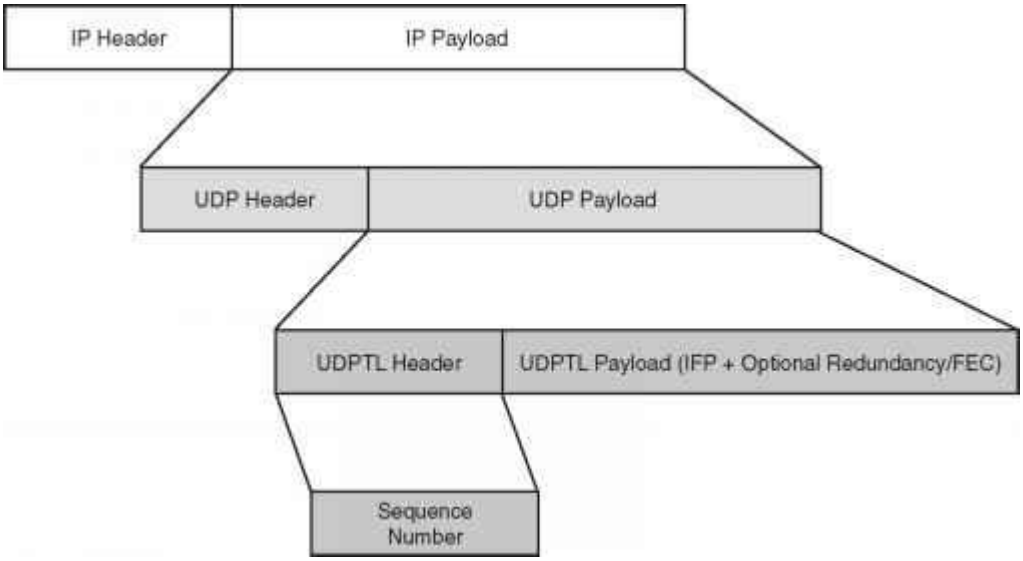
This chart plots relative transit times during a Skype call with both audio streams (red) and video streams (blue). Although the traffic was captured at the client there were occasional large delays in preparing video data for transmission. The traffic from the server also showed large delays in the video stream when there was very little delay in the audio stream.

RTP packet pop-ups indicate a packet's sequence number, its RTP stream number (if not the first of the connection), and its media payload type.

The payload type can't be used to identify an RTP stream because one stream can use different payload types. The Skype video stream in this example used payload types 122 and 123 for different packets.

6.2 T.38 Facsimile over IP (FoIP)

NetData now decodes facsimile-transfer packets that use the ITU-T T.38 protocol running on the UDP Transport Layer (UDPTL) on UDP/IP. This is the only use of UDPTL.



The UDPTL *primary* payload, and optional *secondary* payloads (for error recovery), contain packets of the Internet Facsimile Protocol (IFP).

```
Frame 4188
Content: Indicator V.21 Preamble (HDLC flags)
    T.38/UDPTL  sequence     88
    primary IFP pkt [1]      Indicator (06h) V.21 Preamble (HDLC flags)
    Error-recovery secondaries:0
```

An IFP T.30 *Indicator* packet that precedes an HDLC frame

```
Frame 4201
Content: T.4 Non-ECM
    T.38/UDPTL  sequence     2478
    Primary IFP pkt [41]     Data (hdr D001E00023D9A8h) V.17 14400
        field type (6)       T.4 Non-ECM
        image data           D9A80014D9A80014D9A80014D9A80014D9A80014D9A80014D9A80014D9A80(
    Error-recovery secondaries:2
        (2477)secondary [41] Data (hdr D001E00023D9A8h) V.17 14400
            field type (6)   T.4 Non-ECM
            image data       D9A80014D9A80014D9A80014D9A80014D9A80014D9A80014D9A80014D9A80(
        (2476)secondary [41] Data (hdr D001E00023D9A8h) V.17 14400
            field type (6)   T.4 Non-ECM
            image data       D9A80014D9A80014D9A80014D9A80014D9A80014D9A80014D9A80014D9A80(
```

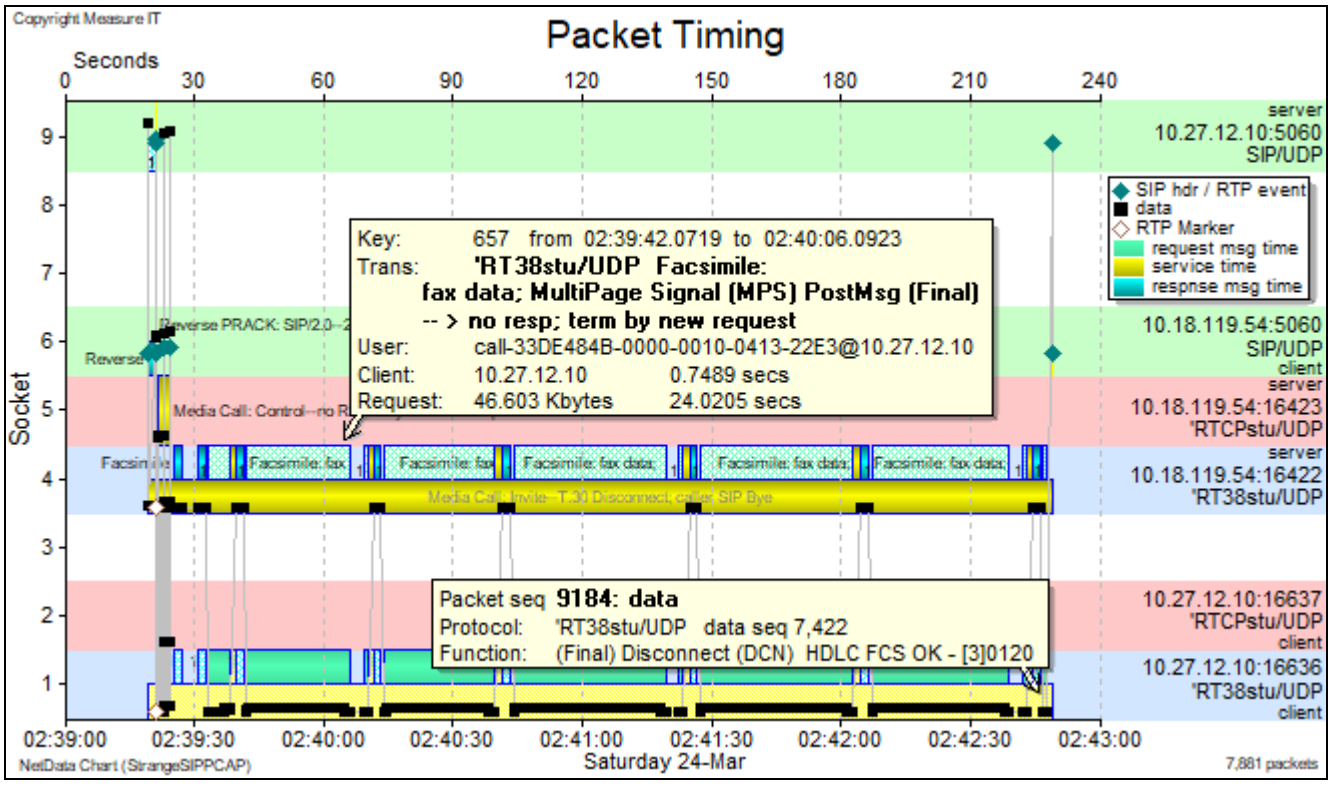
An IFP *Data* packet that conveys image data.

NetData re-assembles and decodes the complete HDLC frames that convey T.30 signals, and assembles a list of all the indicators, fax data blocks and signal frames in the message sequences between reversals of data flow. NetData normally terminates a message sequence when an HDLC frame contains a Final bit in its control byte. NetData's description of a packet includes an HDLC frame description if the packet completes a frame, or a message-sequence list if the packet completes a sequence, as below.

```
Frame 4194
Content: (Final) Message Confirmation (MCF) PostMsgResp HDLC FCS OK
    T.38/UDPTL  sequence     92
    Primary IFP pkt [3]      Data (C00120h) V.21 300
        field type (2)       HDLC FCS OK (no data field)
    Fax message sequence:
        Indicator            No signal
        Indicator            V.21 Preamble (HDLC flags)
        HDLC signal [3] FIN  Message Confirmation (MCF) PostMsgResp
    Error-recovery secondaries:3
        (91)secondary [6]    Data (C00180000031h) V.21 300
        (90)secondary [6]    Data (C001800000C8h) V.21 300
        (89)secondary [6]    Data (C001800000FFh) V.21 300
```

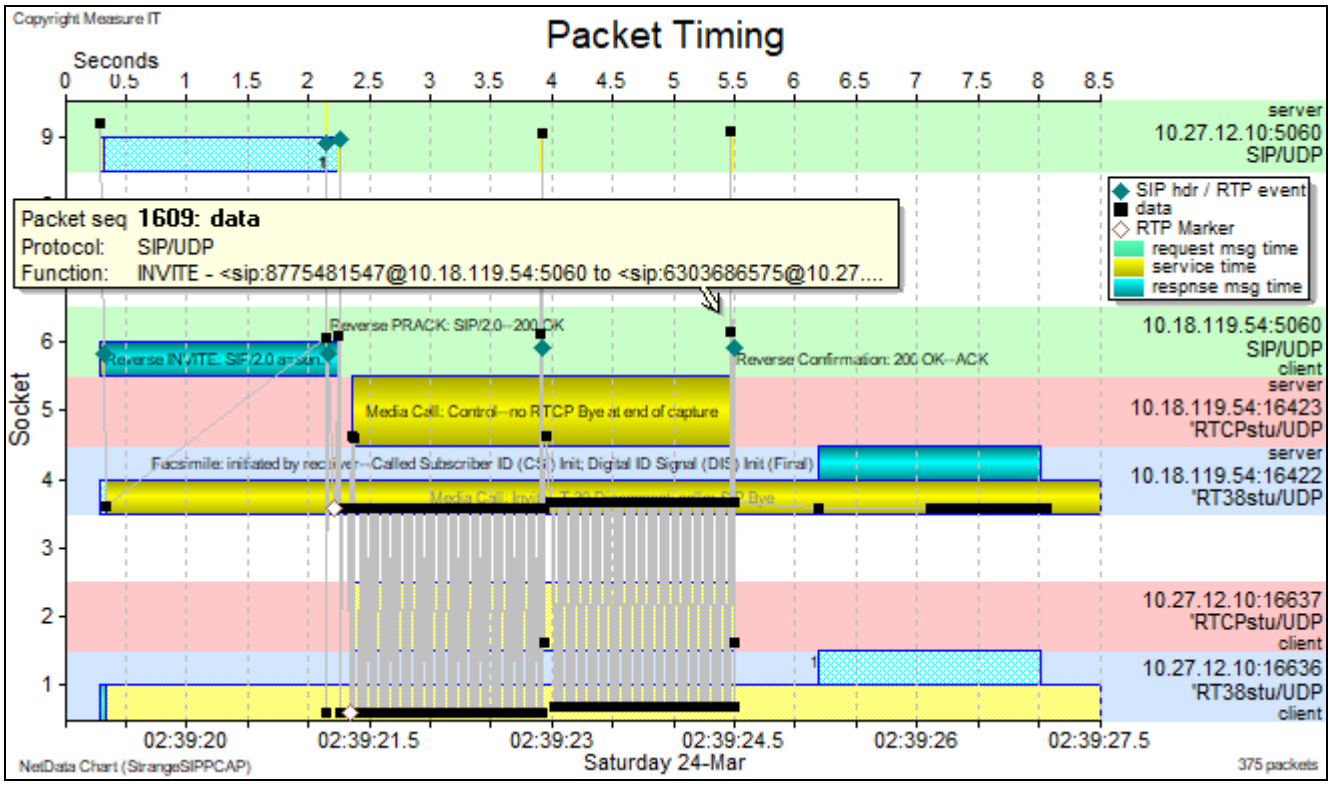
An IFP T.30 *Data* packet that indicates the end of an HDLC frame. The three bytes that constitute this signal frame (FFC831h) can be seen at the end of the contents of the three preceding IFP packets that have been included as secondaries in the same UDP packet.

T.38 facsimile packets may appear in a UDP connection after it has conveyed RTP packets, and the connection may also convey STUN (Session Traversal Utilities for NAT) messages, as in the following FoIP session:



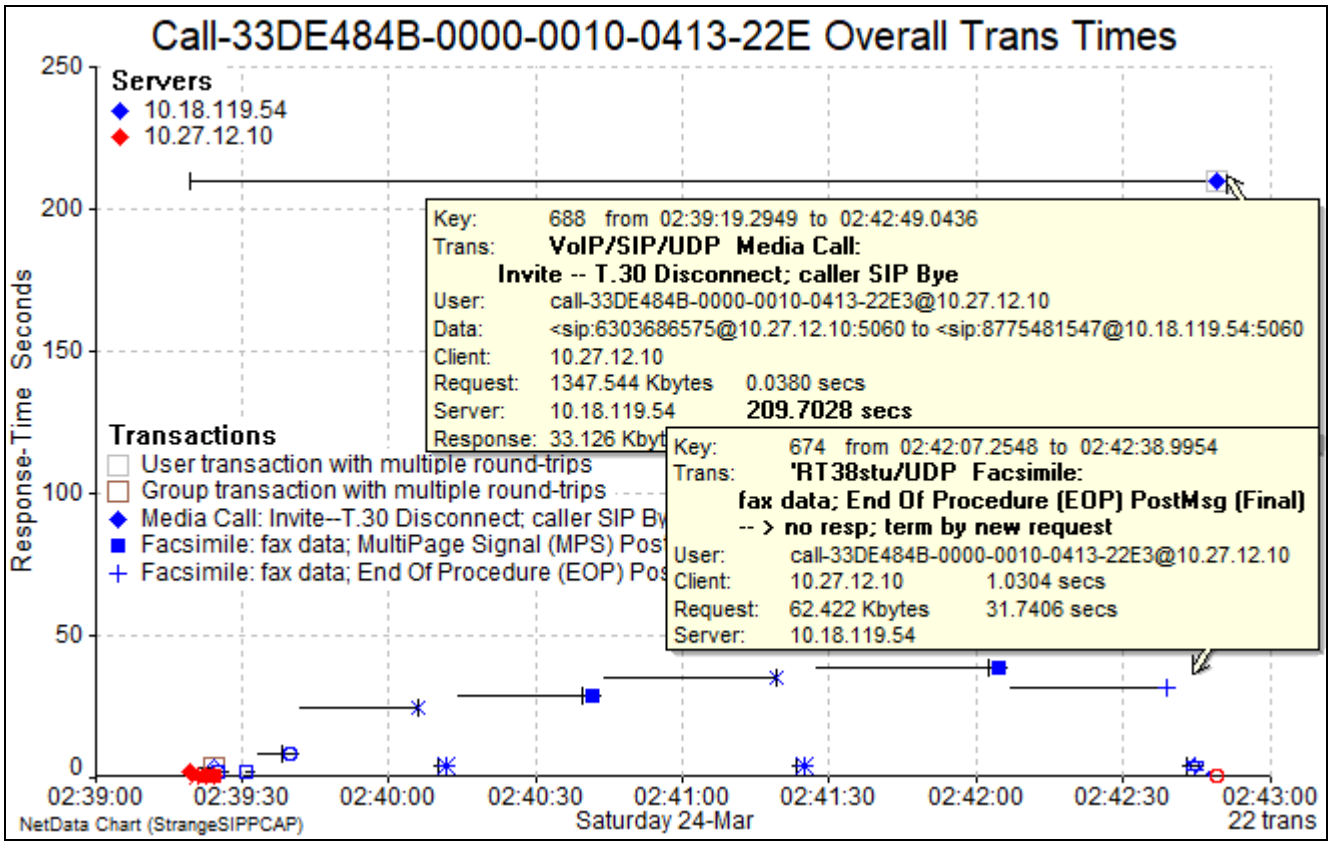
Pairs of message sequences in round-trips are recorded as facsimile transactions. The transactions in this FoIP session reveal three 3-second timeouts, and retransmissions of HDLC signals with Final bits set to solicit a response.

This session also had three SIP Invite transactions specifying different media: G.729 audio, G.711 audio and T.38 image. The two audio periods lasted for only 2.5 seconds each, and after 210 seconds the session ended with a T.30 Disconnect message and a SIP Bye message.

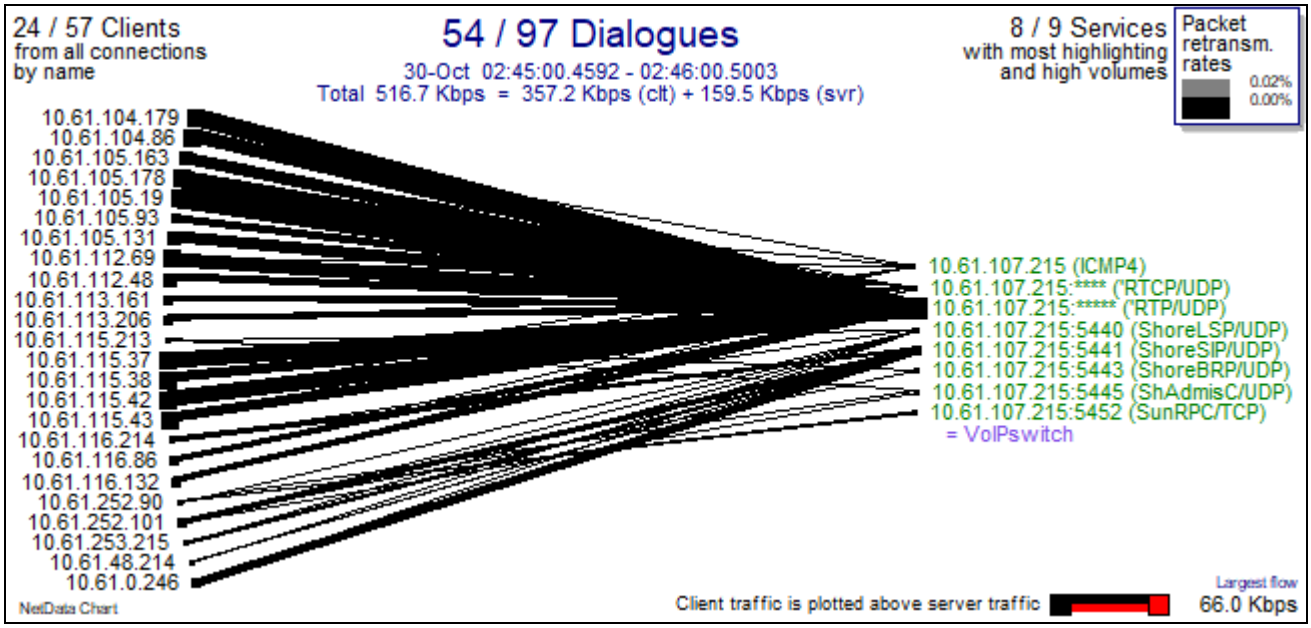


NetData tracks T.38 packet sequence numbers and highlights sequence gaps in the normal way, with red packet markers on the timing chart. As far as possible, UDPTL recovers the data in lost packets by referring to the secondary packets in UDP payloads, and NetData fully emulates this forward error-recovery system to assemble and display messages correctly.

The group transaction that characterises the whole media session indicates that it started with a SIP Invite and ended with a T.30 Disconnect and a SIP Bye.



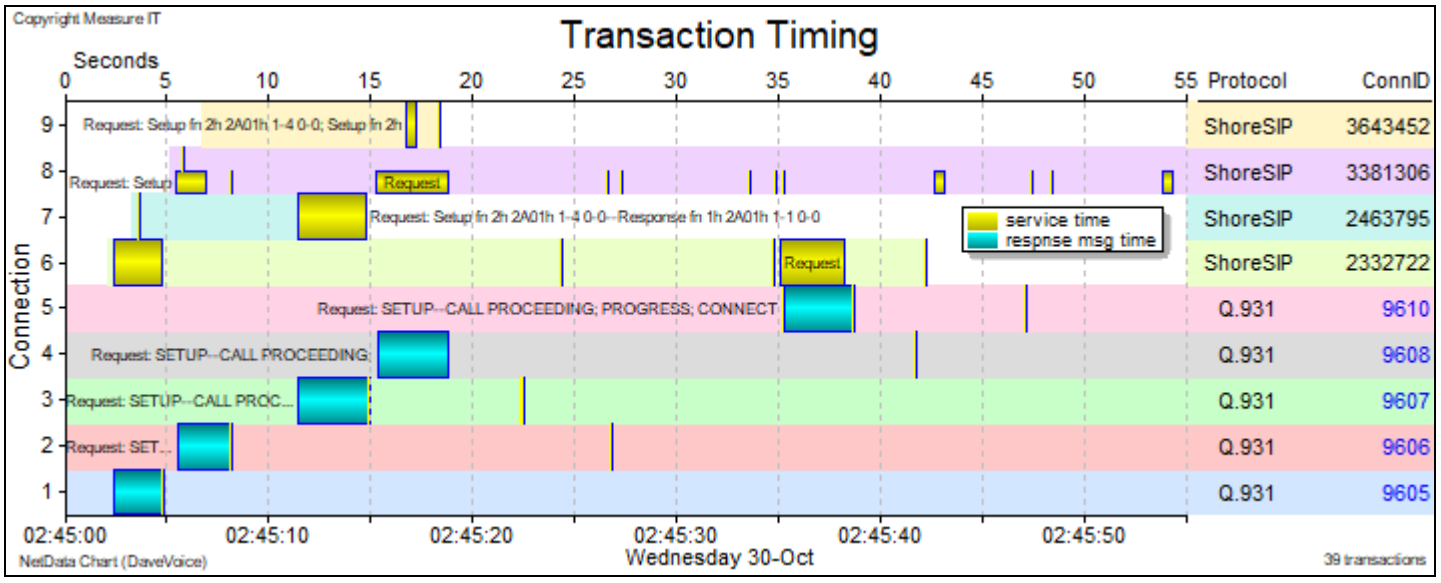
6.3 ShoreTel ShoreWare Call Management



A ShoreTel voice switch exchanges call management information through a group of UDP ports with numbers from 5440 to 5446. Port 5440 uses a Location Service Protocol (LSP) and responds to LSP pings to verify switch availability. Port 5442 provides a call Distributed Routing Service (DRS), and port 5446 a DRS keep-alive service. Port 5443 uses a Bandwidth Reservation Protocol (BRP) and port 5445 provides an admission control service associated with bandwidth reservation.

Port 5442 provides a call control service like SIP. Most of its messages contain strings of text within binary-coded data and NetData extracts significant texts for display in the KeyData column of the packet table. Some text strings can be recognised as SDP (Session Description Protocol) statements as found in the bodies of SIP messages, and others specify phone numbers that start with their international prefix (e.g. +1). NetData characterises round-trip Setup, Connect and Release transactions that correspond to Q.931 transactions.

	Time Of Day	ConnID	AppType	Data	Function	KeyData
■	02:45:18.841745	3381306	ShoreSIP	1196	Response fn 1h...	ID 20070; TGrp_32,p1; 10.35.115.156:10166; +15635435020
■	02:45:26.613933	3381306	ShoreSIP	1196	Response fn 1h...	ID 20029; 3177
■	02:45:27.244417	3381306	ShoreSIP	274	Connect fn Ah ...	ID 20021; TGrp_14,p23; TRUNK_INTERVAL=00000078;00000046;TGrp_14
■	02:45:33.498978	3381306	ShoreSIP	274	Connect fn Ah ...	ID 19891; TGrp_32,p11; TRUNK_INTERVAL=000001c2;00000077;TGrp_32
■	02:45:33.500092	3381306	ShoreSIP	1196	Response fn 1h...	ID 19891; 3476
■	02:45:34.78772	3381306	ShoreSIP	272	Connect fn Ah ...	ID 20064; TGrp_32,p4; TRUNK_INTERVAL=0000001e;00000000;TGrp_32,p
■	02:45:35.160875	3381306	ShoreSIP	1372	Setup fn 2h 2A...	ID 20076; 3521; +18002234139; TGrp_32
■	02:45:35.161842	3381306	ShoreSIP	1196	Response fn 1h...	ID 20076; TGrp_32
■	02:45:41.5883	3381306	ShoreSIP	200	Release fn 3h 2...	ID 20070; 0010495481f3
■	02:45:43.177698	3381306	ShoreSIP	1196	Response fn 1h...	ID 20080; TGrp_32,p6; 10.35.115.156:10172; +15632451717
■	02:45:47.341545	3381306	ShoreSIP	210	Connect fn Ah ...	ID 20080; TGrp_32,p6; FAREND_ANSWER
■	02:45:48.40815	3381306	ShoreSIP	1372	Setup fn 2h 2A...	ID 20082; 3313; +15635435020; TGrp_32
■	02:45:48.409095	3381306	ShoreSIP	1196	Response fn 1h...	ID 20082; TGrp_32
■	02:45:52.151275	3381306	ShoreSIP	1372	Setup fn 2h 2A...	ID 20086; 3177; +15635561658; TGrp_32
■	02:45:54.333962	3381306	ShoreSIP	1196	Response fn 1h...	ID 20086; TGrp_32,p5; 10.35.115.156:10176; +15635561658
■	02:45:59.59186	3381306	ShoreSIP	200	Release fn 3h 2...	ID 20080; 00104954813f
■	02:45:07.01967	3643452	ShoreSIP	200	Release fn 3h 2...	ID 13681; 0010493742f2
■	02:45:16.719997	3643452	ShoreSIP	1372	Setup fn 2h 2A...	ID 13682; 3802; +16087442440; TGrp_32
■	02:45:16.727463	3643452	ShoreSIP	1372	Setup fn 2h 2A...	ID 13682; 3802; +16087442440
■	02:45:16.740037	3643452	ShoreSIP	259	Setup fn 2h 2A...	ID 13682; 3802; =no\^a=rtpmap:0 PCMU/8000\^a=rtpmap:102 telephone-ev
■	02:45:18.280435	3643452	ShoreSIP	1196	Response fn 1h...	ID 13679; 3810; 001049376f90

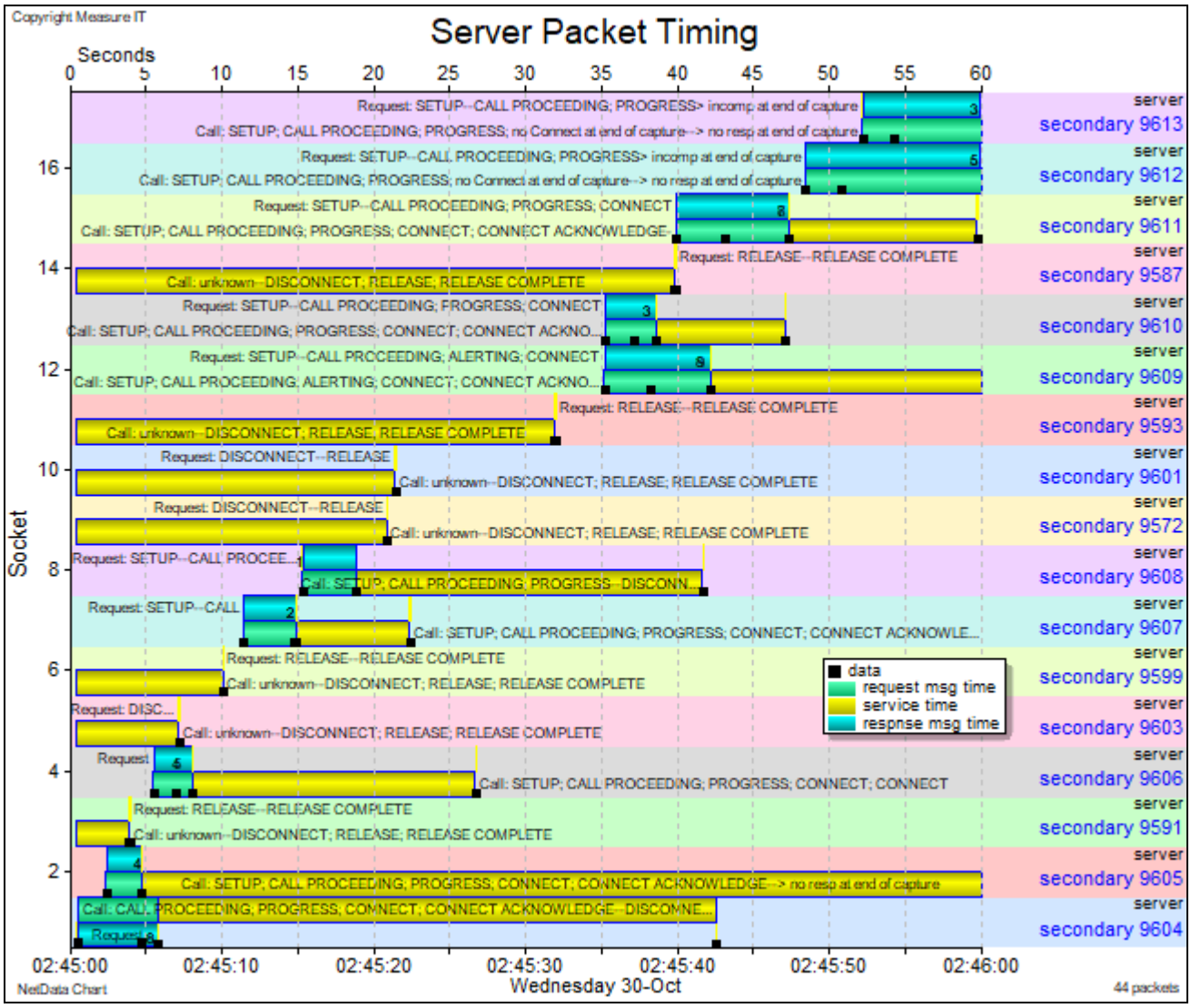


6.4 Q.931/LAPD Call Signalling

Voice switches that use the Primary Rate Interface (PRI), rather than VoIP SIP (Session Initiation Protocol), exchange signalling information with their peers in ISDN D channels using the ITU Q.931 protocol over Q.921, a protocol that is also known as LAPD (Link Access Protocol for Data channel). LAPD is an LLC2 (Logical Link Control Type 2) protocol similar to ISO HDLC (High-Level Data Link Control) and IBM's SDLC (Synchronous Data Link Control).

The sniffer in a ShoreTel PRI switch will record all its IP and LAPD traffic in a single capture file even though the packets of these protocols are recorded with different frame formats. IP packets are recorded with the standard Ethernet format (pcap link-layer header type of 1) whereas the LAPD

packets are recorded with the link-type of 177 which precedes the contents of LAPD packets with a Linux-LAPD header conveying 16 bytes of metadata.



NetData now decodes LAPD packets of this link type. The metadata is summarised in the Frame Description column of the packet table, and the LAPD header is summarised in the LLC column. NetData also decodes Q.931 messages, and characterises Q.931 transactions. In addition to recording individual round-trips such as call Setup and Disconnect requests, it also characterises as a user transaction the complete lifespan of each call, from Setup to Disconnect; the Setup phase is displayed with a green bar and the Disconnect phase with a blue bar.

The final response to a Setup request is a Connect message which must be acknowledged by the caller to complete a three-way handshake. Disconnection also requires a three-way handshake involving Disconnect, Release and Release Acknowledgement messages. Each three-way handshake is recorded by NetData as two consecutive round-trips sharing a common message (Connect for a Setup pair and Release for a Disconnect pair).

The header of every Q.931 message includes a call reference, a number that NetData displays in the Key Data column of the transaction table and uses to identify secondary connections as in the chart above. The Key Data column also displays caller and callee phone numbers.

	Type	Description	Svr Time	All Time	Key Data
•	Q.931/LAPD	Request: SETUP--CALL PROCEEDING; PROG...	0.0428	2.3739	9605; caller 4324899405; callee 1800300
◆	Q.931/LAPD	Call: SETUP; CALL PROCEEDING; PROGRES...	55.7493	58.1255	9605; caller 4324899405; callee 1800300
•	Q.931/LAPD	Reverse Request: DISCONNECT--RELEASE	0.0017	0.0017	9591
•	Q.931/LAPD	Request: RELEASE--RELEASE COMPLETE	0.0992	0.0992	9591
•	Q.931/LAPD	Reverse Request: CONNECT--CONNECT ACK...	0.0023	0.0023	9605
•	Q.931/LAPD	Request: SETUP--CALL PROCEEDING; PROG...	0.0665	2.6015	9606; caller 4324892400; callee 1866722
✳	Q.931/LAPD	Call: SETUP; CALL PROCEEDING; PROGRES...	18.6096	21.2552	9606; caller 4324892400; callee 1866722
•	Q.931/LAPD	Reverse Request: CONNECT--CONNECT ACK...	0.0009	0.0009	9604
•	Q.931/LAPD	Request: DISCONNECT--RELEASE	0.0777	0.0777	9603
•	Q.931/LAPD	Reverse Request: RELEASE--RELEASE COMP...	0.0015	0.0015	9603
•	Q.931/LAPD	Reverse Request: CONNECT--CONNECT ACK...	0.0009	0.0009	9606
•	Q.931/LAPD	Reverse Request: DISCONNECT--RELEASE	0.0014	0.0014	9599
•	Q.931/LAPD	Request: RELEASE--RELEASE COMPLETE	0.0591	0.0591	9599
•	Q.931/LAPD	Request: SETUP--CALL PROCEEDING; PROG...	0.0608	3.4587	9607; caller 4324899920; callee 4324920
✳	Q.931/LAPD	Call: SETUP; CALL PROCEEDING; PROGRES...	7.4758	10.9772	9607; caller 4324899920; callee 4324920
•	Q.931/LAPD	Reverse Request: CONNECT--CONNECT ACK...	0.0006	0.0006	9607
•	Q.931/LAPD	Request: SETUP--CALL PROCEEDING; PROG...	0.0822	3.5587	9608; caller 4324892210; callee 4324924
+	Q.931/LAPD	Call: SETUP; CALL PROCEEDING; PROGRES...	22.8293	26.4263	9608; caller 4324892210; callee 4324924
•	Q.931/LAPD	Request: DISCONNECT--RELEASE	0.0289	0.0289	9572
•	Q.931/LAPD	Reverse Request: RELEASE--RELEASE COMP...	0.0008	0.0008	9572
•	Q.931/LAPD	Request: DISCONNECT--RELEASE	0.0893	0.0893	9601

6.5 Data Streaming for Teams and Skype for Business

NetData detects streams of data from TCP port 25461 that are associated with Microsoft functions of Skype for Business and Teams. It tags this traffic as ‘Skype4B’.

6.6 Unknown Traffic with UDP Port 3478 (STUN and Teams)

NetData detects unknown traffic with UDP port 3478 which is normally used for STUN and Teams traffic. The first byte in most packets has the code 15h and many have only seven bytes such as

151B 0202 01FF FFh

Significant exceptions are packets of 37 bytes beginning with a code of 3 and containing a UUID, such as

9aac017f-76fc-487f-b821-bb7854a9f3c2

6.7 Microsoft Teams Codec Packet Formats

NetData decodes RTP packets in a new and apparently undocumented format found in UDP connections carrying Microsoft Teams audio and video data streams. These packets are distinguished by their first two bytes which have codes FF10h, and a length indicator in the next two bytes. In the Function column of the packet table NetData labels these packets as ‘Lync’ and they often appear with payloads of types 104 (audio ‘SILK’), 108 (video), 118 (audio), 122 (video) and 123 (video).

As it does with standard RTP packets, NetData extracts sequence numbers, marker flags, timestamps and synch-source information. All packets have been seen with a header extension in the standard RTP format (RFC 5385). The payload codec data is assumed to be encrypted.

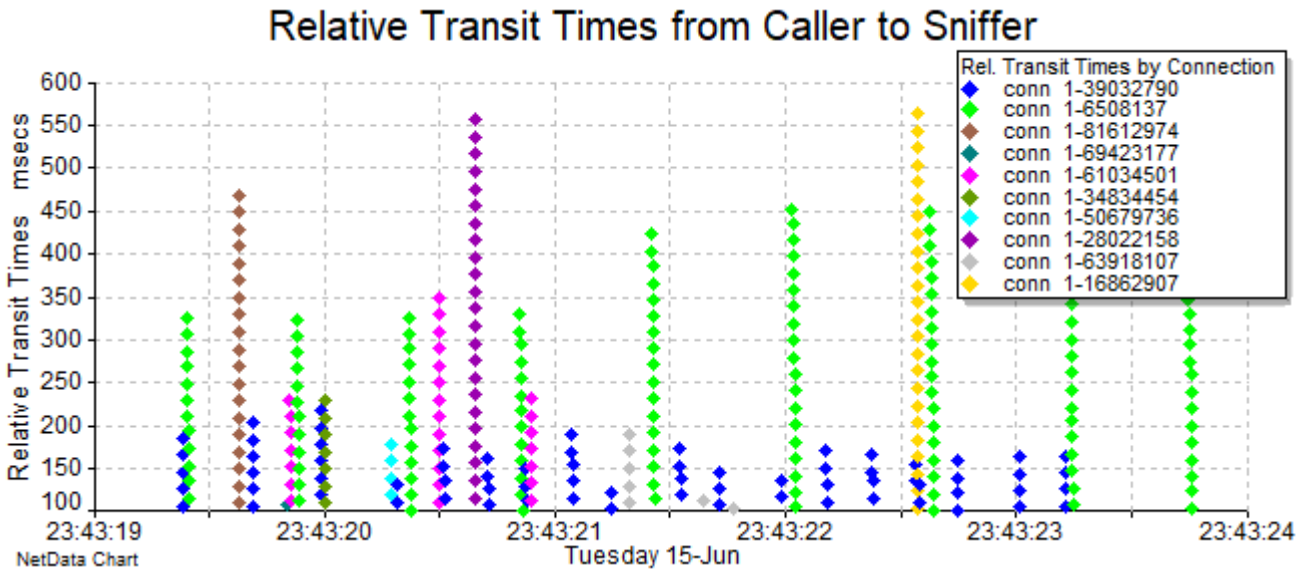
Similar types of Lync packets convey RTCP information and a new type of message that includes a table of parameters in a TLV (Type-Length-Value) format, as in:

```
Frame 2411069
Content: Lync 1h Params
    Lync header              FF10h
    length                   112
    stream ID                07F7 1F74  628A 6176h
    type                     0001h
                             2112 A442 FAA2 C768  E94A 3B1B F976 F869h
    Parameters:              [84]
        Type  Len   Value
        ------------------------------------------------------------------------------------
        0006h   9 562F 564B 3A32 5270  7700 0000h                        V/VK:2Rpwzzz
        8070h   4 0000 0007h                                             7
        0024h   4 6667 C017h                                            fg`.
        802Ah   8 0000 7FF3 9B72 35F0h
        8037h   4 0000 0002h                                            2
        0008h  20 2F80 C572 FA1A 9B12  BC82 BA34 FAC9 D4F8  96B7 86EFh  /``r`.`.```4````````
        8028h   4 A7A2 14CCh                                            2812417228
```

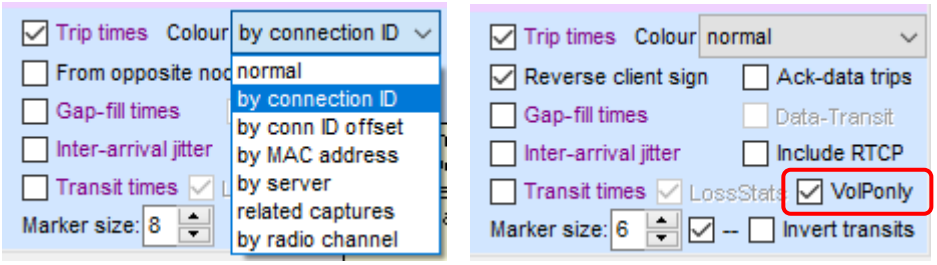
6.8 Displaying RTP (VoIP) Transit Times

NetData has additional options for displaying RTP (VoIP) relative transit times that in some cases provide invaluable insight to the causes of severe jitter affecting call quality.

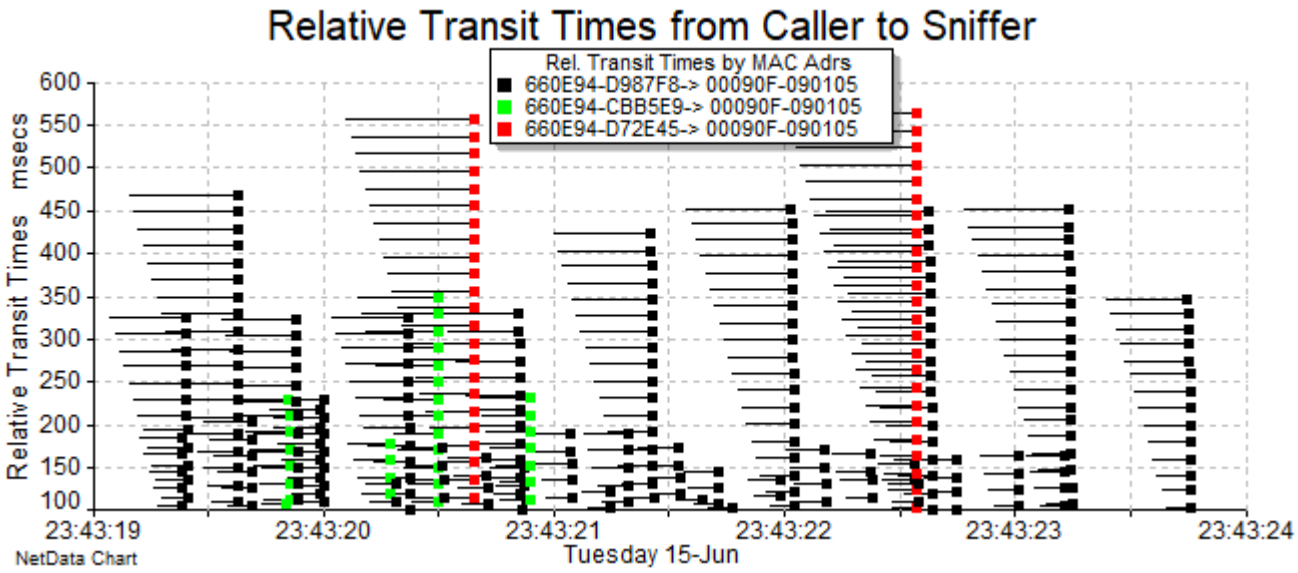
All the transit-time markers in the charts below relate to Microsoft Teams audio packets addressed to UDP port 3479, and the vertical towers of markers reveal evidence of packet blocking for periods of up to half a second.



For this chart the markers have been assigned colours according to their connection ID and it indicates that all the victimes of each blockage belong to the same connection, and some connections are victimised repeatedly. The menu of marker colouring options now has two options for distinguishing different connections; the second option adds different vertical offsets to the markers of each connection, to separate the bands of markers of quite small transit times that would otherwise overwrite each other.



Some blockages indicated by the marker towers appear to occur at regular intervals, while others are closely spaced in time. The question arises as to the extent that the blockages overlap, if at all, and this is answered by the next chart which adds horizontal lines to each marker to indicate its time in the blockage, from the time at which the packet should have been seen. To plot these lines for RTP packet transit times from the sender the VoIP-only box and the box labelled with a dash must be checked.



This chart reveals that large blockages often overlapped with small blockages and sometimes with other large blockages (up to a third of a second). However, because marker colours have been assigned according to packet MAC addresses, we see that the overlapping large blockages affected packet streams from different source addresses.

Findings in other charts of firewall transit times (see *Matching and Plotting Packets in Large Related Captures*) have attributed these types of blockages to faulty firewall software, and the charts above suggest that there may be several such firewalls in the network between the VoIP users and the sniffer.

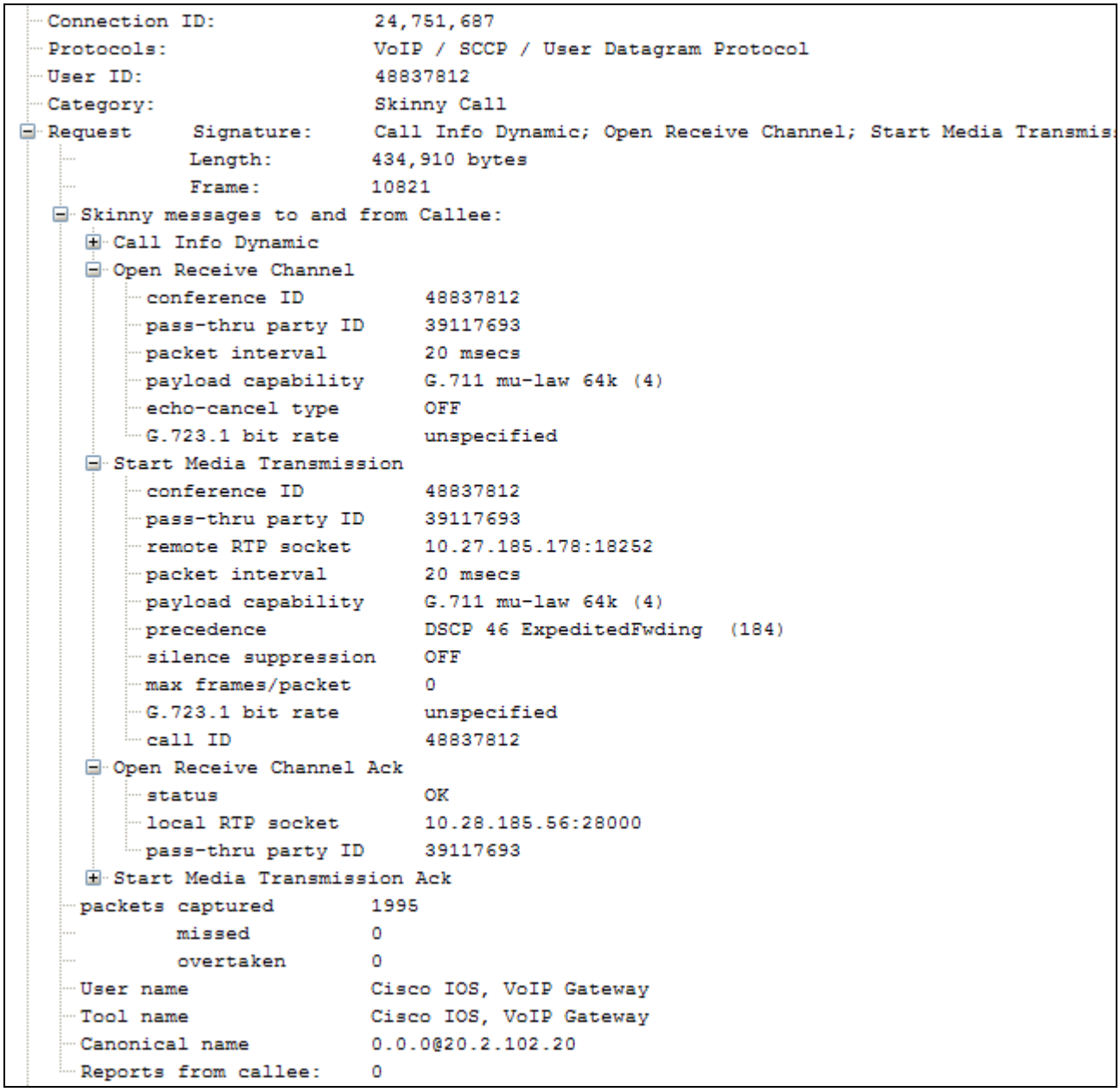
6.9 Miralix Phone Monitor

NetData decodes the traffic of the Miralix phone monitor that uses TCP port 3001.

6.10 VoIP/SCCP (Skinny) Decoder

The decoder for the Skinny Client Control Protocol (SCCP) can now decode and display the contents of many types of messages sent between a phone and Cisco CallManager. Messages that form responses are matched with their requests to characterise Skinny transactions. Complete calls

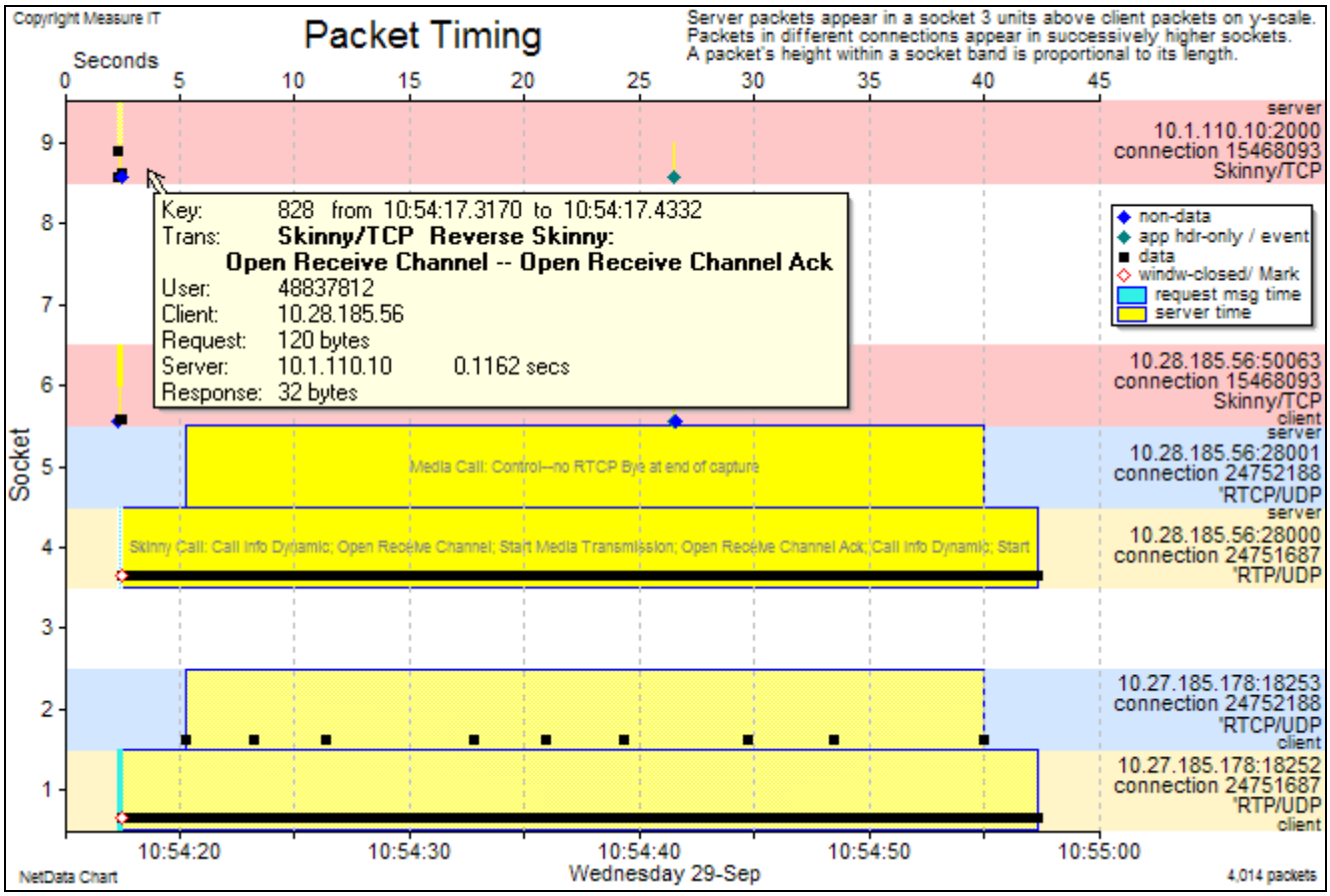
are also characterised, and they provide descriptions of call-initiation messages such as Open Receive Channel, Start Media Transmission and Call Info as in the following Skinny call description:



Complete calls are classed as *user* transactions and they are related to their associated streams of RTP and RTCP packets to form VoIP sessions that are recorded in NetData's session table. VoIP/SCCP calls are now handled as thoroughly as VoIP/SIP calls.

The Skinny decoder also recognises and characterises multicast streams of RTP and RTCP packets that broadcast audio data for Music On Hold and paging functions.

With a right-click on a session in the session table or on the session timing chart it is possible to request a packet-timing chart that plots all the Skinny, RTP and RTCP packets of a call, on bands that identify the relevant connections and on bars that identify the relevant transactions, as in the following chart:

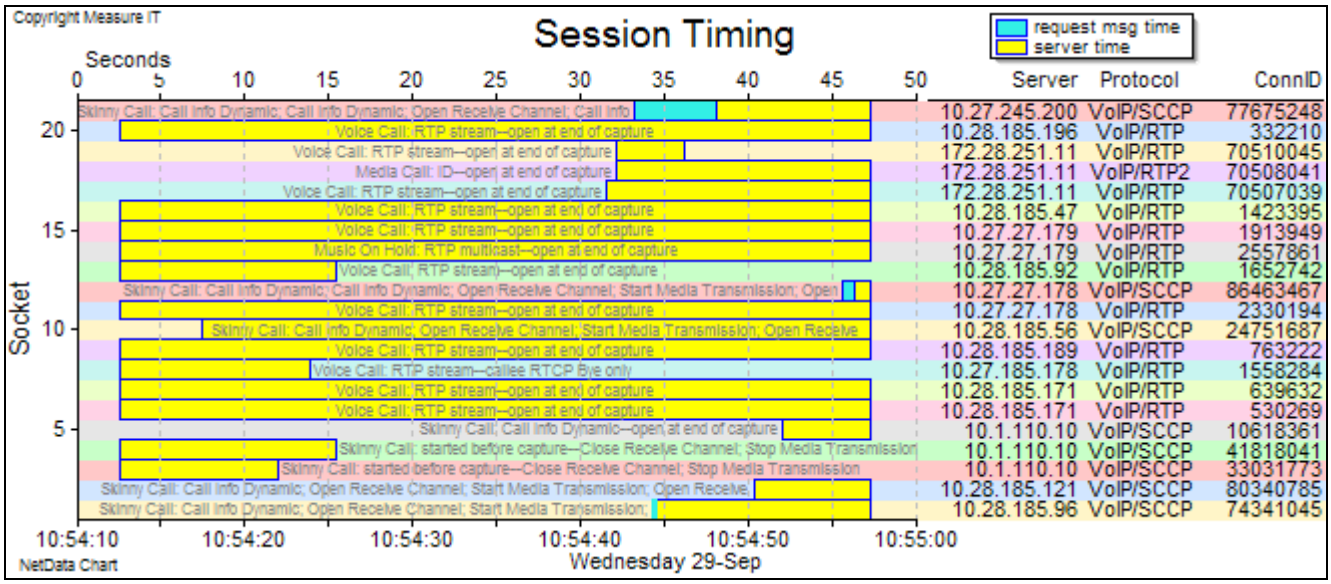


Markers on the pink bands indicate Skinny messages from the phone and the call manager. In this case there were only three request-response pairs: requests to open the phone's receive channel and start media transmission, with their acks from the phone; and a keep-alive round-trip 25 seconds later.

The yellow bars on the pale-blue RTCP connection bands represent a *group* transaction and mark the extent of the sequence of RTCP packets.

The yellow bars on the cream RTP connection bands represent a *user* transaction and mark the time span of the complete call, from initiating Skinny message to the last RTP packet.

An investigation of VoIP traffic usually begins with an overview chart that plots the time span of every call in the captured traffic:



6.11 VoIP SIP Decoder

The SIP (Session Initiation Protocol) decoder handles patterns of SIP transactions seen in a network with an Asterisk PBX and a large number of ATAs (Analogue Telephone Adapters). Some phones address their SIP requests to a proxy SIP server but receive responses from the real server which has a different IP address. NetData searches UDP connections to match responses with their requests and characterise these SIP transactions.

NetData also handles SIP-transaction multiplexing through proxies, when the SIP server issues multiple terminating responses and receives multiple Ack requests to confirm those responses. Some ATAs issue their first Invite request without an authorisation field and receive repeated responses with a status code of 401 (Unauthorized); before the second response is received and acknowledged the ATA may issue an Invite request with an authorisation code. Several SIP Options transactions may be conducted between the same UDP sockets while an Invite transaction is in progress.

When a call request is terminated with a SIP Cancel request NetData indicates this form of termination in the response signature of the pseudo transaction that characterises the call session as a whole. In this respect NetData treats the Cancel request much like a SIP Bye request.

NetData usually creates a session record when it sees a status response of 200 (OK) because it needs to extract information from the SDP (Session Description Protocol) message in the SIP response body (to relate SIP transactions with their RTP data streams). NetData creates a session record when it sees an SDP message in any SIP response, such as a status response of 183 (Session Progress). Some phones start transmitting voice data on receipt of a 183-status (rather than waiting for a 200-status) and this means that NetData can characterise a complete session even when a status message of 200 is not issued by the SIP server or is missed by the sniffer. NetData also creates a session record for SIP Invite requests that receive a failure-status response.

Messages with a 200-status in response to either an Invite or Options request should carry an SDP message. If not, NetData indicates “no body” in the response signature of the SIP transactions, making it possible to search for all such problems in the transaction-class tree and load them onto a chart.

NetData analyses VoIP traffic at much the same speed as other traffic, but takes longer to calculate RTP relative transit times than TCP round-trip times because it performs a statistical regression analysis to correlate sniffer timestamps with voice-encoder timestamps. A capture file with nearly 2 million VoIP packets in 600 Mbytes takes about 20 minutes to analyse but over an hour to calculate transit times. However, it is not necessary to calculate transit times for all packets because when a chart of transit times is requested NetData will offer to perform the calculation for only the packets of the charted connections.

An option in the session table’s context menu will plot all of a session’s SIP transactions and its packets. Other options remain to load charts with all of a session’s transactions and packets; load only its RTP packets; or load only its RTCP packets.

The quickest way to plot the transit times of a particular session is to right-click its row in the session table and from the context menu choose to plot all its packets or only its RTP packets. On the timing chart click the Flow button and let NetData calculate trip times for only the selected dialogue.

6.12 Avaya RTA Interface

NetData decodes traffic of the RTA (Real Time Adherence) interface of the Avaya CMS (Call Management System). This traffic is usually addressed to TCP port 6996 of a third-party WorkFlow Management (WFM) system such as VXI China VisionWFM, Invision Enterprise WFM, Inova LightLink and GMT Planet.

6.13 Universal Alcatel Signalling Protocol

NetData now detects and partially decodes traffic of the Universal Alcatel (UA) proprietary signalling protocol over UDP, usually from port 32128 to phones such as the Alcatel IP Touch.

6.14 RTMP Chunk Stream Decoder

Adobe's RTMP chunk-stream protocol has been widely used to download audio and video files. Part of its popularity may stem from its versatility and efficiency. A single TCP connection can multiplex many streams and many concurrent transactions, and chunk headers may have four different sizes, from 12 bytes down to a single byte. A stream always starts with a 12-byte header to set timestamp, message-length, message type and message-ID information, and, provided the time delta remains constant, all subsequent chunks need only a single-byte header.

NetData matches responses with commands to characterise and measure transaction response times. It constructs special 'user' transactions to characterise the activity of complete streams. It detects RTMP traffic even if the server doesn't use the standard port number (1935), and has extensive heuristics to determine a stream's chunk size even when the sniffer drops many packets. The decoder is complex because there is so little redundancy in RTMP headers.

6.15 MGCP Decoder

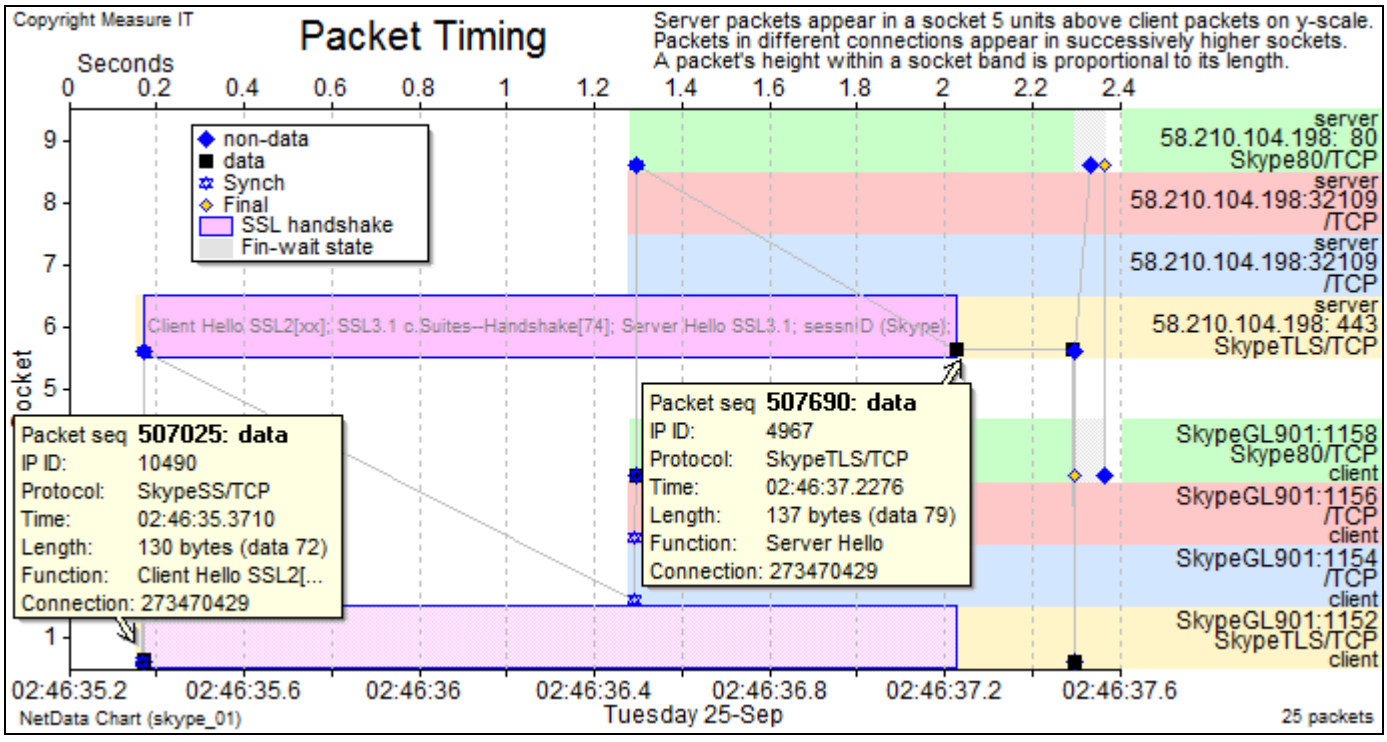
NetData decodes the traffic of the Media Gateway Control Protocol (MGCP) which is used between a Call Agent and a Media Gateway.

7 Peer to Peer

7.1 Skype Traffic Detection (2012)

Skype Internet telephony traffic is based on a peer-to-peer architecture and employs obfuscation as well as encryption to foil traffic-blocking attempts. However, NetData recognises patterns in the first data packets of some TCP connections and will tag as Skype traffic (*SkypeD*) all TCP and UDP connections to the same ports. It also recognises attempts to establish media connections through TCP port 80 (tagged *Skype80*) and port 443, and connections with what is believed to be a login server (tagged *SkypeLS*) through either port 80 or port 443. Connections with a Login Server (LS) begin with a null SSL3 Handshake.

Media connections through port 443 begin with an SSL2 Client Hello record and are first tagged *SkypeSS*. If port 443 has a Skype listener it responds with an SSL3 Server Hello record whose random string and session ID are both Skype constants (embodying the timestamp 18:23:18 31/01/2004), and NetData changes the tag to *SkypeTLS*. If a conventional HTTPS service responds then NetData changes the tag to *HTTPS*. The Skype client requests connections through ports 443 and 80 at the same time, and when one is successful the other is closed, as illustrated by the following timing chart:



If port 80 has a conventional HTTP service that responds to the Skype message with some error message, the new port-80 connection is closed immediately.

NetData also recognises Skype IP address exchanges in UDP traffic and characterises them as three-way transactions that start with a probe packet (type 2) from one node, receive a public-address indication (packet type 7) from the other node, and conclude with an address indication (type 3) from the first node. Probe packets that don't progress to an address exchange are recorded as probes without responses.

7.2 BitTorrent Local Peer Discovery (LPD)

NetData detects traffic from BitTorrent and similar programs that search for peers using the Local Peer Discovery (LPD) protocol with UDP port 6771.

7.3 Short Message Peer-to-Peer Protocol

NetData decodes traffic using the Short Message Peer-to-Peer Protocol (SMPP). The default TCP port for this service is 2775, but NetData detects SMPP with any port number. If the port number is 8490 NetData assumes that the traffic is handled by the Telstra Short Message Service (SMS) Access Manager and tags the application as ‘SMPPtam’.

7.4 BitTorrent

BitTorrent, a peer-to-peer file-sharing protocol, was once said to account for 25% or more of all Internet traffic. It was designed to spread processing load and bandwidth requirements across client machines, reducing the load on the file provider, when a large file is downloaded by many clients. Wikipedia provided the following warning on network impact: "Routers that use NAT, Network Address Translation, must maintain tables of source and destination IP addresses and ports. Typical home routers are limited to about 2000 table entries while some more expensive routers have larger table capacities. BitTorrent frequently contacts 300–500 servers per second rapidly filling the NAT tables. This is a common cause of home routers locking up."

The BitTorrent decoder recognises all message types, parses handshakes, and parses ‘bencoding’ texts found in extended handshake messages. It also decodes the Tracker protocol that runs on HTTP, and the alternative Tracker protocol that runs on UDP. Tracker/HTTP responses are ‘bencoding’ texts that are also decoded fully, even if compressed with gzip.

A download begins by retrieving from a web server a .torrent file that contains information about the file to be downloaded. It is a ‘bencoding’ text, and an SHA1 hash of its ‘info’ section serves as a key to identify the downloaded file in all transactions with trackers and peers of the client. This info hash is recorded in the user-ID field of NetData’s connection and transaction records, making it possible to load from the database all the transactions involved in a particular download (i.e. ‘torrent’). The _conn.log file includes a table of all the torrents found in a capture sequence, with their info hashes, and the easiest way to investigate a torrent’s activity is to paste an info hash from that table into the Load User field of the load-data window, and load all the related transactions. As with SMB file-reading and -writing transactions, NetData can plot transfer rates and movement in the file pointer as pieces of the file are exchanged with peers.

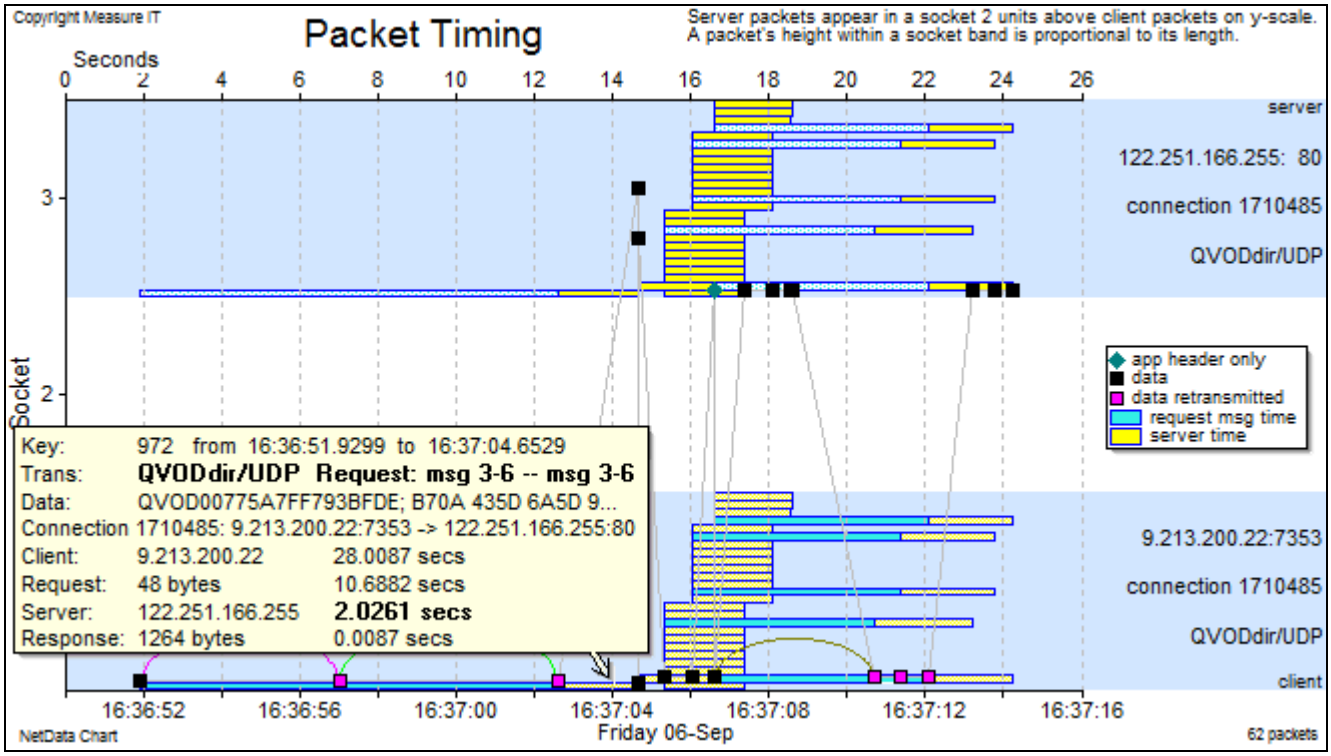
7.5 Lotus Sametime Connect Peer-To-Peer File Transfer

NetData decodes the peer-to-peer file transfer traffic of Lotus Sametime Connect that uses port 5656. It is tagged as ‘ICTfileT’.

7.6 QVOD Peer-to-Peer Video Player and Streaming Media Server

QVOD Player from Shenzhen Qvod Technology Co is a free video player that will also act as a streaming media server to share files peer-to-peer. It has been adopted by thousands of small and medium-size websites to distribute video without the expense of large server farms, and at the end of 2011 had more than 200 million users in China. It should not be confused with a streaming-video product from InfoValue Computing Inc also called QVOD (QuickVideo OnDemand) that uses the standard Real-Time Streaming Protocol (RTSP).

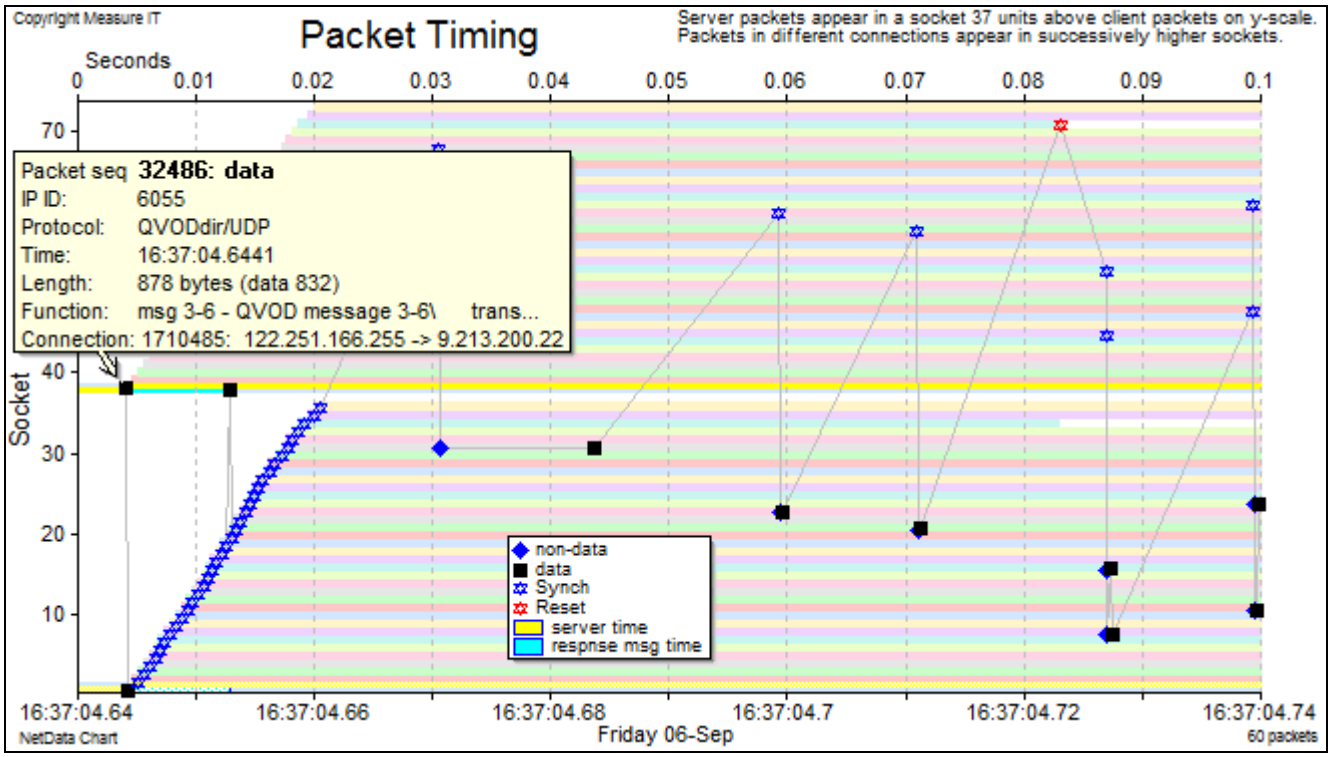
An individual user may exchange compressed and possibly encrypted media streams with other QVOD players, using dozens of concurrent TCP connections. The IP addresses and port numbers of peers are obtained in TCP or UDP transactions with port 80 of what NetData describes as a *directory* server for the required media file. NetData decodes the directory transactions but cannot decode the connections that exchange media streams with peers. However, if it finds their socket information in a directory transaction it is able to tag them (with 'QVODstrm').



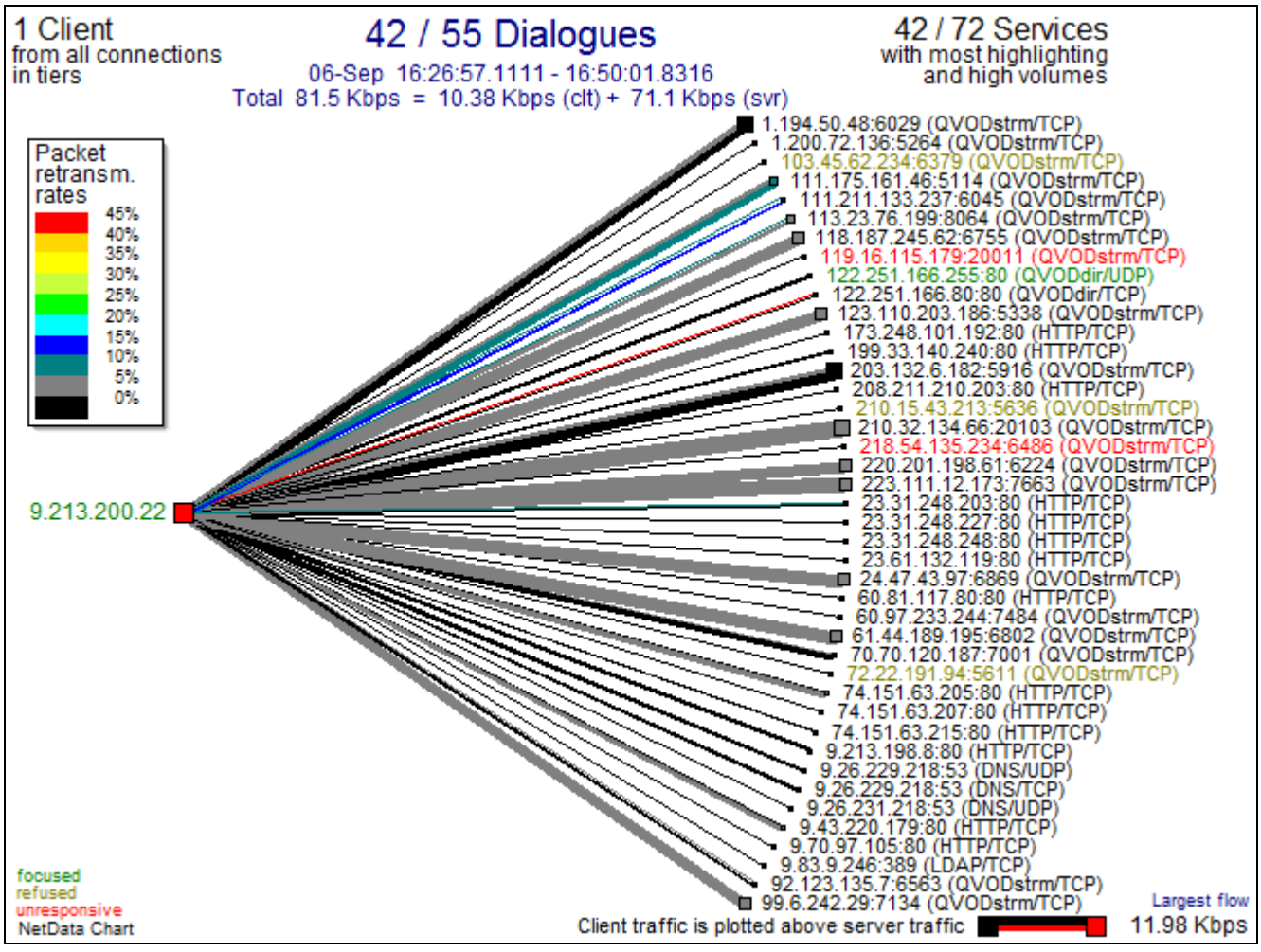
The first UDP transaction in this group requested a list of peers. The request had to be retransmitted twice before it received a response, in two UDP packets. Immediately on receiving the first response packet, which conveyed a table of 40 peer socket addresses, it attempted to open TCP connections with 35 of those sockets to start streaming data from them, and initiated 25 concurrent UDP transactions with the QVOD directory server, as shown by the above chart. The patterns of behaviour illustrated by this and the following two charts may help others recognise the presence of QVOD traffic on a network

```
peers                    40
  ID                peer address:port   type
  ----------------------------------------
  B8663FC69FF4  219. 68.103.193:6431   0303h
  B866DE6C26DA  101.131. 67.122:7973   0003h
  B8670FA76F59   92. 97. 52. 95:6563   0003h
```

This is part of NetData's table of peer addresses in a response from a QVOD directory server.



This chart describes the first response packet with a list of peer addresses and plots the subsequent Synch packets requesting connections with 35 peers. Some ignored requests and others refused connections.



Advice found on web: "*uninstall this player please..for your own good..bcoz this application will run background and it occupied your bandwidth silently..in order to share files to others..so..eventually slow down your connection..kill it..*"

8 Mail Service

8.1 Experian QAS (QuickAddress)

NetData decodes transactions of the QAS application that is used for quick entry and verification of mail addresses. The protocol is usually found on port 2021.

9 Messaging Systems

9.1 WhatsApp Traffic

The performance analyst can learn little from the content of WhatsApp messaging and voice-call packets because it is securely encrypted, but, given that WhatsApp has more than a billion users, it may be important to recognise the presence of WhatsApp traffic on a network, and to see what connections are involved. If there are performance problems, much can be learnt as usual from packet timing.

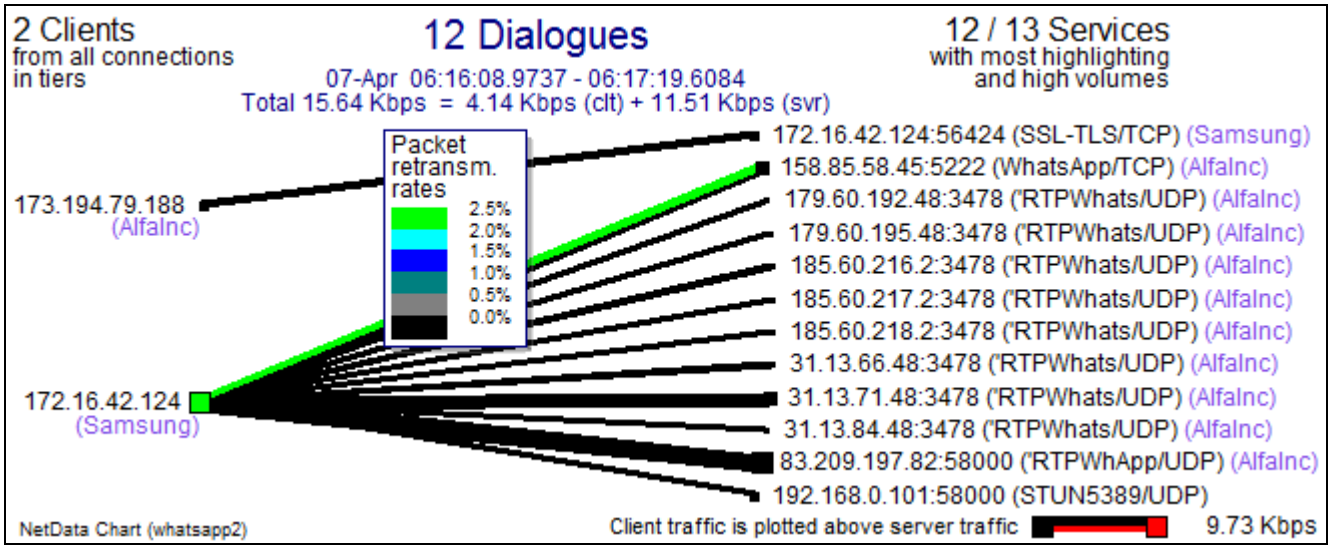
NetData recognises WhatsApp voice traffic as well as its messaging traffic and is able to demultiplex the RTP, RTCP, STUN TURN (Traversal Using Relays around NAT) and keep-alive data streams that flow between various pairs of UDP sockets. The RTP decoder not only handles RTP-RTCP multiplexing, but also tracks gaps in RTP sequence numbers; it reliably distinguishes between lost, overtaken, and retransmitted or duplicated packets. RTP sequence tracking uses the same windowing scheme used by NetData's DTLS and IPsec ESP decoders. The flow chart can plot RTP sequence numbers, as it does for the other two protocols, to reveal more clearly the patterns of any sequence abnormalities (see examples below).

WhatsApp traffic has several distinguishing features. All the voice traffic handled by a phone uses a single UDP port, which has UDP 'connections' with several Facebook servers and the other phone. NetData recognises two different dialects for connections exchanging RTP packets, and they are tagged `RTPWhApp` and `RTPWhats`. The first type of connection usually joins the two phones directly and mixes RTP, RTCP and STUN (NAT traversal) packets. The RTP packets convey the two audio streams with a payload type of 120 and the phones use an Opus encoder with a sampling clock rate of 16,000 ticks per second.

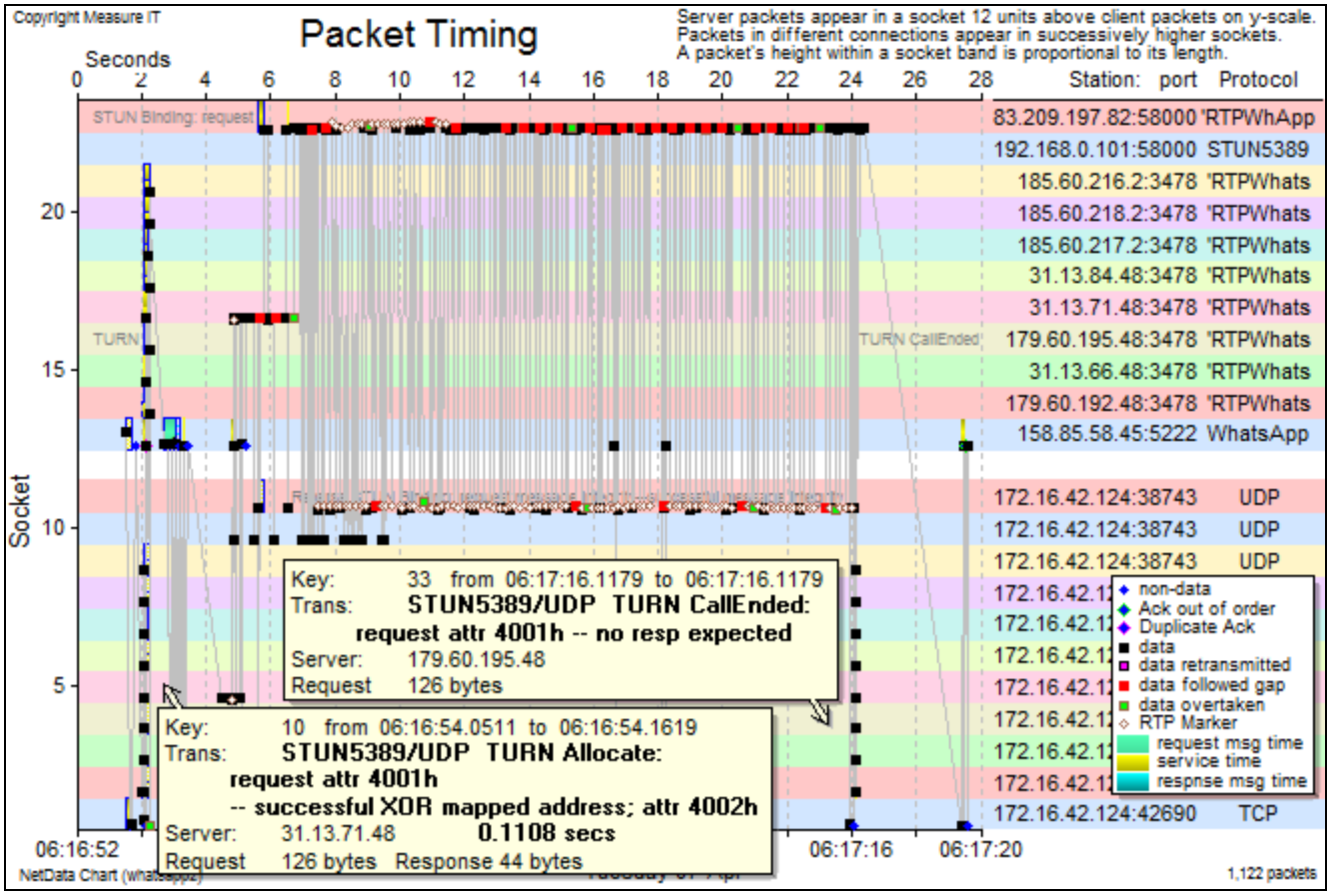
The second type of connection, `RTPWhats`, links a phone to several Facebook servers located near both phones. In general, these connections multiplex the three types of packet handled by the first type of connection, and add a fourth type of packet that serves a keep-alive purpose. At 3-second intervals the phone (client) sends the 6-letter text 'health' to the Facebook server and usually receives the text 'OK' in response. NetData records these round-trips as server transactions.

The UDP socket of a phone is regarded by NetData as a client, largely because it initiates round-trips to Facebook servers which use port 3478, the standard listening port assigned for STUN servers.

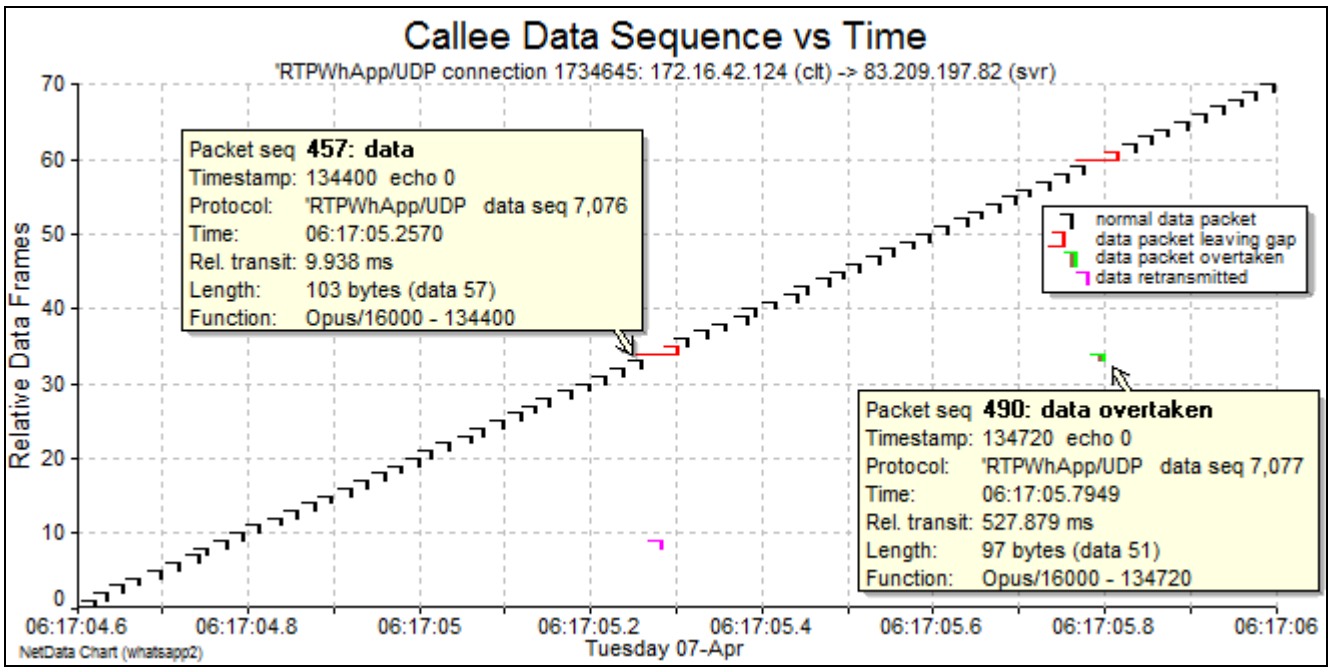
A more thorough description of WhatsApp handling of voice calls is provided by *WhatsApp Exposed, Investigative Report*, by Philipp Hancke (Chief WebRTC Engineer), May 2015. That report is accompanied by a set of nine capture files, and the capture file of Session 2 generated the following four sample charts.



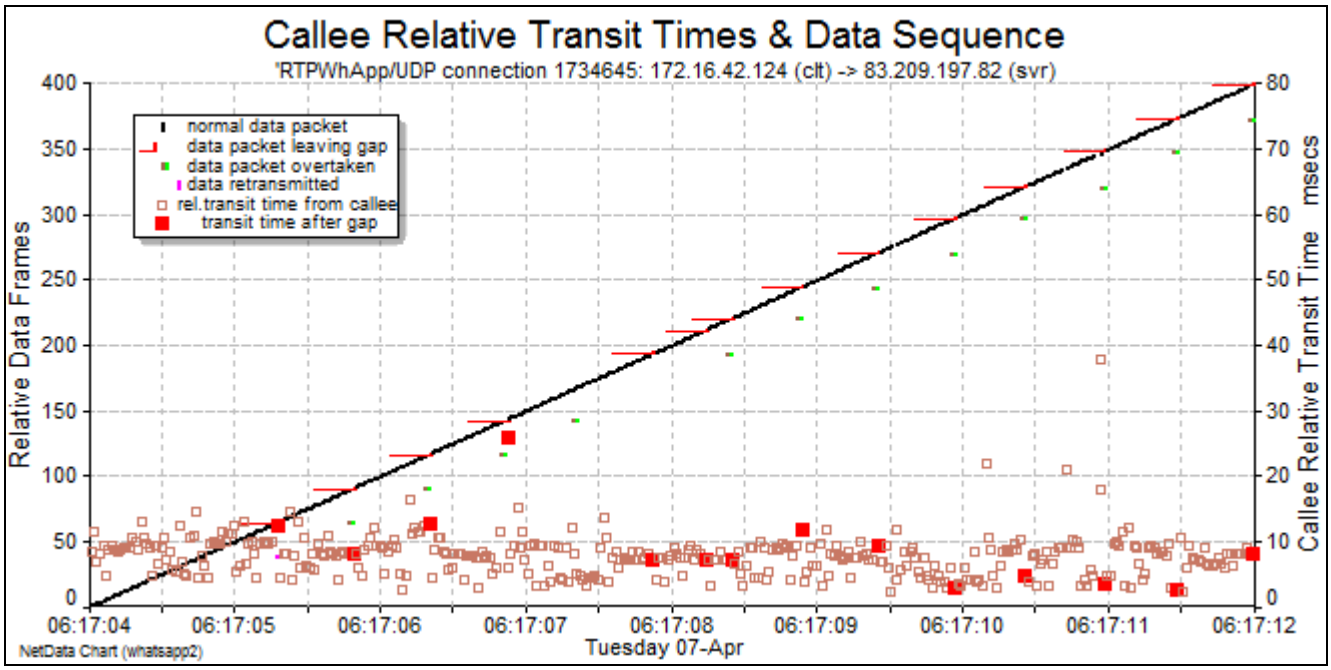
The above dialogue chart shows the client Samsung phone on the left connected to a messaging server (158.85.58.45) port 5222, eight Facebook STUN servers in several locations using the RTPWhats dialect to port 3478, and the other phone directly (83.209.197.83).



Call audio data was initially relayed through the Facebook server at 31.13.71.48 (chart sockets 5, 17) and then exchanged directly (sockets 11, 23) with many sequence gaps (red markers).

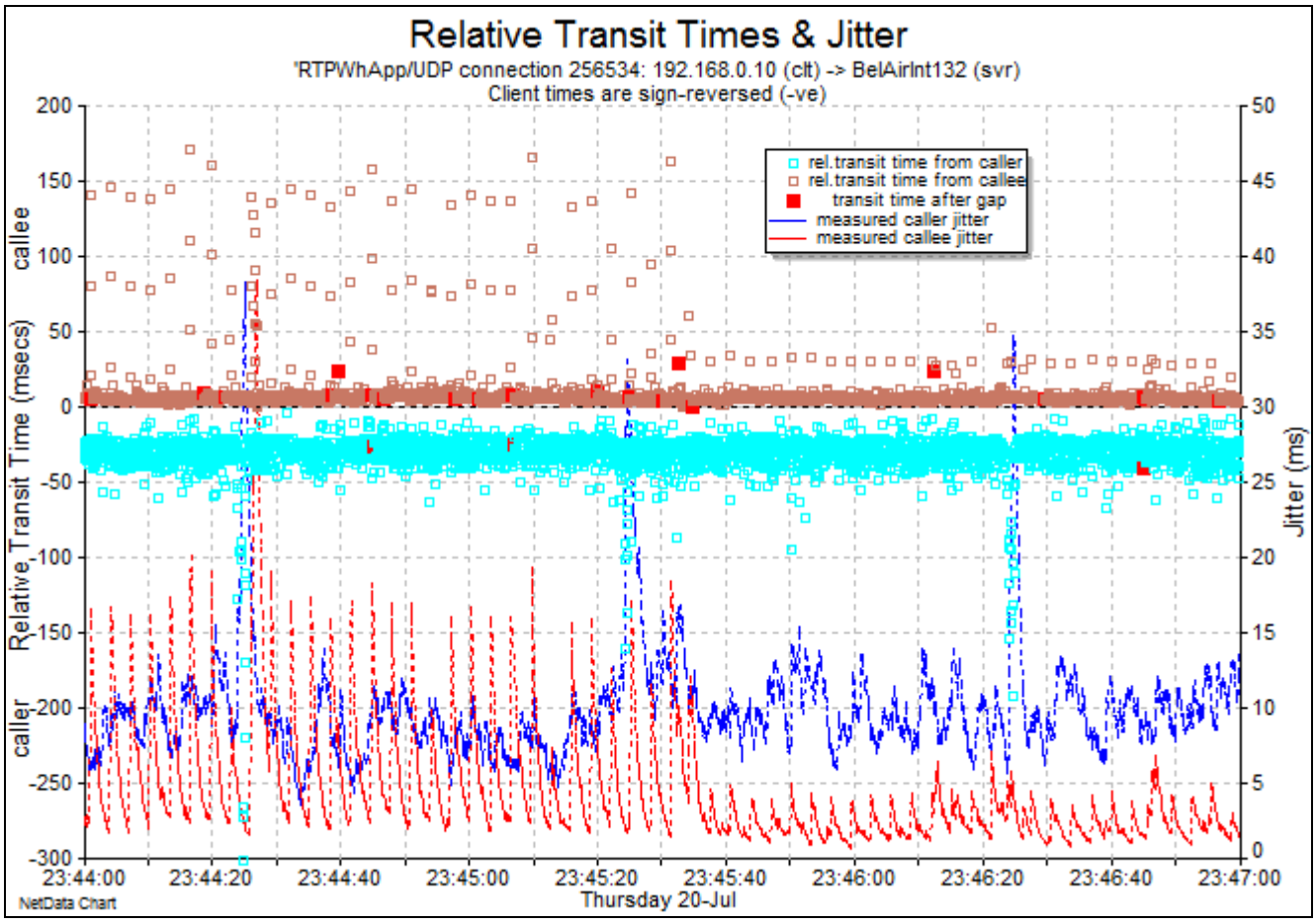


The chart of callee RTP packet sequence numbers illustrates a strange anomaly in that every half a second, one packet was sidelined and forwarded at the next half-second event. Judging by the RTP sequence numbers (7,076 and 7,077), packet 490 should have followed packet 457.

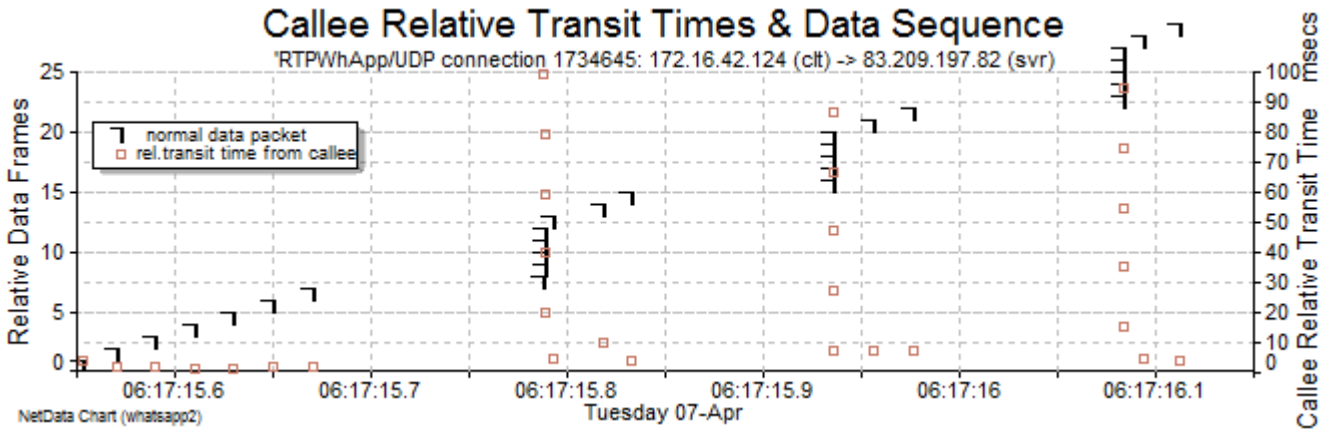


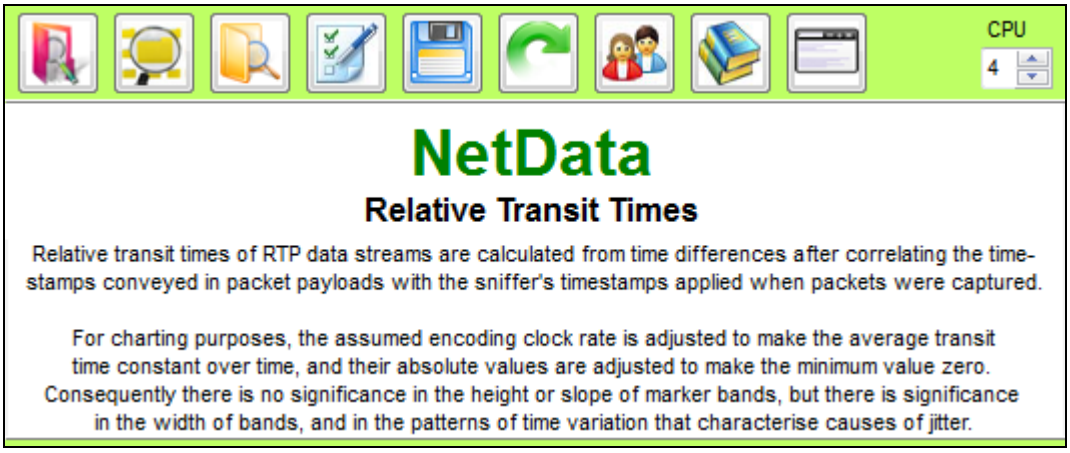
Over a span of 8 seconds this chart of callee sequence numbers confirms the regularity of delayed packets, and the overlaid markers of relative transit times show that most varied by less than 10 ms.

The following chart depicts a recent WhatsApp call that demonstrates NetData's ability to reveal changing patterns of jitter behaviour. Callee jitter dropped substantially half way through the call and then was smaller than the caller jitter, even though the traffic was captured only one WiFi link away from the caller in Sydney, and the callee was in Los Angeles.



The following chart reveals a more serious cause of jitter – several blockages each lasting 100 ms.





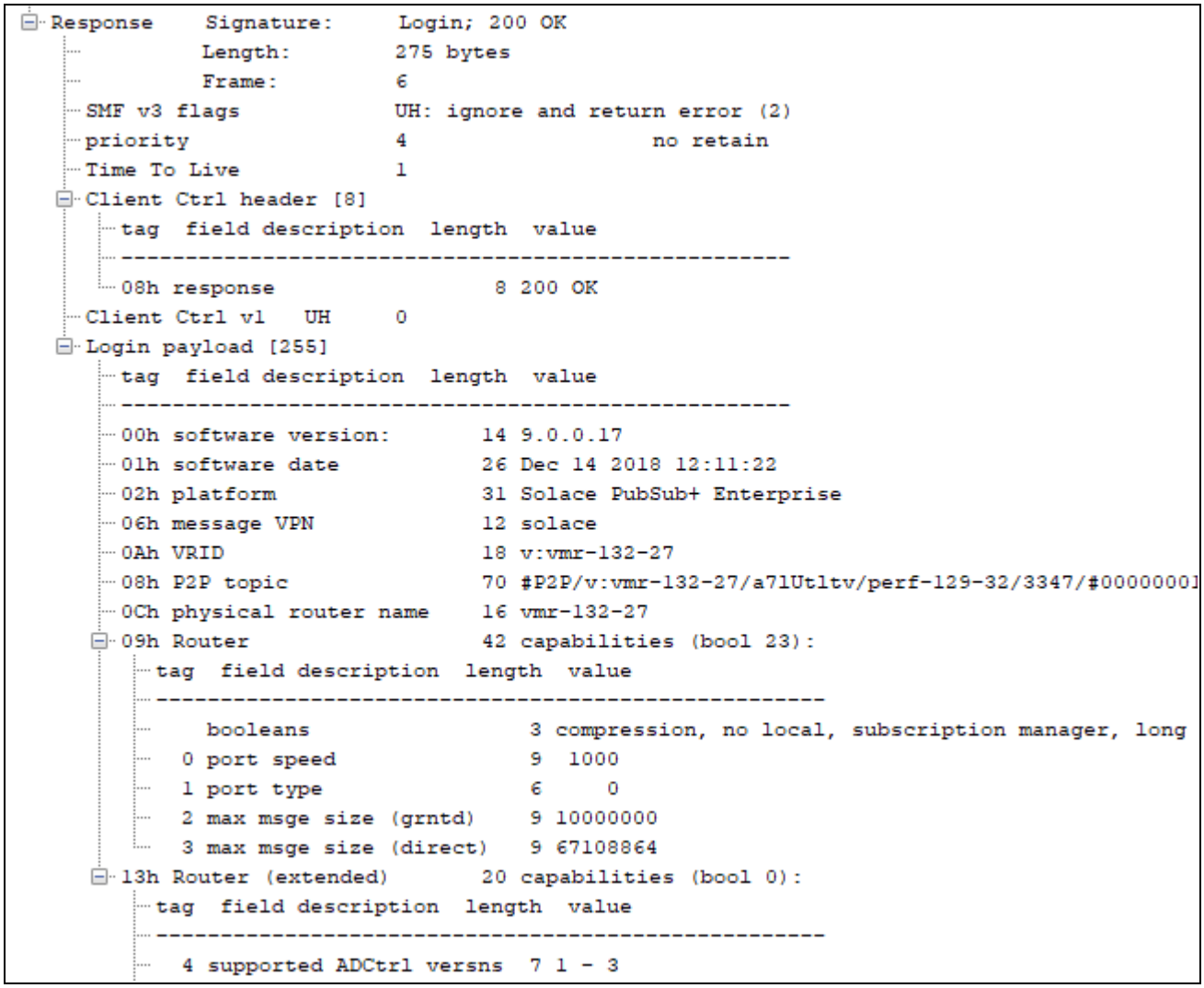
The WhatsApp messaging decoder has also been upgraded to make more realistic characterisations of transactions – deducing relationships between client and server messages that occur in quick succession – even though content encryption makes the results uncertain. Round-trips may be initiated by both client and server independently, and complete messages comprise a sequence of data blocks preceded by their length indicators. Most blocks are sent in separate packets. NetData characterises messages in terms of the lengths of their constituent blocks, much as it does with patterns of SSL record lengths for HTTPS messages.

9.2 Solace Message Format

The Canadian company Solace provides message-oriented middleware appliances and software capable of routing messages between disparate messaging systems using such widely-used protocols as JMS (Java Message Service), MQTT (OASIS Message Queuing Telemetry Transport for IoT), AMQP (Advanced Message Queuing Protocol), REST (using HTTP features) and WebSocket. Besides handling those common protocols, Solace also uses its own proprietary protocol, SMF (Solace Message Format). NetData now decodes SMF messages.

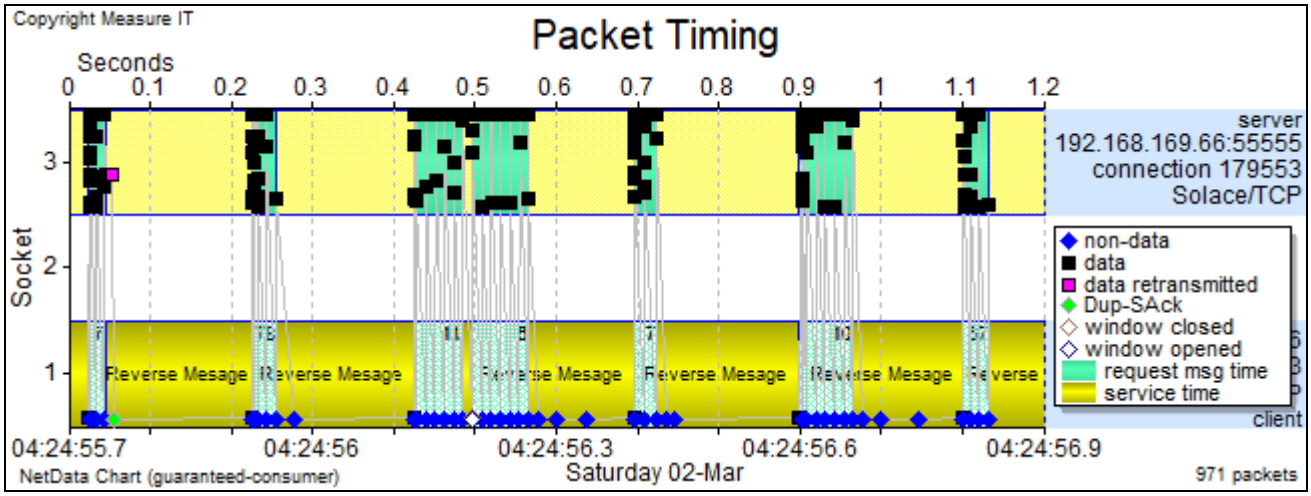
SMF uses a variety of subsidiary protocols for different purposes including Client Control, Subscription Management, Assured Delivery Control, Topic-Routed Messaging, and Keep-Alive. NetData identifies the protocol, the protocol's message type, and all the parameters associated with each message type in both the header and body of each message. Parameters use a Tag-Length-Value (TLV) format and NetData displays parameters in tables with an additional column that describes each parameter.

Solace messages may be compressed using the Deflate function of the standard zLib library. NetData recognises compressed messages and will be able to fully decode them provided that the capture file includes all the packets of the connection since it was opened, and packets have not been truncated. The loss of any data in a connection invalidates its compression dictionary.

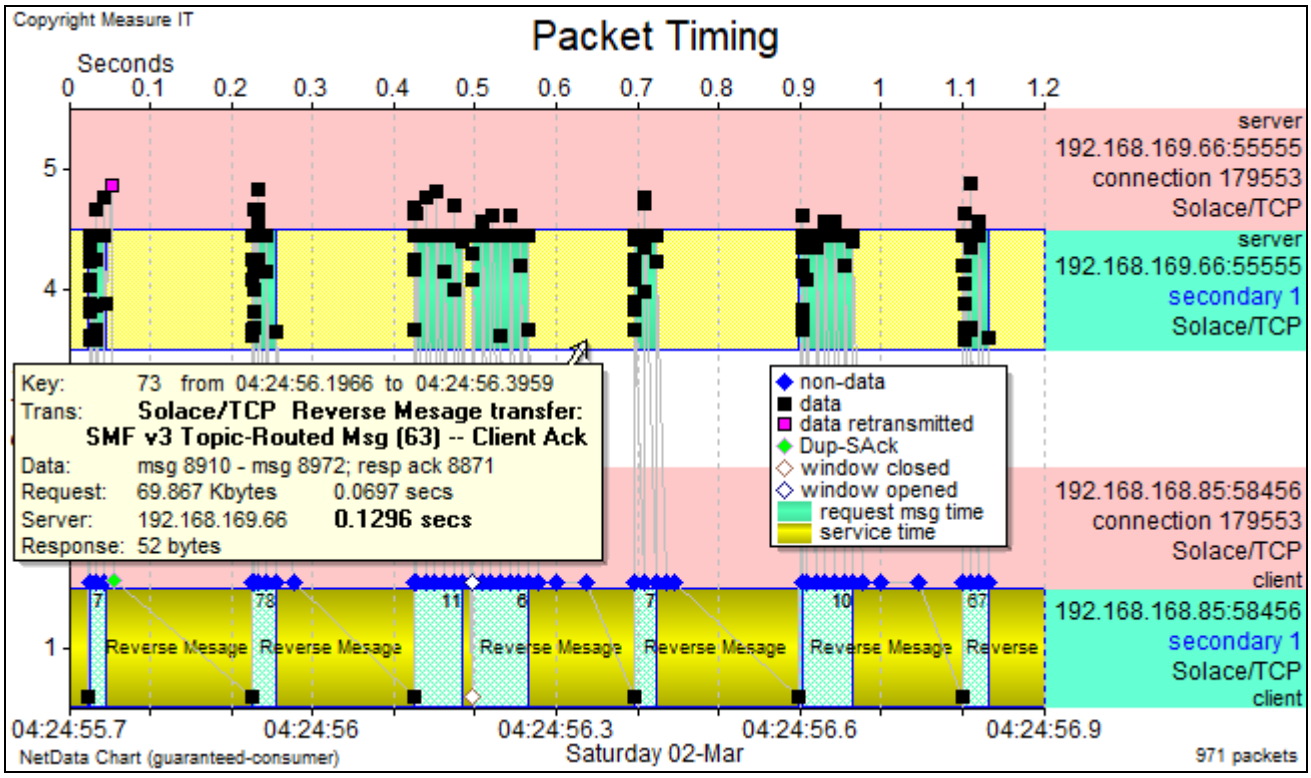


9.2.1 Assured Delivery and Flow Control

Assured-delivery (AD) messages are associated with a flow ID and many flows may be handled by a single TCP connection. Each flow is subject to a separate flow-control mechanism that assigns sequential message IDs to AD messages, specifies a window size – the maximum number of unacknowledged messages (typically 255) – and control messages that acknowledge AD messages up to a specifield ID.

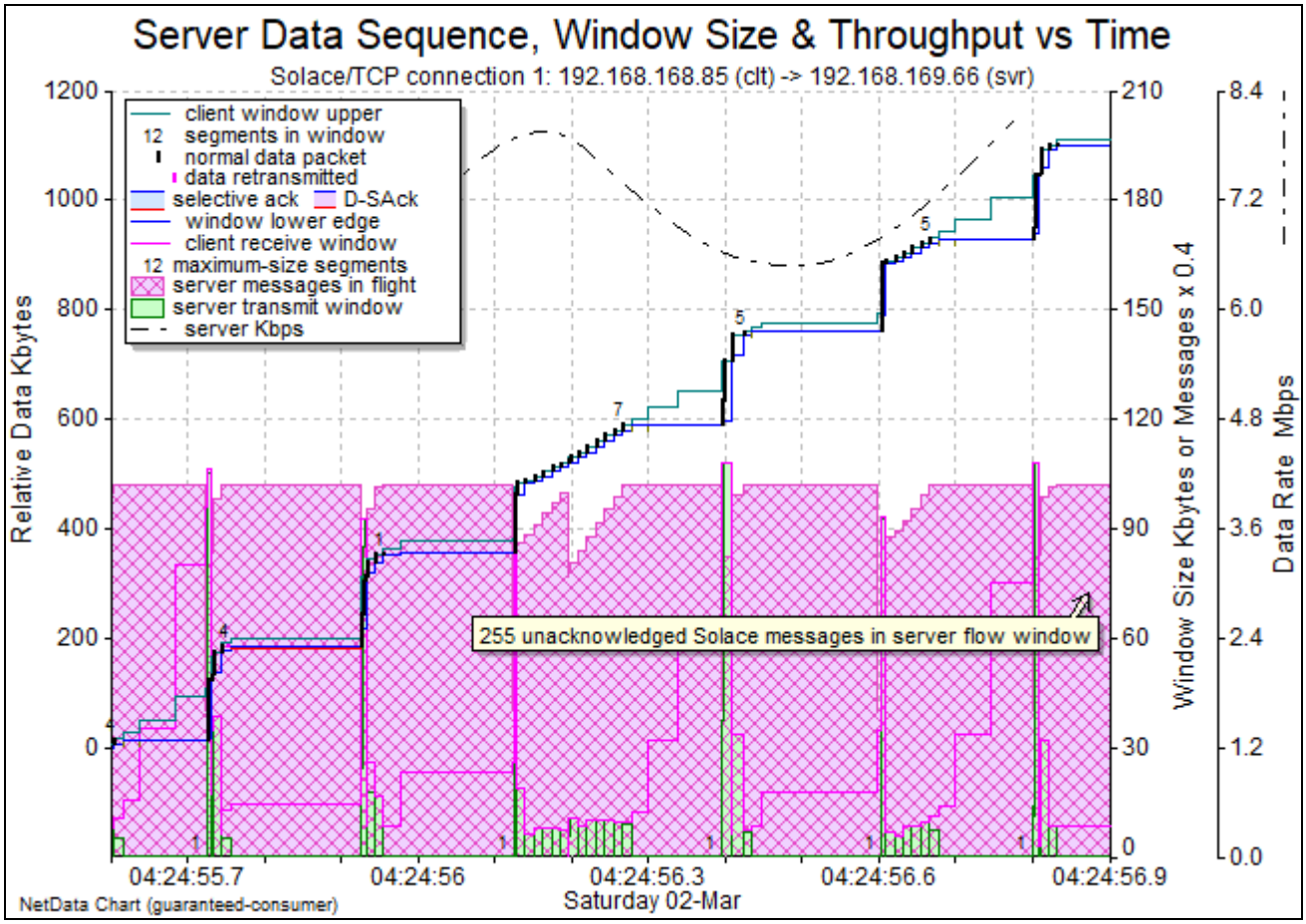


NetData treats flow IDs as secondary connection IDs and the timing chart can display the individual flows on separate bands as in the example below which transfers messages with a flow ID of one (‘secondary 1’). NetData characterises the flow with pseudo transactions that start with a seqiuence of AD messages and end with a ‘Client’ Ack. The popup on the sample chart indicates that the client Ack doesm’t necessarily acknowledge all the messages in flight.



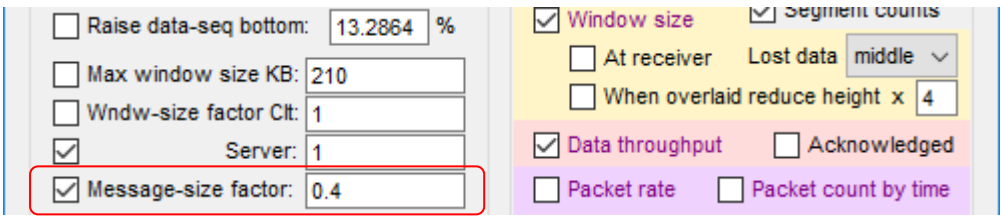
9.2.2 Messages In Flight

NetData is able to overlay graphs of both the TCP bytes-in-flight and Solace messages-in-flight on the flow chart with the TCP sliding window, as in the chart below. A particular Solace flow is chosen by right-clicking its secondary-connection band on the timing chart (as above) and choosing to 'Plot Flow Chart of This Connection'.



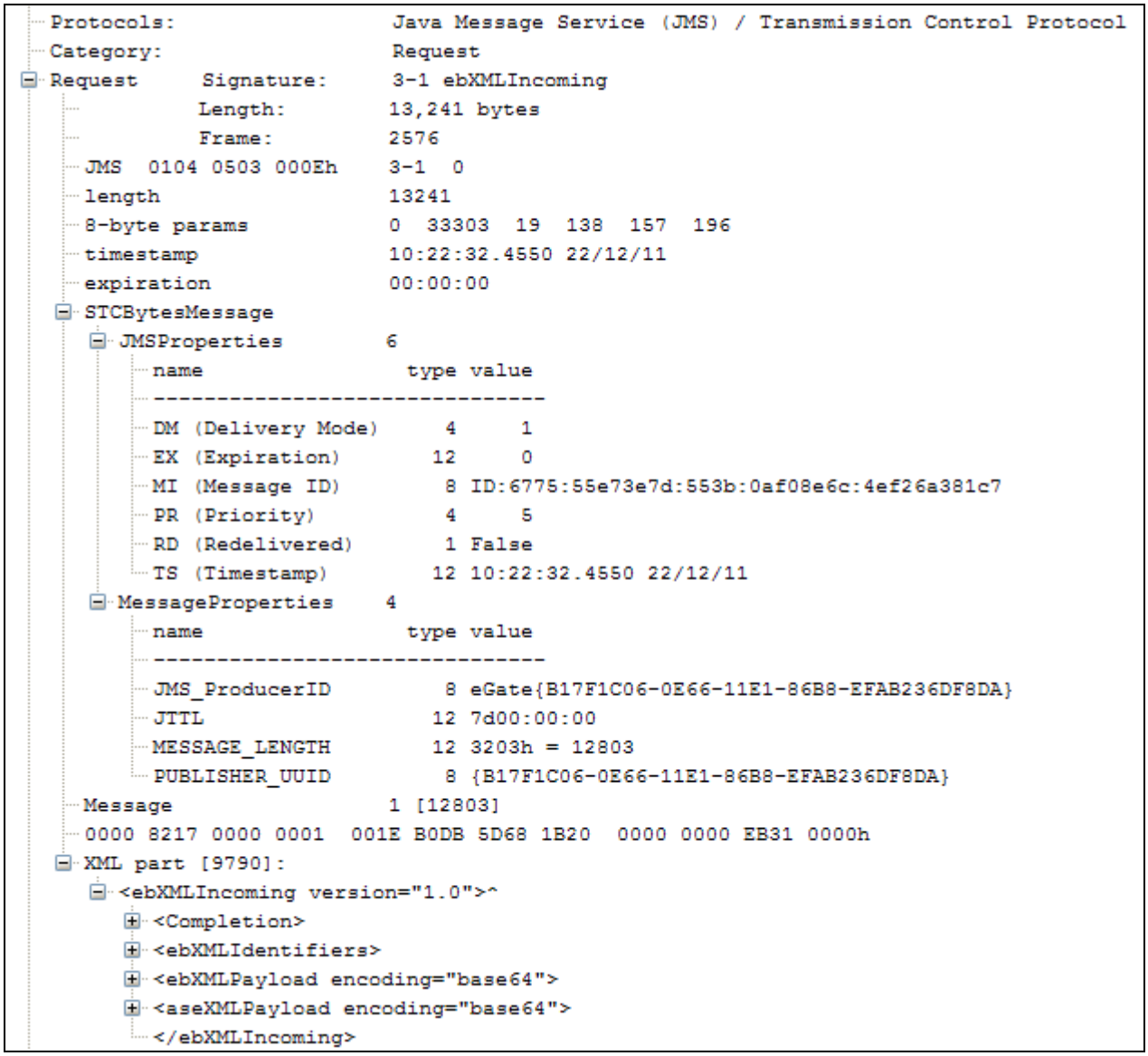
The flow in this case was constrained largely by the receiver's ability to secure received AD messages – perhaps by writing them to disk – and issue client Acks. This bottleneck caused the flow window (messages in flight) to remain full most of the time. The bottleneck was also apparent in the TCP window which closed rapidly and throttled the flow until it was forced to stop when the Solace window became full. The weakness in this flow control arises from the paucity of client acks which seem to appear only when the window could be opened to half its size. The consequence was that most client acks were followed by a large and rapid burst of packets that could put undue pressure on queue buffers in network equipment.

The messages- and bytes-in-flight graphs share the same scale. Initially a message is equated with 1000 bytes and NetData applies a factor other than one as necessary to give the two window graphs a similar height. The automatic message-size factor can be adjusted manually in the format-control window:



9.3 Java Message Service Decoder (2012)

The Java Message Service (JMS), part of the Java Platform Enterprise Edition, allows loosely coupled, reliable, and asynchronous communication between JMS clients within a distributed application. NetData has a decoder for JMS traffic, but because JMS is defined as an API—an interface between client and JMS provider—and is not a wire protocol, the decoder may not be able to handle JMS traffic in all Java environments.



This description of a JMS message displays the 8-byte parameters in the initial header, a set of standard JMS message properties, and a set of application-specific properties. The message body has binary-coded data preceding a large XML message, and that message embeds two XML messages encoded in base-64. NetData decodes expiration times and timestamps encoded as *Java milliseconds*, a number of milliseconds since the start of 1970.

The JMS provider of this protocol uses pairs of connections to Put and Get messages in queues. One connection might poll a queue regularly to get a message, and when a message is returned the second connection of its pair conducts a round-trip with the provider to confirm receipt of the message. In this case NetData records a group of three transactions: two for the conventional round-trips, one on each connection; and the third to measure the client's response time to issue the receipt confirmation. Another connection sends messages to a queue without getting a response on the

same connection, but the second connection of its pair issues a request for confirmation of message receipt. NetData records all that activity in characterising a single transaction that uses both connections.

9.4 SwiftMQ JMS for VMware VDM Connection Servers

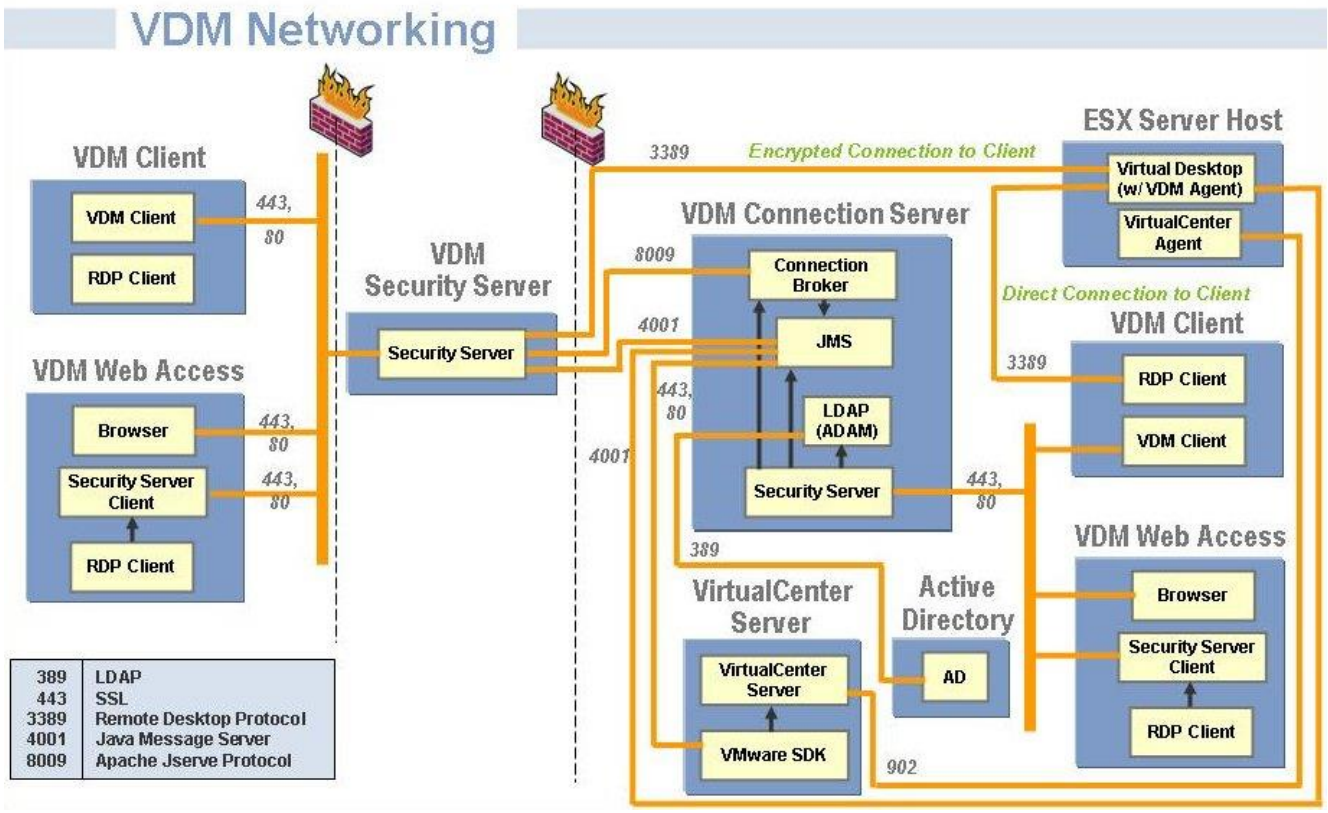
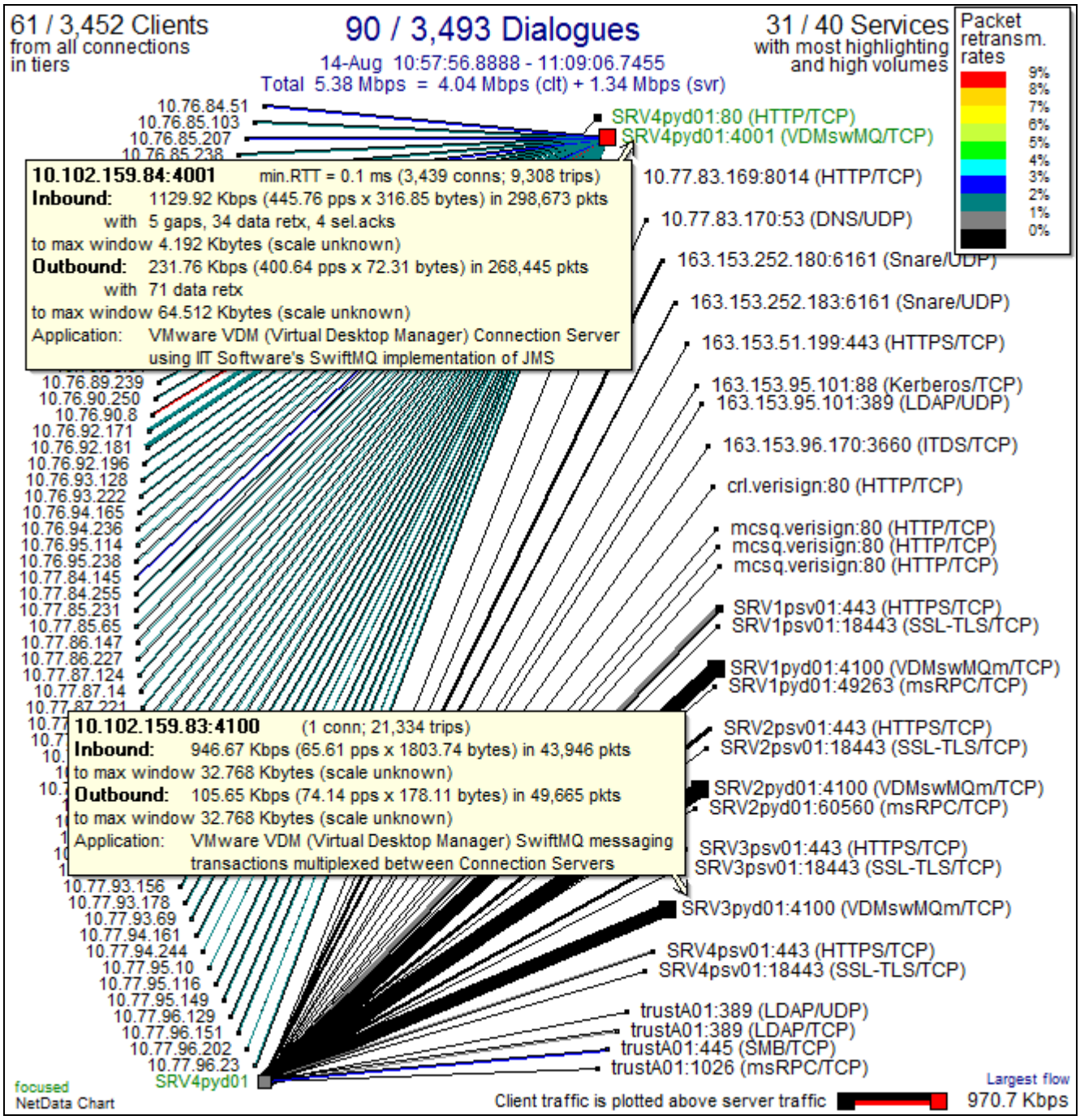


Diagram from VMware web site

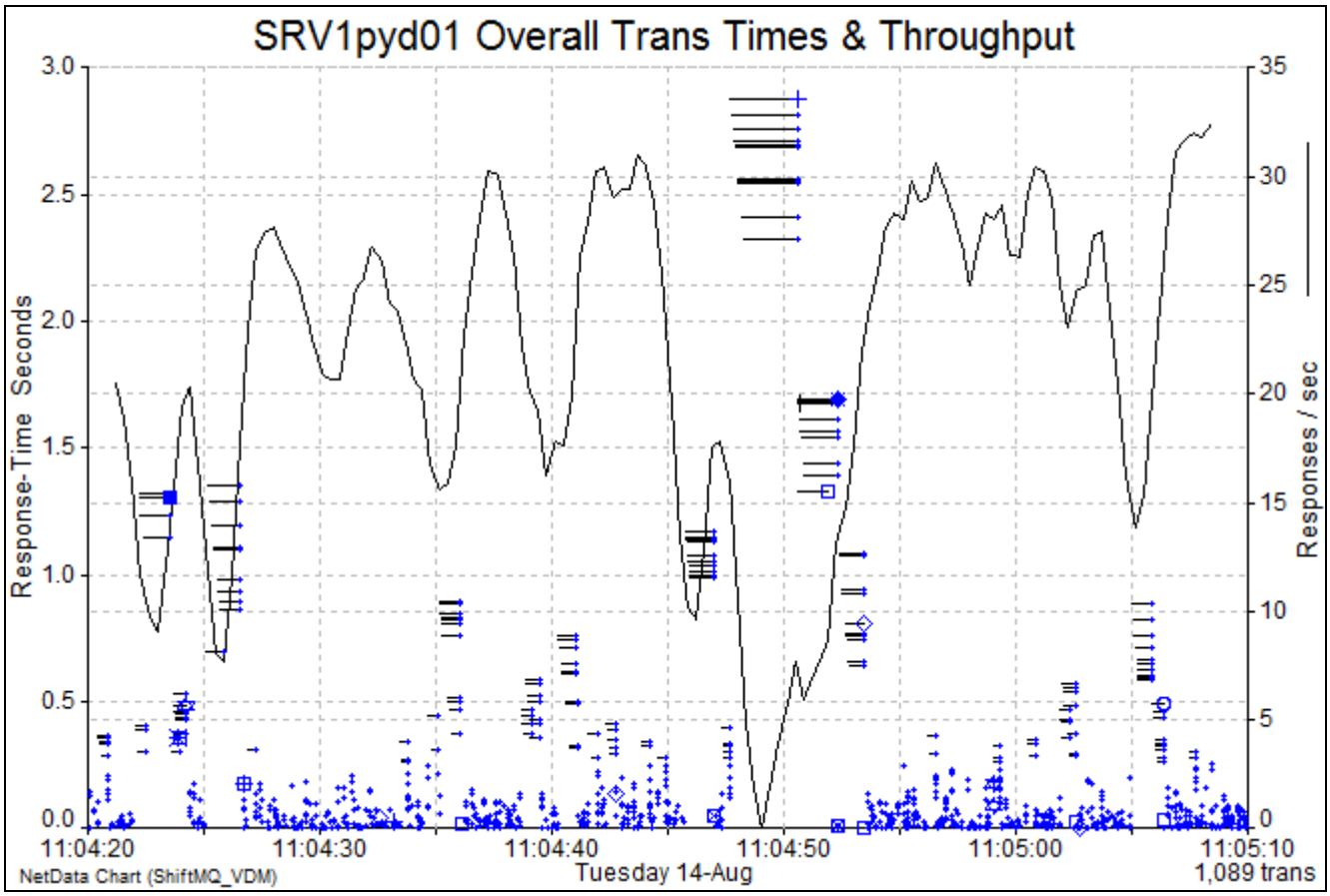
A VMware VDM (Virtual Desktop Manager) Connection Server is accessed by its VDM clients through TCP port 4001 using the Java Message Service (JMS), and connection servers in a cluster exchange terminal information with each other, also using JMS, through port 4100. NetData decodes VDM connection server traffic when JMS is implemented with SwiftMQ, a product of the German middleware specialist IIT Software.

Because JMS standardises an API and not a wire protocol, NetData could encounter different connection-server protocols in a VMware system, and the SwiftMQ decoder is NetData's second JMS decoder (see above). In a Windows environment, traffic captured on a connection-server machine revealed the following dialogues:



This chart separates the front- and backend tiers. The front-end connects the connection server through port 4001 to more than 3000 VDM clients, and the backend includes fairly busy connections with three other connection servers in the cluster. In both types of connection, round-trips are initiated by both the client and server, and those initiated by the server usually become a ‘three-way handshake’ when the server issues an acknowledgement to the client’s response. NetData records the response-times of both parts of such handshakes in separate transactions.

Apart from server acknowledgement of client responses, the backend protocol is entirely symmetrical, with both client and server sending messages in the same type of envelope (format 122). Furthermore either node may issue many send-message requests concurrently, and to match responses with their requests and their acknowledgements the header of every protocol message includes a transaction ID that comprises a stream ID and a sequence number. The following chart displays the transaction response times on a backend connection.

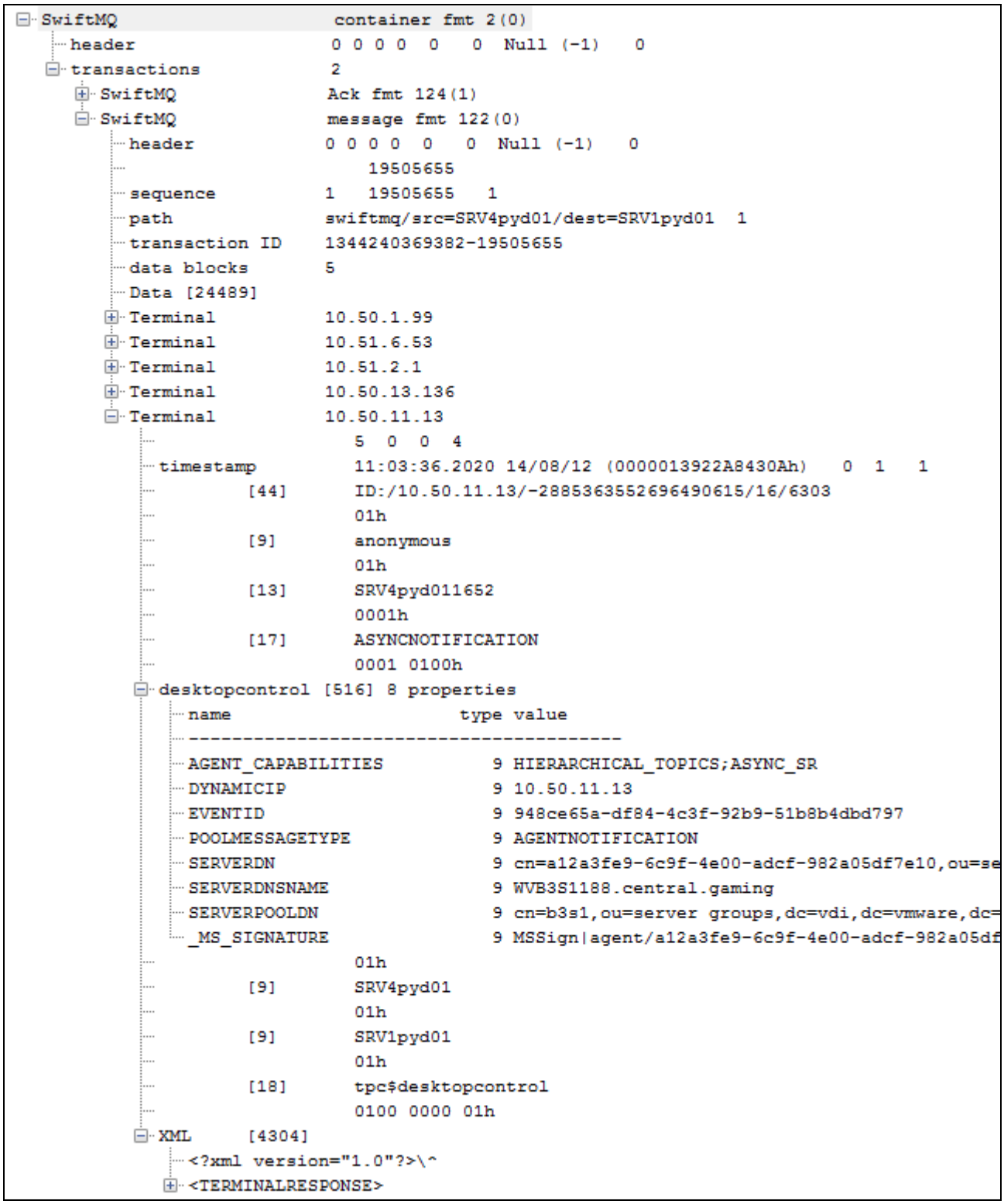


In this 50-second period the connection server exhibited a propensity to block send-message requests and the client stopped sending new requests when 11 requests were in progress. It is believed that this blocking behaviour was the consequence of an overloaded server.

ShiftMQ improves networking efficiency by sometimes assembling several protocol messages into a special container message. The following container message (format 2) conveys seven acknowledgement messages (format 124), each with a different transaction ID:

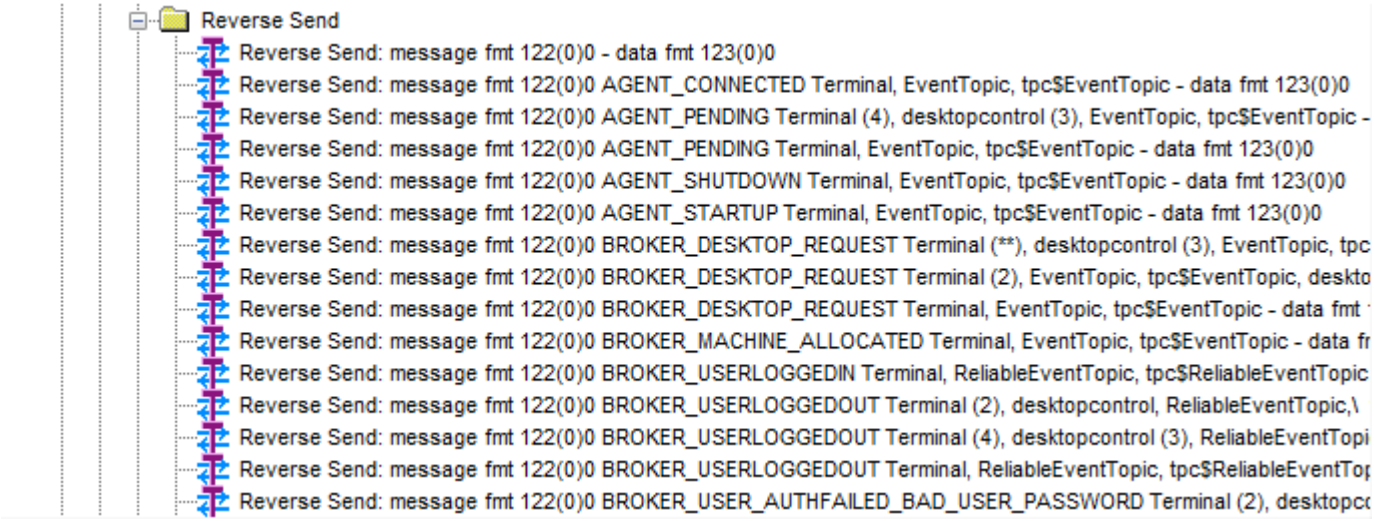
```
SwiftMQ                container fmt 2(0)
  header               0 0 0 0  0   0  Null (-1)   0
  transactions         7
    SwiftMQ            Ack fmt 124(1)
      header           1 0
      sequence         1   12763570   1
      path             swiftmq/src=SRV2pyd01/dest=SRV4pyd01  1
      transaction ID   1344085172823-12763570
    SwiftMQ            Ack fmt 124(1)
    SwiftMQ            Ack fmt 124(1)
    SwiftMQ            Ack fmt 124(1)
    SwiftMQ            Ack fmt 124(1)
    SwiftMQ            Ack fmt 124(1)
    SwiftMQ            Ack fmt 124(1)
      header           1 0
      sequence         1   12763576   1
      path             swiftmq/src=SRV2pyd01/dest=SRV4pyd01  1
      transaction ID   1344085172823-12763576
```

Each contained message is described within the description of its related transaction, and the complete container contents are described as above in the Contents column of the packet table.



The container message above conveys an acknowledgement (format 124) and a send request (format 122). The latter message contains five data blocks that refer to different terminals identified by their IP addresses. Most of the parameters for a terminal are contained in an XML block that is assumed to be a connection-server message body, and the body is preceded by what may be a JMS header that starts with a timestamp and a message ID. NetData can't interpret all the binary codes and tokens in the header but it does recognise embedded 3-column tables of properties such as the *desktopcontrol* table in this example.

NetData extracts event and command types from messages for inclusion in its transaction signatures. It also extracts terminal IP addresses to display in the data and user-ID columns of the transaction table. This characterisation of VDM connection-server transactions makes it possible to track changes in the state of particular terminals. State-changing messages are readily found in the transaction-class tree, as in this extract of types of transactions that were initiated by the server (indicated by the word 'Reverse'):



9.5 IBM Java Message Service of a Service Integration Bus

NetData characterises transactions of IBM's Java Message Service (JMS) that uses the messaging engine of a Service Integration Bus (SIB) on the Java2 Platform, Standard Edition (J2SE) with WebSphere Application Server. All its messages begin with a two-byte sequence whose hex code is BEEFh. NetData's application tag is 'JMS-SIB'.

9.6 IBM WebSphere MQ

The WebSphere version 6 of the protocol is able to handle complete application transactions with a single connection, placing request messages in their queues with Put commands, and fetching response messages from their queues with a polling scheme using Get commands. Version 7 of WebSphere MQ can avoid polling by allowing messages to be sent from queues asynchronously – without waiting for a Get command – and handles concurrent commands in a single connection.

NetData accommodates quite different ways of working, with many connections involved in the handling of a single application transaction. Application request messages may be fetched from their queues with Get commands, and response messages placed in their queues with Put commands. It is quite common for a Get command to poll a queue while offering a short message-receive buffer of, say, only 512 bytes. If the buffer is too small for a complete message segment (typically up to 32 Kbytes), the client then issues another Get request for the specific message, advertising an adequate buffer size. NetData has seen many concurrent, non-specific Get requests fetch the same details of a new message, and then a similar set of connections all issue the same specific Get requests for the message. Only one such request is successful in fetching the new message. In one scheme the application used two sets of MQ connections, one set dedicated to Get requests and the other set dedicated to Put requests. In those circumstances NetData finds the first Get request whose reply refers to a particular request message; assembles only one transaction record for the application transaction; and updates that transaction record with relevant details from

all the associated non-specific Get requests, the specific Get requests, and the eventual Put requests that complete the transaction.

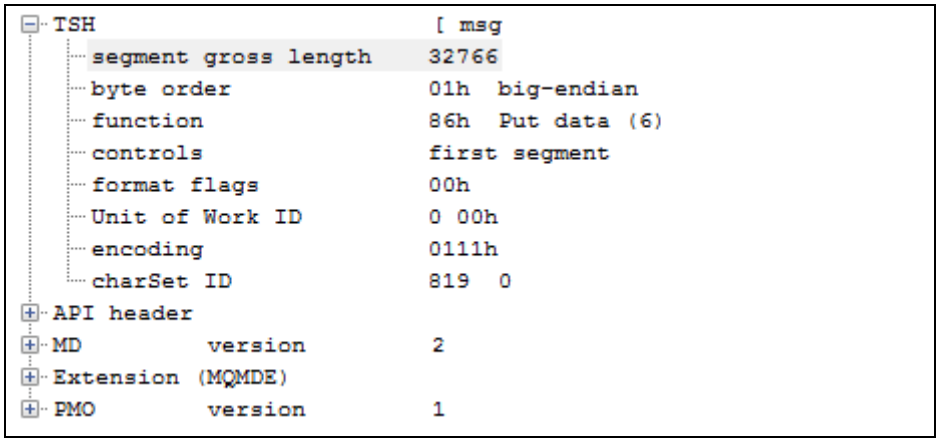
As it does for the older MQ protocol, NetData assigns a different application type for application transactions, to separate them in the transaction-class tree from supervisory transactions such as those that open and close access to specific queues. For the older, 'batching' protocol the application tag is MQvB/TCP, and for the WebSphere version the application tag is either MQvW/TCP or MQv7/TCP (for WebSphere Version 7).

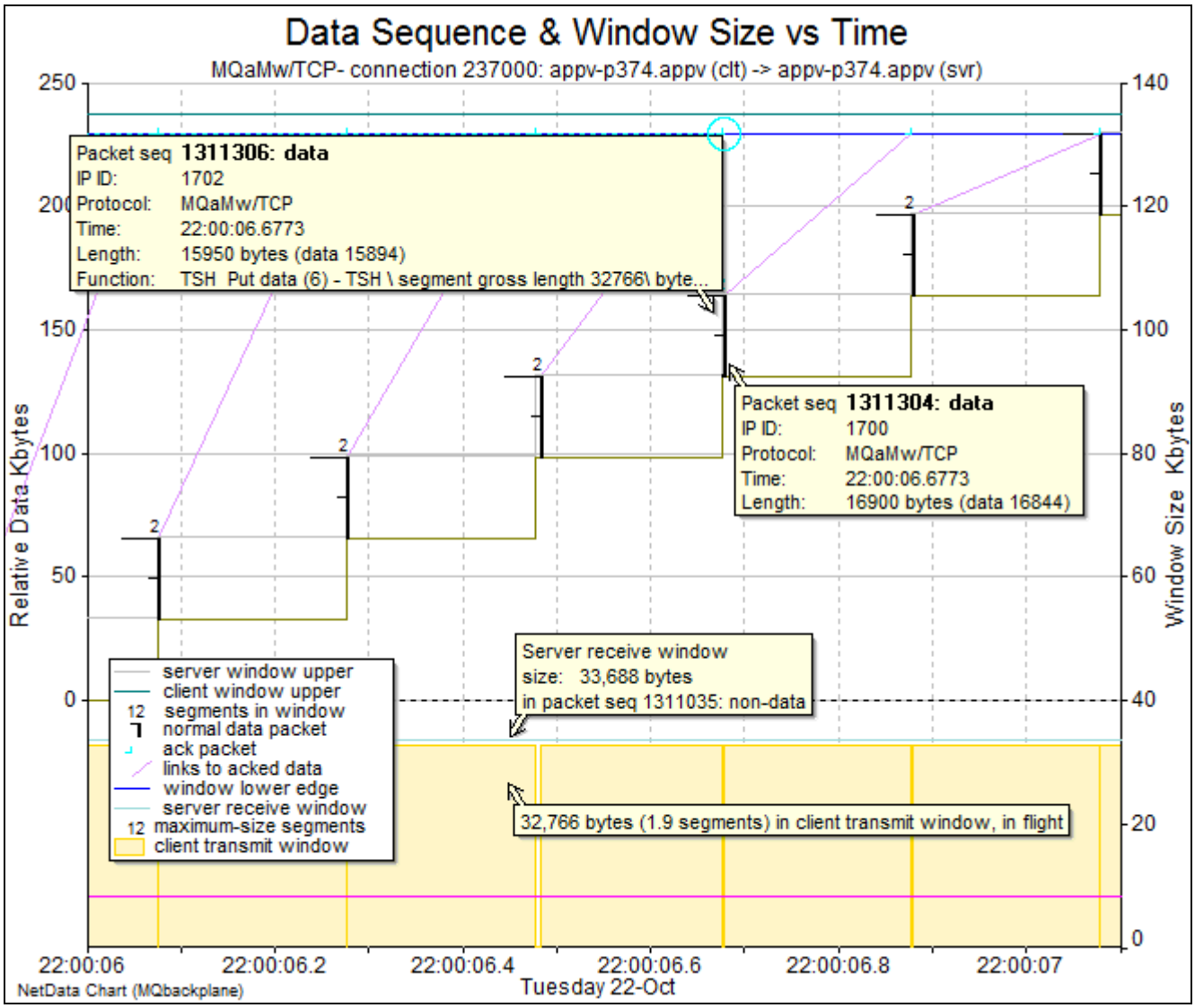
In all protocol versions NetData parses any XML text found in request and response messages, and will extract the values with specified XML tags for inclusion in transaction signature and key-data fields. By this means NetData is able to distinguish MQ transactions of different types and can separate failed transactions from successful transactions.

9.6.1 Slow Power System Backplane Transfers

In an IBM pSeries or Power System server with many logical partitions (LPARs) it was quite common to find a partition dedicated to handling all external communications and using the older MQvB protocol, while relaying MQ messages to and from the relevant application LPARs using the MQvW protocol, across the server's backplane either with specific LPAR IP addresses, or using the loop-back address. In these servers, however, standard TCP settings can reduce transfer rates within the machine to less than 2 Mbps. When using the loop-back address, or any single IP address for both client and server, default settings impose a jumbo segment size (with a typical MSS of 16,844), and a receive-window size that fits exactly two maximum-size segments. If the application buffer size is less than two maximum-size segments, and send-buffer space can't accommodate more than one application buffer, the sender must wait for an ack after sending each application buffer; and because an immediate ack is sent only when the equivalent of two maximum-size segments have been received, every ack is delayed. The TCP stack in AIX, like some other Unix systems, counts bytes rather than packets when deciding to send immediate acks.

In the system described below, MQ used a maximum transmission segment size of 32,766 bytes (as stated in the Transmission Segment Header (TSH) in the panel below), and that is also the largest number of bytes that were ever in flight for TCP (see window-size chart). Although every burst included three packets, every ack was delayed by 200 ms, and throughput was limited to 1.3 Mbps.





Small light-blue markers across the top of the chart indicate the occurrence of server acks, and purple lines link them to the client data packets that they acknowledge; they all acknowledged data that was received 200 ms earlier. Acks were delayed because application and TCP buffer sizes never allowed two maximum-size segments to be put in flight at any time. If send-buffer space can't be increased to at least twice the MQ segment size, or the segment size can't be reduced, the simplest work-around for this performance problem may be to configure TCP to acknowledge every data packet.

9.7 IBM MQ Series Telemetry Transport (MQTT)

NetData decodes transactions that use the MQ Series Telemetry Transport (MQTT), a protocol developed originally by IBM and Arcom jointly to communicate efficiently with simple process-control devices, particularly for SCADA (supervisory control and data acquisition) systems. The protocol is supported by IBM WebSphere MQ.

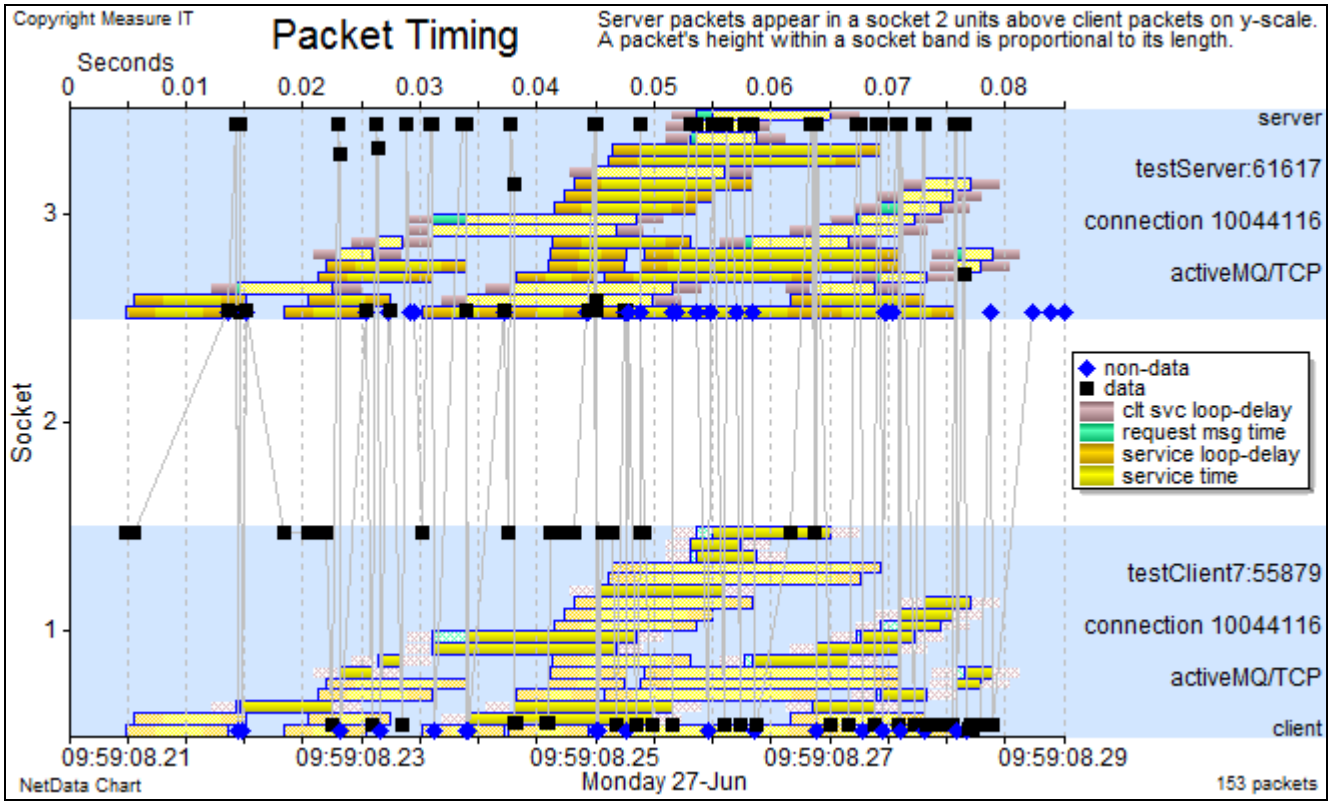
9.8 Apache ActiveMQ

Apache ActiveMQ, is an open-source messaging and Integration Patterns server that supports JMS (Java Message Service) and a message broker service. NetData decodes ActiveMQ traffic that uses OpenWire formats on TCP. It recognises the codes of 63 message types (called 'commands') and characterises common round-trips that can be initiated by either client or server and may run concurrently. Some messages act as responses to commands, and other messages convey information without needing a response.

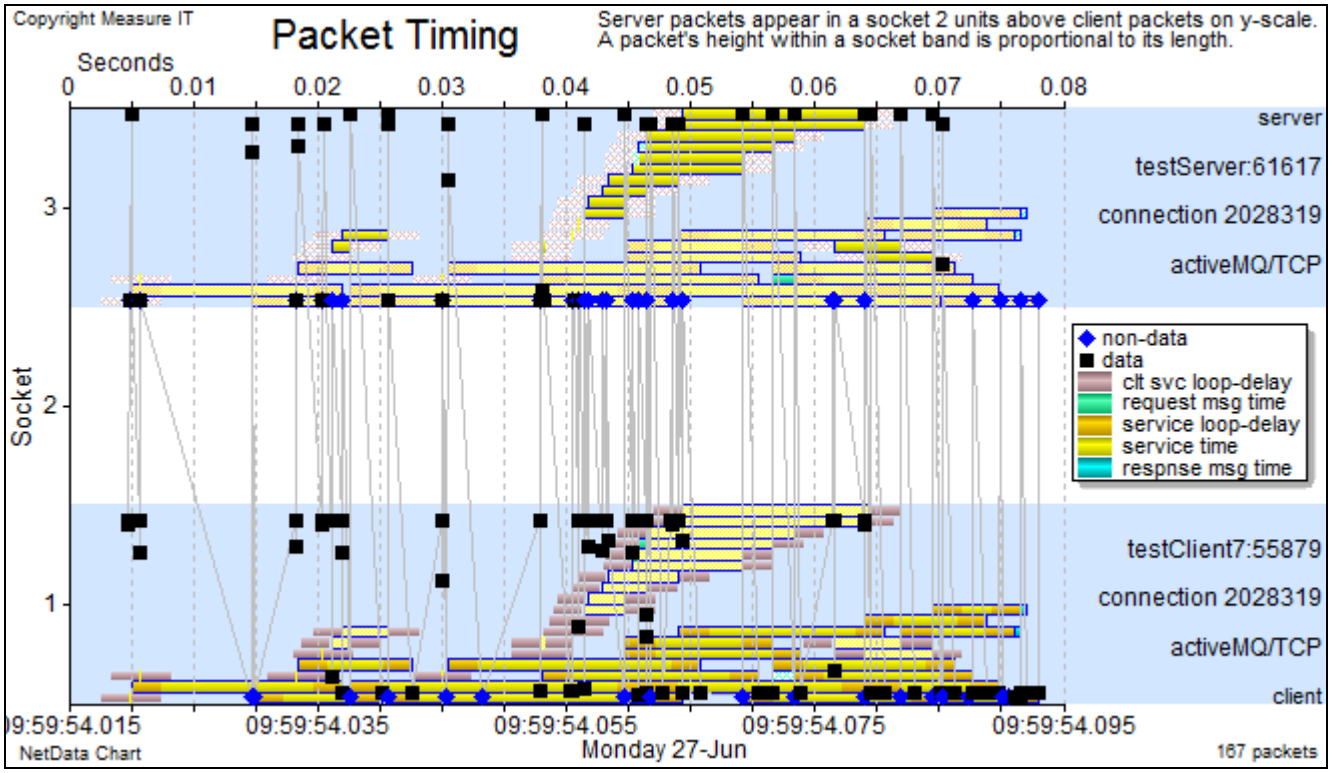
NetData displays the structure of any XML or Java objects that may be embedded in messages.

9.8.1 Charting Propagation Displays

The chart below displays the transfer of 21 messages to a messasge broker and the same 21 messages subsequently distributed from the broker. The streams of large and small packets travelling in opposite directions reveal up to 18 concurrent round-trips over a period of just 75 ms.



The loop-delay between client and server was 5 ms and, for normal round-trips initiated by the client, is painted as orange bars that subtract from the yellow service time, but for *reverse* round–trips initiated by the server the delay is painted as brown bars that add to the *client* service time. By taking into account the time for a packet to propagate from one end of the network to the other, the brown-bar extensions provide an estimate of a transaction's true time span, and the orange-bar contractions provide an estimate of a transaction's true service time.



This chart displays the same burst of messages submitted to the message broker, captured at the server rather than the client. Much of the server data was recorded in packets of 2774 bytes, twice the MSS, before segmentation by TCP.

For this capture the brown extension bars provide estimates of true transaction time spans for round-trips initiated by the client.

9.9 Microsoft Message Queuing

NetData's decoder for the Microsoft Message Queuing (MSMQ) protocol decodes the MQ headers in all packets and characterises the delivery of transactional messages as pseudo server transactions. In NetData terms the time to transmit the message is regarded as request-message transfer time; the 'response' is assumed to start when the sender receives an ack on any level, and ends when the sender has received the acks expected on all levels.

NetData transactions characterised in this way can indicate on a performance chart the times when significant messages were sent to a queue, and large server response times may indicate stress in the receiving queue manager. Large overall response times, however, may not be significant because end-to-end order acks may be delayed many seconds for efficiency – with one packet acknowledging many messages – somewhat like a TCP delayed ack. Unlike the IBM MQ decoder, the MSMQ decoder doesn't yet try to match application request messages with their response messages.

A description of MSMQ messages and protocol is provided by two Microsoft open specifications:

[MS-MQMQ]: *Message Queuing (MSMQ): Data Structures*, v20150630; and

[MS-MQQB]: *Message Queuing (MSMQ): Message Queuing Binary Protocol*, v20150630

The protocol involves many different categories of messages. The first split is between *Internal* messages, for connection establishment and session acknowledgements, and *User* messages. User messages may have two optional and significant attributes, one making them *recoverable* and subject to an additional low-level session acknowledgement system, and the other making recoverable messages *transactional* and tracked end-to-end (to the end destination queue) with an additional sequence number.

In MQ terminology an MQ 'packet' is a NetData message that contains one or more MQ headers and an optional MQ message body; it may involve a large number of network packets.

At the beginning of a TCP connection there are two round-trips with Internal messages to establish the MQ connection and exchange connection parameters such as delayed-acknowledgement timeouts. NetData characterises these trips as server transactions.

User messages are given sequential numbers starting with one at the beginning of a session (a new TCP connection). Session headers conveyed in SessionAck Internal messages, and sometimes appended to User messages, convey four sequence numbers: the sequence numbers of the last user and recoverable messages acknowledged as received in the *opposite* direction; and the sequence numbers of the last user and recoverable messages sent in the *same* direction. User messages don't convey their own sequence number, but NetData displays their assumed numbers in the Context column of the packet table, and in the same column also displays the four sequence numbers conveyed in session headers. In transaction descriptions the assumed session sequence number is displayed at the end of the user header.

Transactional messages belong also to a second sequence of transmitted messages. Within one connection there can be many concurrent transactional-message sequences, and the transaction header of a transactional message conveys a TxSequence ID and a sequence number within the identified sequence.

Transactional messages receive acks on two or three levels: a low-level *SessionAck*, a higher-level, end-to-end *QM Ordering Ack* (or *OrderAck*), and a high-level *FinalAck* such as 'ack receive'. Session acks are issued for all user messages and one ack acknowledges all messages with lower or equal session sequence numbers. An OrderAck acknowledges all transactional messages with lower or equal sequence numbers in the identified transaction sequence. A FinalAck, however,

acknowledges only the message with the identified message number from the identified source-queue manager.

A FinalAck is required only if flags in the user header specify either positive (JP) or negative (JN) journaling. The most common FinalAck is *ack receive* which indicates that the receiving user-level application has retrieved the message from the destination queue, but a FinalAck can also provide a negative acknowledgement of eleven different types. NetData records every occurrence of a negative ack as a network event in the application-error category.

An *administration acknowledgement* is a fourth type of ack that sends a positive or negative ack message to an administration queue designated in the user header of the original message, only if requested by a flag in the message-properties header. NetData doesn't associate administration acks with delivery transactions.

The KeyData column of the packet table displays both a transaction-sequence ID and the message's index in that sequence:

Time Of Day	Source	Destination	Len	Data	Function	Context	KeyData
10:26:28.4543	57. 30. 40...	57. 30. 41...	330	272	recoverable user mes...	45337540; usrSeq521; recSeq172	
10:26:28.6627	57. 30. 41...	57. 30. 40...	64				
10:26:28.9593	57. 30. 41...	57. 30. 40...	94	36	SessionAck	ack521; 172; snt360; 360	
10:26:29.161	57. 30. 40...	57. 30. 41...	64				
10:27:17.2172	57. 30. 41...	57. 30. 40...	970	912	EventNotification; rec...	18026385; usrSeq361; recSeq361	122-1435292734s185; Rts.
10:27:17.4118	57. 30. 40...	57. 30. 41...	64				
10:27:17.7199	57. 30. 40...	57. 30. 41...	94	36	SessionAck-7	ack361; 0; snt521; 172	
10:27:17.9268	57. 30. 41...	57. 30. 40...	64				
10:27:17.9269	57. 30. 40...	57. 30. 41...	394	336	QM Ordering Ack exp...	45337764; usrSeq522	122-1435292734s184
10:27:18.1431	57. 30. 41...	57. 30. 40...	64				
10:27:18.3822	57. 30. 41...	57. 30. 40...	94	36	SessionAck	ack522; 0; snt361; 361	
10:27:18.5819	57. 30. 40...	57. 30. 41...	64				
10:27:20.1631	57. 30. 40...	57. 30. 41...	94	36	SessionAck-5	ack361; 361; snt522; 172	

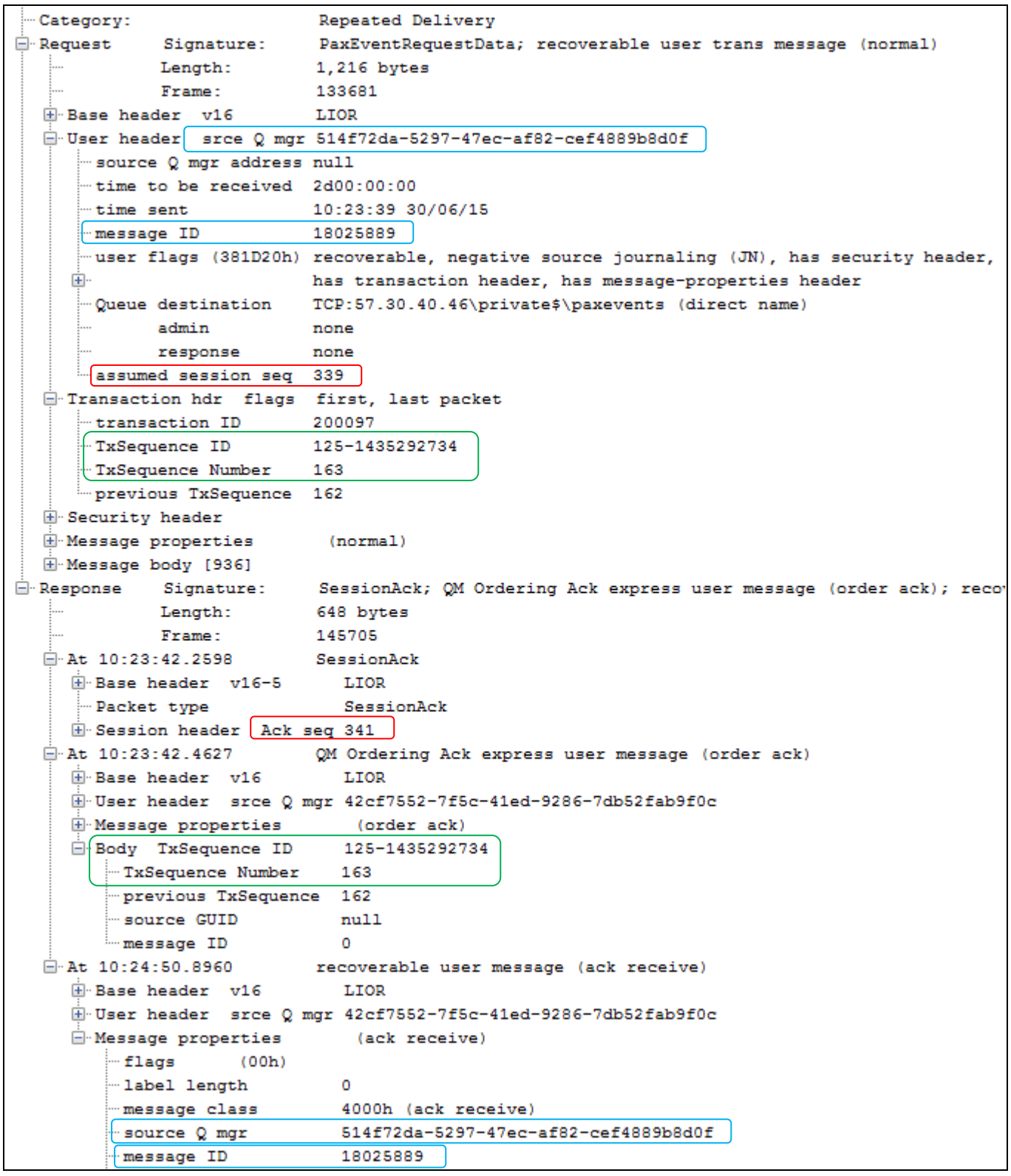
In this sequence of network packets there are two MQ SequenceAck packets in each direction; two user messages from the server with sequence numbers 521 and 522; and one transactional user message from the client with session sequence number 361, transaction sequence ID 122-1435292734, and sequence number 185 in that sequence. NetData recognised that the body of this transactional message was XML whose first tag described the message as an EventNotification. NetData can be asked to find specific named fields to extract user IDs and key data to further characterise the delivered messages.

The panel below describes a transactional message with an XML body. NetData's transaction category for MQ messages is always 'Delivery', and it is prefixed with 'Repeated' if the same message had been transmitted earlier. The request signature always contains 'recoverable user trans message (normal)'and in this case has been prefixed with the name of the main element in the XML message body. A description in parenthesis, such as 'normal' or 'order ack', refers to the MQ message *class*.

The transaction's response signature describes two or three MQ ack messages, including its session ack; order ack (identifying its MQ transaction sequence and its number within that sequence), and an optional admin ack. User messages may have a message label, and the label for an order-ack message must always be 'QM Ordering Ack'. Because the labels of *non-transactional* MQ messages help to characterise a NetData transaction type they are included in transaction signatures, but the labels of *transactional* MQ messages are often variable and they are included in transaction key-data fields.

The following NetData transaction description (category and signatures) is typical of an MQ message with XML:

> Repeated Delivery: PaxEventRequestData; recoverable user trans message (normal) --SessionAck; QM Ordering Ack express user message (order ack); recoverable user message (ack receive)



The Request part of this description of an MQ message delivery describes four headers: Base, User, Transaction and Security. The message-properties block includes the body of the message itself (with 936 bytes).

The Response part of this description begins with the description of a SessionAck message acknowledging messages up to session sequence 341; it includes an OrderAck message whose body matches information in the request's transaction header; and ends with a FinalAck ack-receive message that identifies the original source-queue manager and message ID.

9.10 Computer Associates Message Queuing (CAM)

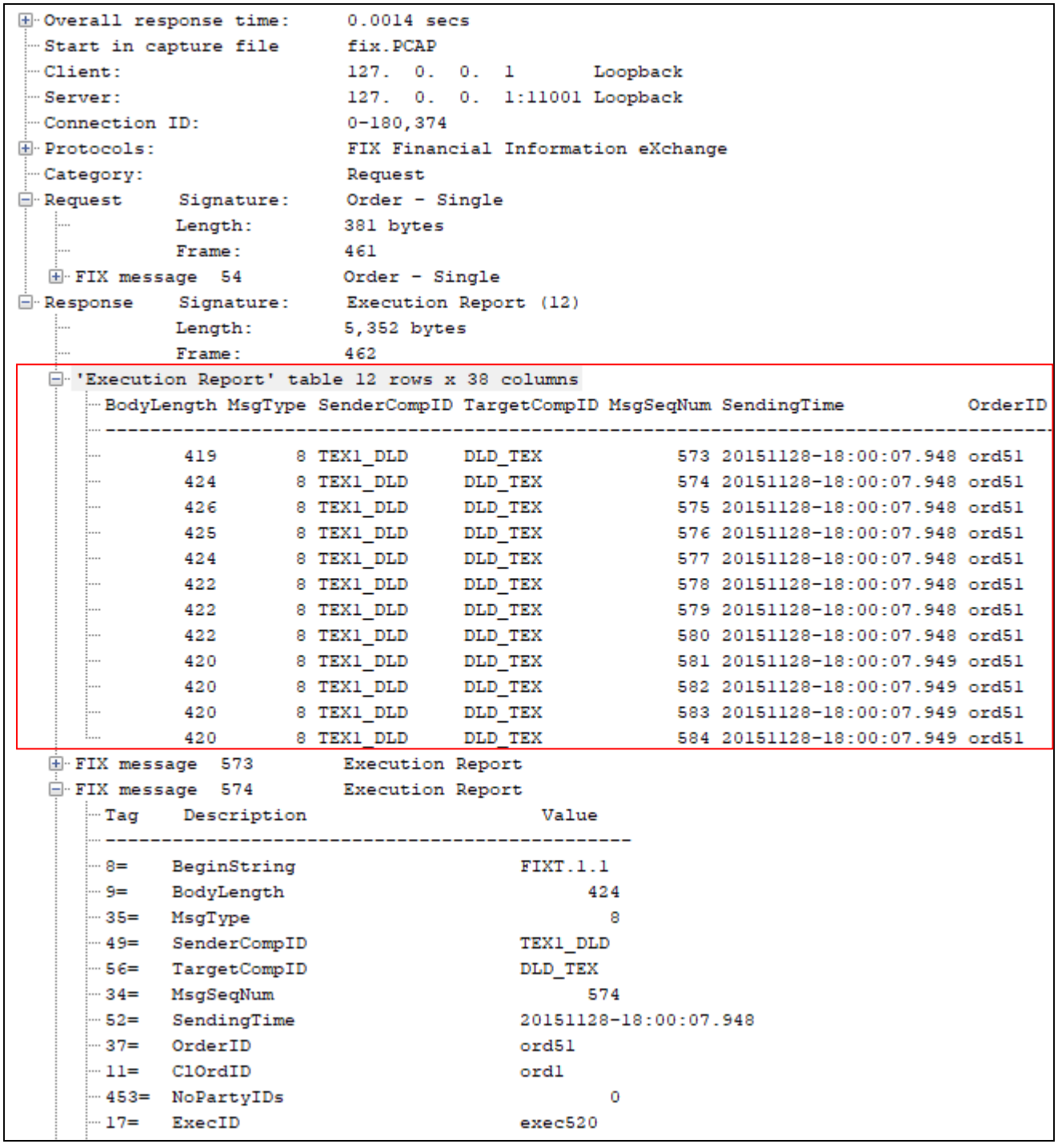
Computer Associates' Message Queuing (CAM) module forms the basis for various Unicenter products including CA File Transfer (CAFT), Application Performance Monitor, and Advantage Data Transport. NetData decodes CAM traffic which normally uses TCP port 4105.

10 Trading Systems

10.1 Financial Information eXchange (FIX)

NetData now decodes transactions that use the Financial Information eXchange (FIX) protocol and the classic FIX Tag=Value encoding. It recognises 1504 field tag values and 105 message types.

For round-trips that convey multiple FIX messages NetData describes all the fields in each message and constructs tables with a row for each message that can be browsed, sorted, searched, filtered and exported like any other NetData table.



```
Overall response time:      0.0014 secs
Start in capture file       fix.PCAP
Client:                     127.  0.  0.  1       Loopback
Server:                     127.  0.  0.  1:11001 Loopback
Connection ID:              0-180,374
Protocols:                  FIX Financial Information eXchange
Category:                   Request
Request      Signature:     Order - Single
             Length:        381 bytes
             Frame:         461
  FIX message  54           Order - Single
Response     Signature:     Execution Report (12)
             Length:        5,352 bytes
             Frame:         462
  'Execution Report' table 12 rows x 38 columns
    BodyLength MsgType SenderCompID TargetCompID MsgSeqNum SendingTime           OrderID
    ----------------------------------------------------------------------------------------
           419       8 TEX1_DLD     DLD_TEX            573 20151128-18:00:07.948 ord51
           424       8 TEX1_DLD     DLD_TEX            574 20151128-18:00:07.948 ord51
           426       8 TEX1_DLD     DLD_TEX            575 20151128-18:00:07.948 ord51
           425       8 TEX1_DLD     DLD_TEX            576 20151128-18:00:07.948 ord51
           424       8 TEX1_DLD     DLD_TEX            577 20151128-18:00:07.948 ord51
           422       8 TEX1_DLD     DLD_TEX            578 20151128-18:00:07.948 ord51
           422       8 TEX1_DLD     DLD_TEX            579 20151128-18:00:07.948 ord51
           422       8 TEX1_DLD     DLD_TEX            580 20151128-18:00:07.948 ord51
           420       8 TEX1_DLD     DLD_TEX            581 20151128-18:00:07.949 ord51
           420       8 TEX1_DLD     DLD_TEX            582 20151128-18:00:07.949 ord51
           420       8 TEX1_DLD     DLD_TEX            583 20151128-18:00:07.949 ord51
           420       8 TEX1_DLD     DLD_TEX            584 20151128-18:00:07.949 ord51
  FIX message  573          Execution Report
  FIX message  574          Execution Report
    Tag    Description                    Value
    ------------------------------------------------
    8=     BeginString                    FIXT.1.1
    9=     BodyLength                           424
    35=    MsgType                                8
    49=    SenderCompID                   TEX1_DLD
    56=    TargetCompID                   DLD_TEX
    34=    MsgSeqNum                            574
    52=    SendingTime                    20151128-18:00:07.948
    37=    OrderID                        ord51
    11=    ClOrdID                        ord1
    453=   NoPartyIDs                             0
    17=    ExecID                         exec520
```

10.1.1 Session Protocol and Message Sequence Number Tracking

The FIX Trading Community (https://www.fixtrading.org/) has specified a session protocol separate from the application protocol. A session can survive breaks in connections and resume its two series of FIX sequence numbers (incoming and outgoing) which ensure that FIX application messages are processed in the correct order, without gaps. Administrative messages can test for sequence gaps and request that specific application messages be resent.

NetData forms a session name from SenderCompID and TargetCompID field values and assigns it to the UserID field of all the session's connections, and all its transactions. Also recorded with each connection are its initial and final (next expected) message sequence numbers for the client and server streams. The relevant fields can be displayed in the connection table:

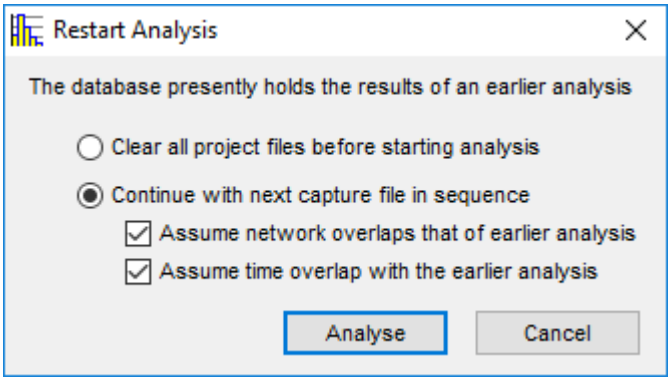
ConnID	UserID	Type	Server (Call...	sPort	Init Clt Seq	Next Clt Seq	Init Svr Seq	Next Svr S...
180374	DLD_TEX - TEX1_DLD	FIX/TCP	Loopback	11001	1	55	1	586

Message sequence numbers appear in the Context column of the packet table.

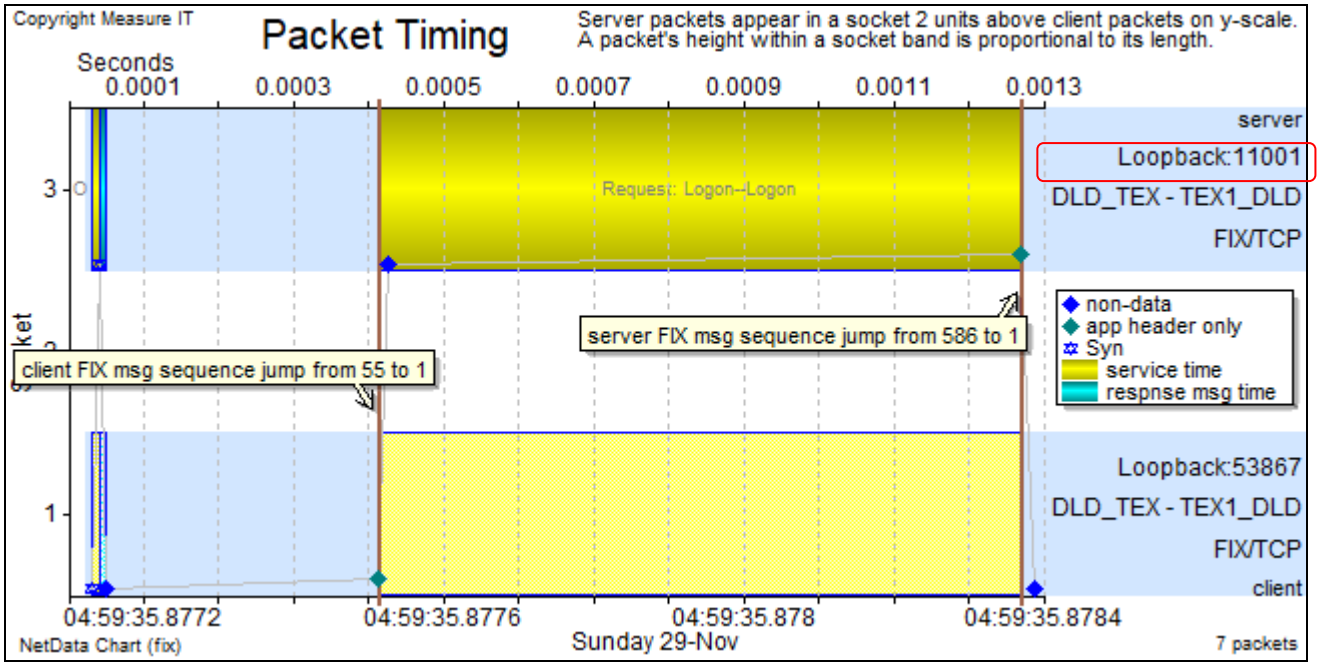
When NetData encounters a new FIX connection it looks for the most recent connection of the same session and uses its final sequence numbers to initialise the next-expected sequence numbers of the new connection.

NetData tracks all sequence numbers and if there is any anomaly records a network event in the project database. If these records are loaded into the charting module they produce brown vertical stripes on both the performance and timing charts to mark the times at which the anomalies were detected.

When FIX sessions are transported by TCP, session gaps should be rare, but NetData's identification of sequence anomalies can be illustrated by forcing NetData to analyse a capture file a second time without first clearing the database:



If connection records are also loaded with transaction and packet records, the descriptive column on the timing chart displays the session name of each connection, as in the following chart. This chart was produced by analysing the same capture file twice, with the result that the connection's Logon messages, from both the client and server, indicated a sequence-number jump from the final number back to the initial number of one.



Session names appear in place of connection IDs, but the connection IDs can be displayed by clicking the timing chart's Addresses button that toggles between addresses and names.

All packets with a FIX administrative message are marked on the timing chart with a dark-green diamond and labelled as 'app-header only'.

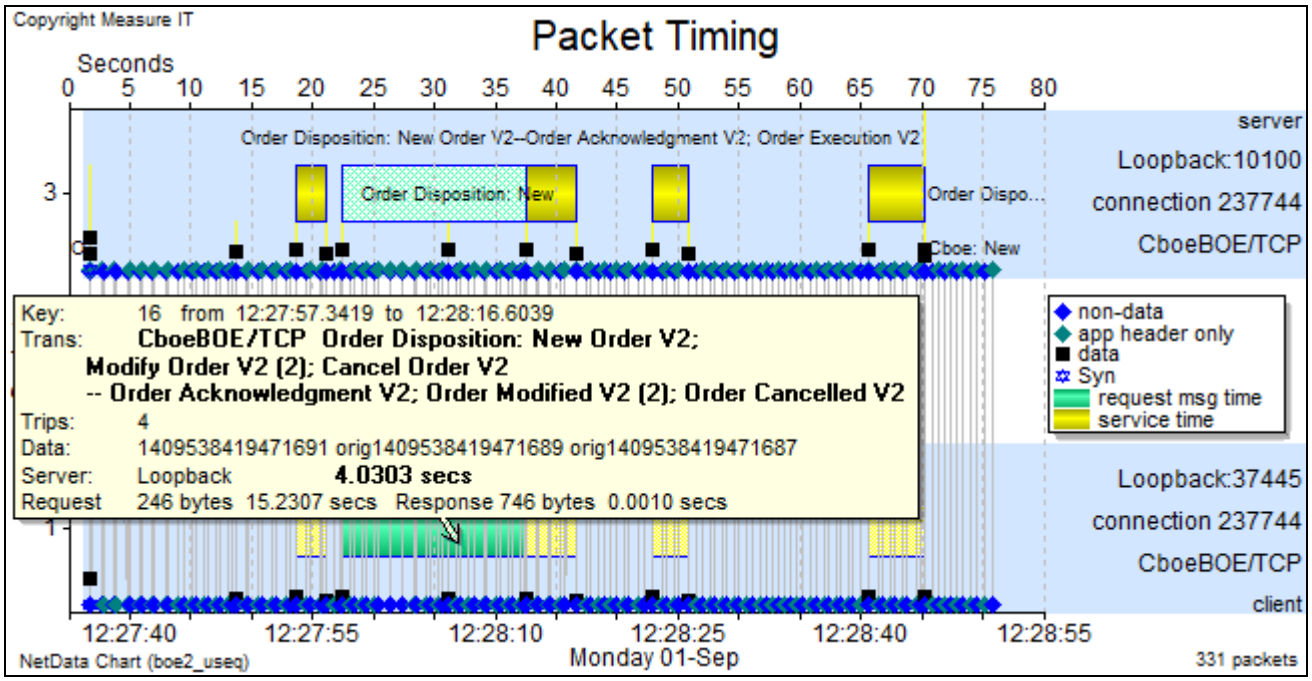
10.2 BATS/Cboe (Chicago Board Options Exchange) Traffic

BATS Global Markets was an innovative global financial-market technology company headquartered in Kansas City. The BATS platform was launched in 2006 and, operating as BATS Exchange Inc, became a fast growing, top-tier equity market in the United States. BATS served the European market through its London-based, FSA-authorised subsidiary, BATS Europe, which operated a Multilateral Trading Facility for European securities.

BATS Options refers to both options markets owned by BATS Global Markets: BZX Options (formerly BATS Options) and EDGX Options. BATS Options was launched as an options exchange in 2010, and EDGX Options was launched in 2015. BATS Global Markets, along with its subsidiary BATS Options, was acquired by the Chicago Board Options Exchange (Cboe) in 2017.

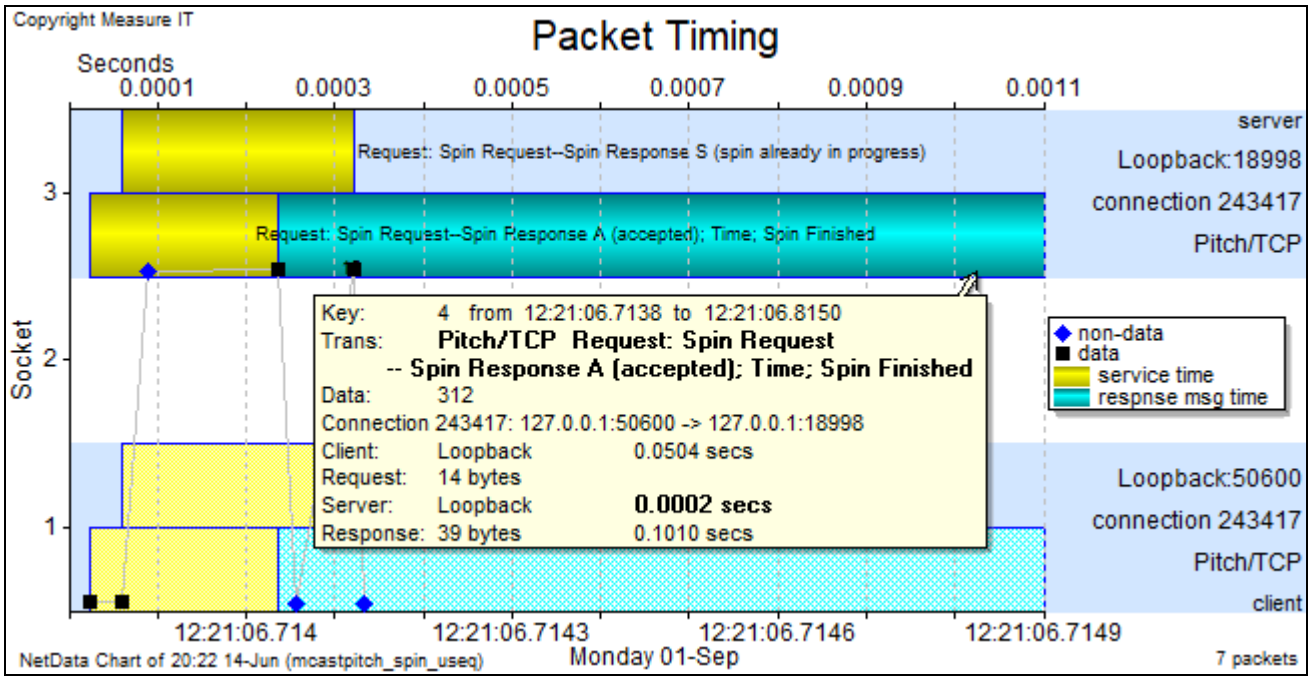
NetData fully decodes the traffic of two Cboe protocols: BOE (Binary Order Entry); and PITCH for receiving real-time full depth-of-book quotations and execution information. The decoder handles different variants of the protocols used by exchanges in Europe and the US.

For BOE traffic NetData characterises two classes of transactions: individual round-trips embracing request-response pairs such as the messages for a New Order and its corresponding Order Acknowledgement; and the sets of all messages referring to the same client order that start with New Order and end with an Execution or Cancellation message. Those in the first class are regarded as *technical* transactions because they characterise the performance of the IT system which uses networks carefully-designed for low latency. The second class might be regarded as *market* transactions and are classified by NetData as User transactions.



Individual round-trips are so quick that their transaction bars are difficult to see on a chart of this scale, but the user (market) transactions are seen more clearly. The long bar in the middle of this chart describes the life of a new order that was modified on two occasions and eventually cancelled.

PITCH traffic uses two types of connections: UDP connections for streams of market-information messages; and TCP connections that allow market participants to request a spin of a unit's order book or a data gap to be filled. NetData measures and characterises all TCP transactions and the chart below depicts two Spin requests in quick succession. The second was not accepted ('spin already in progress') but the first was accepted and the bar continued through to the Spin Finished message.



For each PITCH UDP connection NetData tracks the different sequences of messages for each unit number and flags all sequence gaps and packets out of order (i.e. 'overtaken').

10.3 Australian Energy Market Transaction Decoding (2012)

NetData detects, characterises and plots the B2B transactions of two different protocols used by the Market Settlement and Transfer Solution (MSATS) of the Australian Energy Market. It detects XML in MSATS messages and is able to extract the values of nominated fields for inclusion in a transaction's key-data, signature and user-ID fields. The XML messages convey such energy marketing information as electricity meter data, work requests and acknowledgements.

Traffic of the two protocols is tagged as either MSATSB2B or B2Bsupv. The first protocol uses various request-response pairs to Get, Put and Mark XML messages in queues. The second protocol exchanges short messages at 5-minute intervals, perhaps for keep-alive purposes, but all application messages are sent without responses. Those messages appear to use a proprietary text format to convey statistics and version information concerning traffic with message queues.

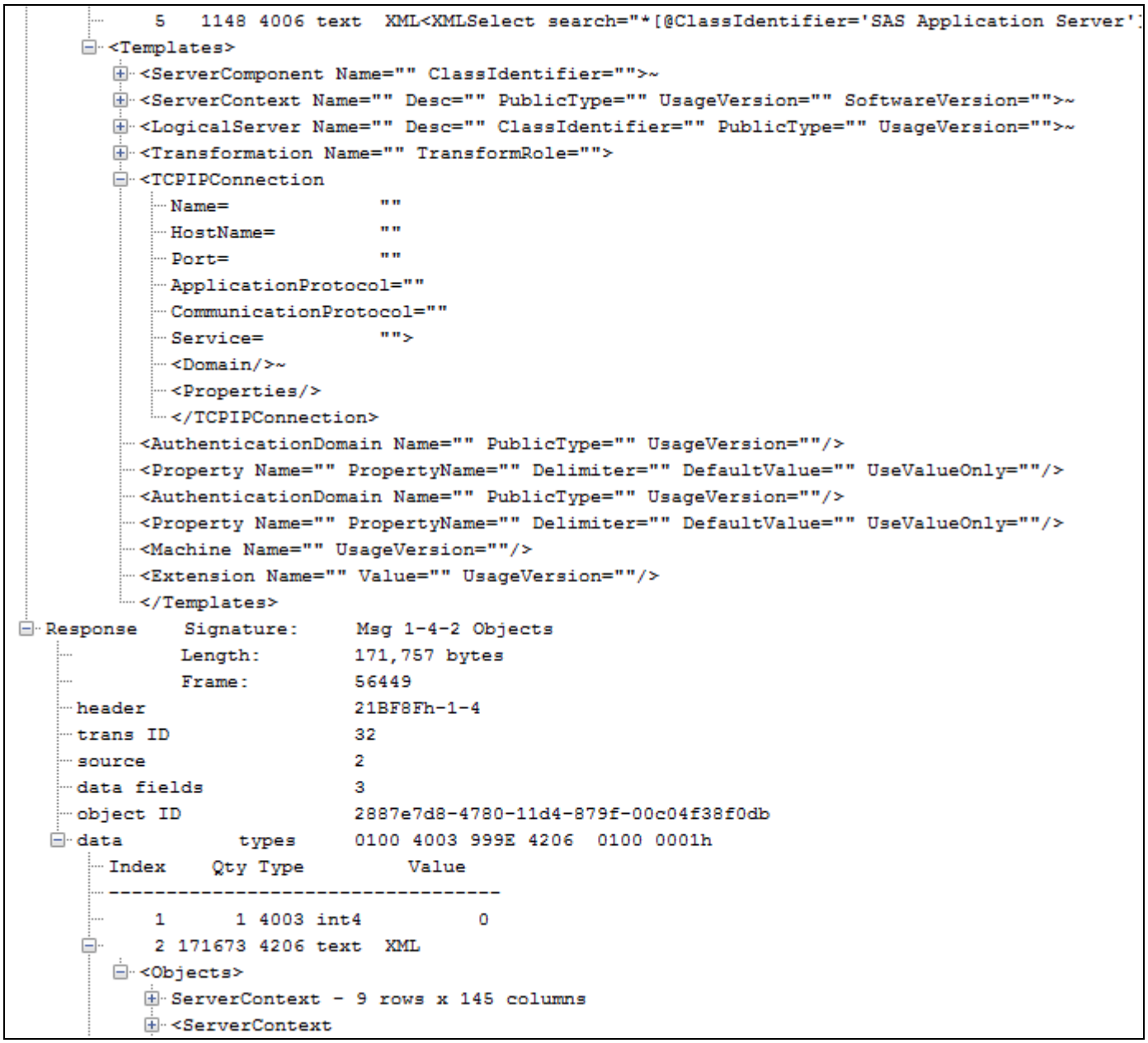
11 System Management

11.1 SAS Intelligence Platform

The decoder for the various types of SAS Intelligence Platform traffic – transactions with the workspace server, metadata server and object spawner – handles little-endian as well as big-endian message formats, characterises transactions even when they run concurrently in the same connection, and displays object UUIDs in the key-data columns of the packet and transaction tables.

The main body of a transaction message contains data fields in a variety of text and numeric binary formats. NetData displays the text strings on separate rows, and if a text contains an XML or SQL statement it is displayed in its normal structured form. Transaction signatures are extended with generalised SQL statements, XML tag names, and the complete contents of XMLSelect elements that specify a SAS database search. An XMLSelect element is followed by a nested element named Templates that specifies a structure for the returned XML document.

The following SAS XML query returned a table with 9 rows in 172 Kbytes, and by expanding the tree NetData would display the sub-tables and sub-sub-tables that might be found in any rows.



In the following two panels NetData describes a server notification containing a SAS log that was produced during the execution of a SAS procedure. It is the equivalent of a printer listing broken into separate pages with page headings such as the text of 133 characters in data-field 9.

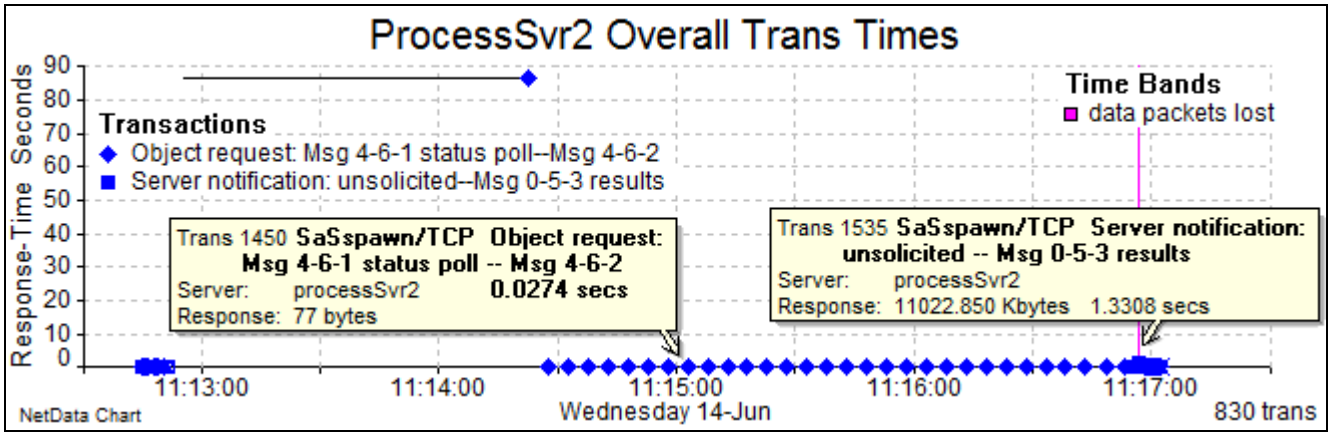
The message header in this case specifies a transaction ID of zero and a source of 3 which together denote an unsolicited server message. The header continues with the number of data-fields (100398), the UUID of the server's object sending the message, and a list of 4-byte specifiers combining the length (quantity) and data-type for each data-field.

```
Protocols:                  Object Spawner for the SAS Intelligence Platform
Key data:                   66660bbb-8b5a-11d0-9f3e-00a024bb830c/ 29/ 31/ 32/ 34/ 36/ 37
Category:                   Server notification
Request     Signature:      unsolicited
Response    Signature:      Msg 0-5-3
            Length:         11,022,850 bytes
            Frame:          75079
  header                    0-Ch-5
  trans ID                  0
  source                    3
  data fields               100398
  object ID                 66660bbb-8b5a-11d0-9f3e-00a024bb830c
  data            types     0100 0002 0100 4001  2688 4103 0100 0002  0100 4001 2688 4103  0100 00
    Index    Qty Type          Value
    ---------------------------------
        1      1 0002 uint2          1
        2      1 4001 int4      100390
        3 100390 4103 int4   1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        4      1 0002 uint2          1
        5      1 4001 int4      100390
        6 100390 4103 int4   3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0
        7      1 0002 uint2          1
        8      1 4001 int4      100390
        9    133 4006 text   1                                                          The SAS
       10      1 4006 text
       11     36 4006 text   1           %_eg_hidenotesandsource;
       12     36 4006 text   5           %_eg_hidenotesandsource;
       13     12 4006 text          29
       14    112 4006 text   30          /* LIBNAME ANL_LIB TERADATA  SERVER=tdatprd  SCHEMA=NDIA
```

This SAS message with mostly text takes the form of a table with 100,390 rows and 3 columns, sent column by column. The first two columns contain 4-byte integers, and the third column contains text. All the integer values are sent in a single data-field (indexes 3 and 6) for each of the first two columns. The quantity for each of those data-fields is specified by three bytes (e.g. 418826h) preceding a one-byte data-type indicator (e.g. 03 for a 4-byte integer), and each column is preceded by a 2-byte integer (indexes 1,4,7) then a 4-byte integer (indexes 2, 5, 8) that indicates the number of rows (100390 in this case).

Most of the text rows log the time at which each of 96,998 data rows were fetched from the Teradata database. The procedure summary at the end of the notification message (shown in the panel below) indicates that the procedure took 90.1 seconds to execute the query and 152.8 seconds to fetch all the data rows, making a total execution time of 243 seconds.

```
100375    99 4006 text  TERADATA:  The fetch time in seconds for  rows is   152.834496 96992
100376    99 4006 text  TERADATA:  The fetch time in seconds for  rows is   152.835312 96993
100377    99 4006 text  TERADATA:  The fetch time in seconds for  rows is   152.835586 96994
100378    99 4006 text  TERADATA:  The fetch time in seconds for  rows is   152.835890 96995
100379    99 4006 text  TERADATA:  The fetch time in seconds for  rows is   152.836027 96996
100380    99 4006 text  TERADATA:  The fetch time in seconds for  rows is   152.836155 96997
100381    86 4006 text  NOTE: Compressing data set WORK.COLUMNS_PASSTHROUGH decreased size b
100382    70 4006 text        Compressed is 119 pages; un-compressed would require 134 pages
100383    77 4006 text  NOTE: Table WORK.COLUMNS_PASSTHROUGH created, with 96998 rows and 3
100384     1 4006 text
100385    38 4006 text    96998 1497403015 no_name 0 SQL (2)
100386    73 4006 text  Summary Statistics for TERADATA are: 96999 1497403015 no_name 0 SQL
100387   101 4006 text  Total row fetch seconds were:                    7722885.001708 970
100388    97 4006 text  Total SQL execution seconds were:                  90.128475 97001 1
100389   133 4006 text  1674                                                        The SAS S
100390     1 4006 text
100391    97 4006 text  Total SQL prepare seconds were:                    0.833267 97002 1
100392   103 4006 text  Total seconds used by the TERADATA ACCESS engine were   243.945493 S
100393    38 4006 text    97004 1497403015 no_name 0 SQL (2)
100394    17 4006 text  54          quit;
100395    47 4006 text  NOTE: PROCEDURE SQL used (Total process time):
100396    34 4006 text        real time           4:05.03
100397    40 4006 text        cpu time            30.98 seconds
100398     7 4006 text
```



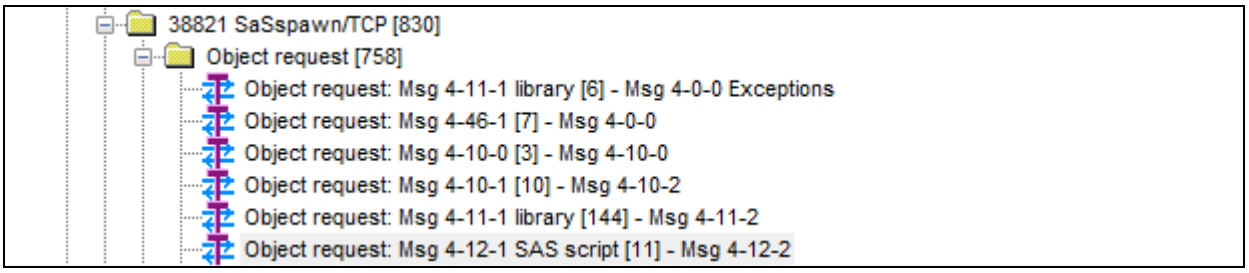
This performance chart of SAS transaction response times illustrates the time spent in each phase of the procedure's operation. Object requests with a message type of 4-6 (blue diamond markers) polled the server at 5-second intervals until the server sent the 11-Mbyte notification with the SAS log indicating that the procedure had finished. The first poll received a response after 86 seconds, delayed presumably until the query had finished. This observation suggests that a SAS application thread is single-threaded and can't respond to a user request while it is waiting for a response from the database.

Polls were answered quickly during the second phase when the resulting data rows were fetched from the database. The fact that the average time for each row was 2 ms, and the time for each row was logged, suggests that the data rows were fetched only one at a time, and this inefficiency accounted for most of the overall procedure time. Analysis of the database traffic is likely to confirm such a diagnosis. However, in the absence of a database traffic capture, much can be learnt from the information in the SAS log, particularly if – as in this case – the log records the relative times at which the SAS server fetched rows from the database.

Database-call timestamps are recorded in the SAS log if the SAS script for the statistical analysis job includes the 'sa' parameter in the tracing option as highlighted in the following script extract:

```
   object ID               c32a4d4b-2b4b-410d-9669-97f241d3435a
   data          types     BA1D 4006h
      Index    Qty   Type       Value
      ---------------------------------
          1   7610 4006 text
      %_eg_hidenotesandsource;
      ;*';*";*/;quit;run;
      .........
      GOPTIONS ACCESSIBLE;
      %_eg_restorenotesandsource;
      /* LIBNAME HARP_LIB TERADATA  SERVER=TeradataPrd  SCHEMA=HARP_PRD_;
      LIBNAME HARPOSTG BASE ... ;
      libname HARPRECN ... ;
      */
      OPTIONS SASTRACE=',,,sa' SASTRACELOC=saslog;
      PROC SQL;CONNECT TO TERADATA(authdomain="TeraAuthHARProd"
SERVER="HARPPRD");
             Create table work.columns_passthrough(compress=yes) as select *
from
                                                          connection to
TERADATA
             (
             select
                    databasename,
                    tablename,
                    columnname
              from dbc.columns
              where databasename like 'HARP%';
             )
      ;
      quit;
```

As indicated at the head of the above panel, NetData finds SAS script in a single text data-field in a request message sent from a workstation to a SAS server. The relevant transactions can be found in the transaction tree because NetData includes the words 'SAS script' in their request signatures:

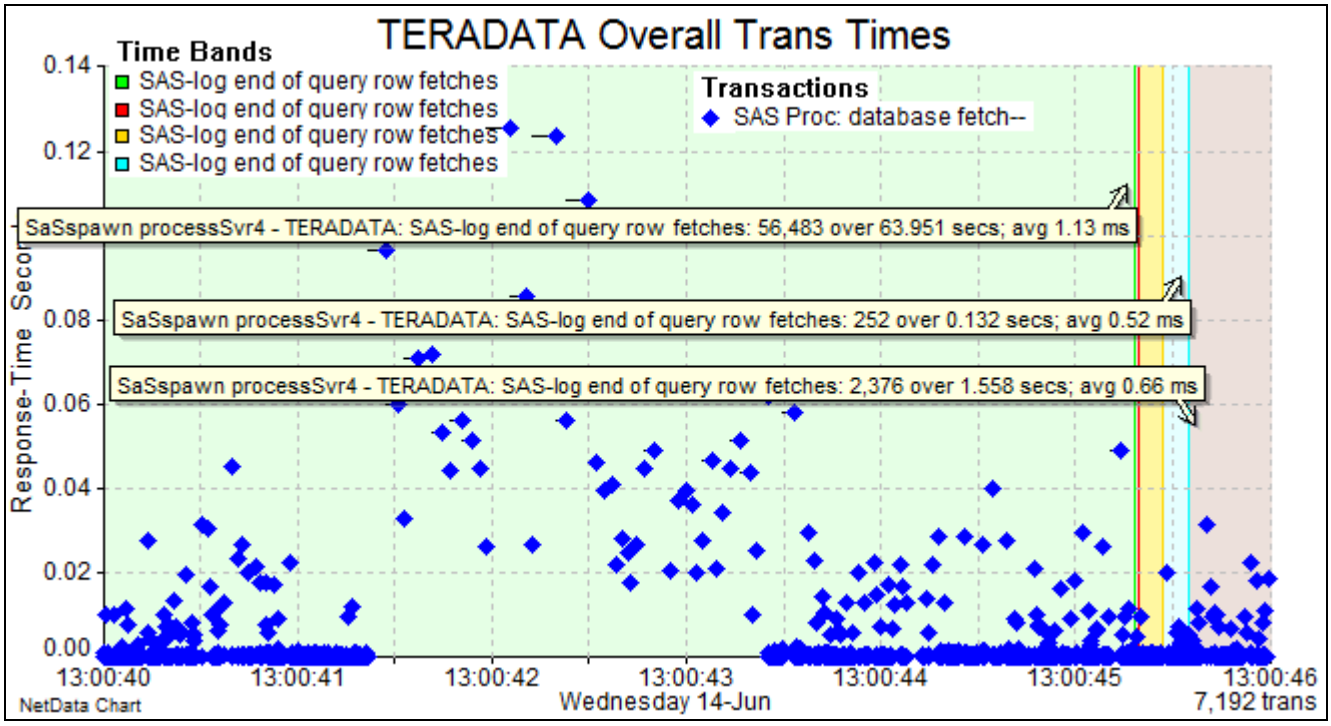


The word 'Exceptions' in a response signature (as above) is the tag name of an XML element that explains why a request failed.

If NetData finds database-call timestamps in a SAS log, in a server notification, it records those database calls in its database as pseudo transactions, with response times derived from the intervals between successive calls. The client recorded with these transactions is the SAS server making the calls; the server assigned to these transactions is a notional server with address 99.99.99.99, port 64999, and a name such as 'TERADATA' that is read from the SAS log.

A chart of the psudo database calls exposes any sign of stress or congestion that delays the SAS Procedure. The following chart plots some of the round-trip times for several queries requested by a single SAS script. It reveals a period of two seconds in which the average round-trip time increased from a third of a millisecond to more than 30 milliseconds, a 100-fold increase in procedure times that accounts for the highly-variable and often unacceptable performance of this SAS application. In this case the simplest remedial action is to reconfigure the relevant SAS database library to fetch

many rows in each round-trip, and the next step is to locate the resource that is sometimes severely overloaded. For example, it might be CPU, memory, a disk channel, or a particular disk spindle, but because the plotted round-trip times embrace time spent in the client SAS server as well as the database server and network, the search must cover all areas.

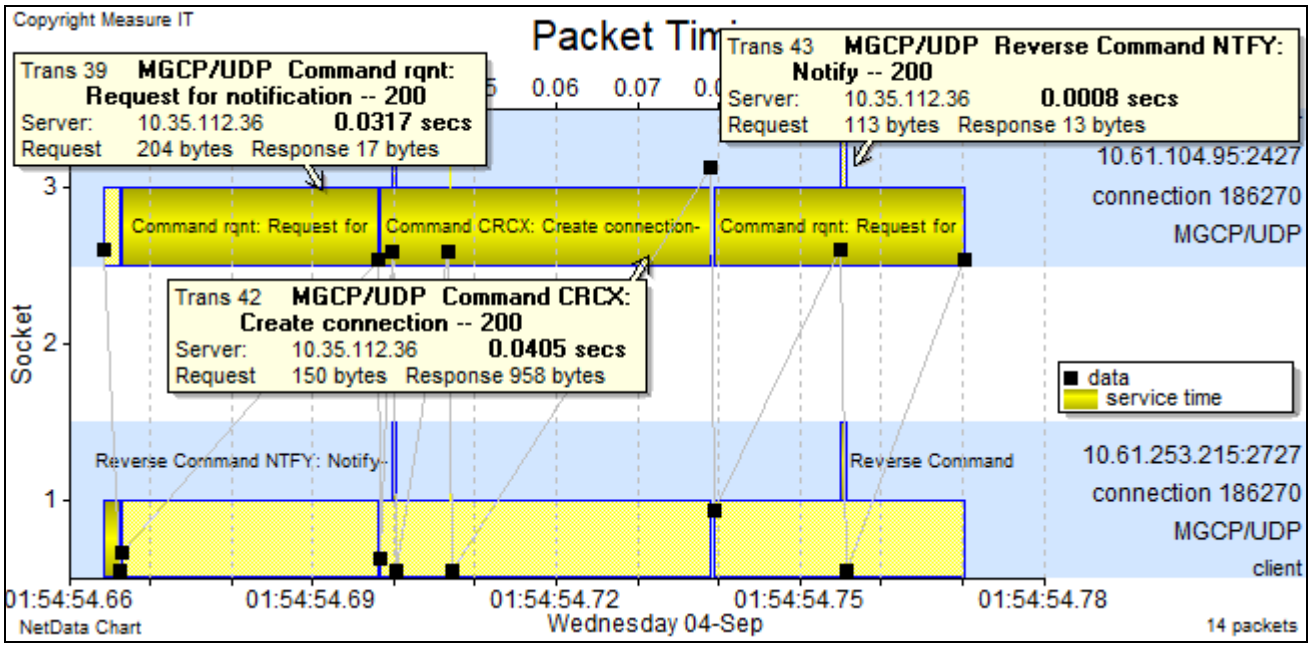


Besides recording pseudo transactions for the logged round-trips, NetData also records a network event that summarises row-fetch statistics for each query, as indicated by the pop-ups on this chart which illustrates fetch performance for several queries.

This case study demonstrates the sometimes critical importance of decoding and viewing message contents, in the context of perfomance charts. Although traffic was captured only in a user's workstation, this analysis – which plots the times for individual round-trips to the database – has characterised the occurrence of severe overloading in the system's backend.

11.2 Media Gateway Control Protocol (MGCP)

NetData has an upgraded decoder for traffic using the Media Gateway Control Protocol (MGCP) that is documented in RFC 3525. This protocol handles configuration commands from a gateway controller such as Cisco CallManager – acting as a client – and port 2427 of media gateway elements (e.g. phones) that terminate media streams and may perform additional functions such as format conversion and media conferencing. Several commands may run concurrently, including a Notify command that is sent from a gateway to the controller.



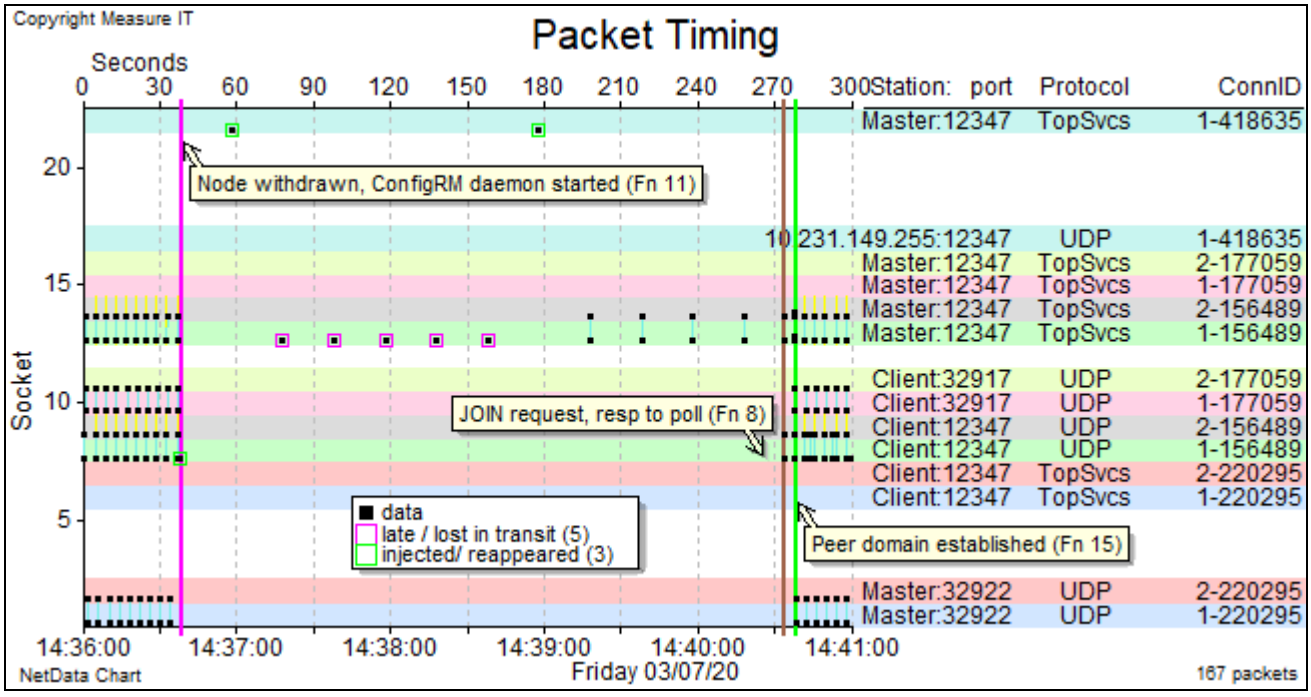
MGCP performs functions similar to, and works with, the Session Initiation Protocol (SIP) which handles communication between gateway controllers. Like SIP, MGCP conveys connection parameters in the Session Description Protocol (SDP).

```
Request      Signature:      Create connection
             Length:         150 bytes
             Frame:          79
  CRCX 461807410 SHOR_0010493CB44A@[10.35.112.36] MGCP 1.0
  Header [97]:
    K Response Ack:       461807408
    C Call ID:            00200002f635c8c7d52005056a36581
    M Connection Mode:    inactive
    N Notified Entity:    s01d11136455bb800@10.35.5.156
Response     Signature:      200
             Length:         958 bytes
             Frame:          84
  200 461807410 OK
  Header [11]:
    I Connection ID:      2039
  SDP Body [968]:
    v: version=           0
    o: owner, sessionID=  Broadcom 8249 461807410 IN IP4 10.35.112.36
    s: session name=      MGCP call
    c: connection=        IN IP4 10.35.112.36
    b: bandwidth=         AS:256
    k: encryption key=    clear:374abb8fb3fde8e85c45efa08ce01c42
    t: time active=       0 0
    m: media, transport=  audio 10000 RTP/AVP 0 8 18 96 97 98 99 9 102 100 101
```

11.3 Topology Services of IBM Reliable Scalable Cluster Technology

Reliable Scalable Cluster Technology (RSCT) underpins many of IBM's High Availability products by managing a domain of peer nodes. Group Services use RSCT to manage a reliable group of applications, and Topology Services manage the network infrastructure connecting a domain of peers. Topology Services, for example, are used by Tivoli System Automation for Multiplatforms (TSAMP).

To establish a peer group and monitor their network, nodes exchange messages in UDP packets to or from port 12347. (Group Services use port 12348.). The third 4-byte integer in the headers of these messages specifies a message type that NetData calls a Function number or labels with names such as 'JOIN Request (8)', 'Heartbeat (10)' and 'PrepareToCommit (13)'. The RSCT decoder has been enhanced to display function names and to create network events to mark the occurrence of significant RSCT messages such as Join Request and Function 15. The following chart plots all the packets captured by two clustered nodes when the client node was rebooted:



Heartbeats ceased during the reboot and the master node repeatedly tried to raise the rebooted client by sending it Function-14 messages (described as polls), first with a broadcast address and then the client's address in unicast packets (markers on socket bands 13 and 14). Eventually the rebooted node responded with a Join request and the cluster was established quickly with the exchange of a few more messages.

11.4 IBM Maestro Tivoli Workload Scheduler (TWS)

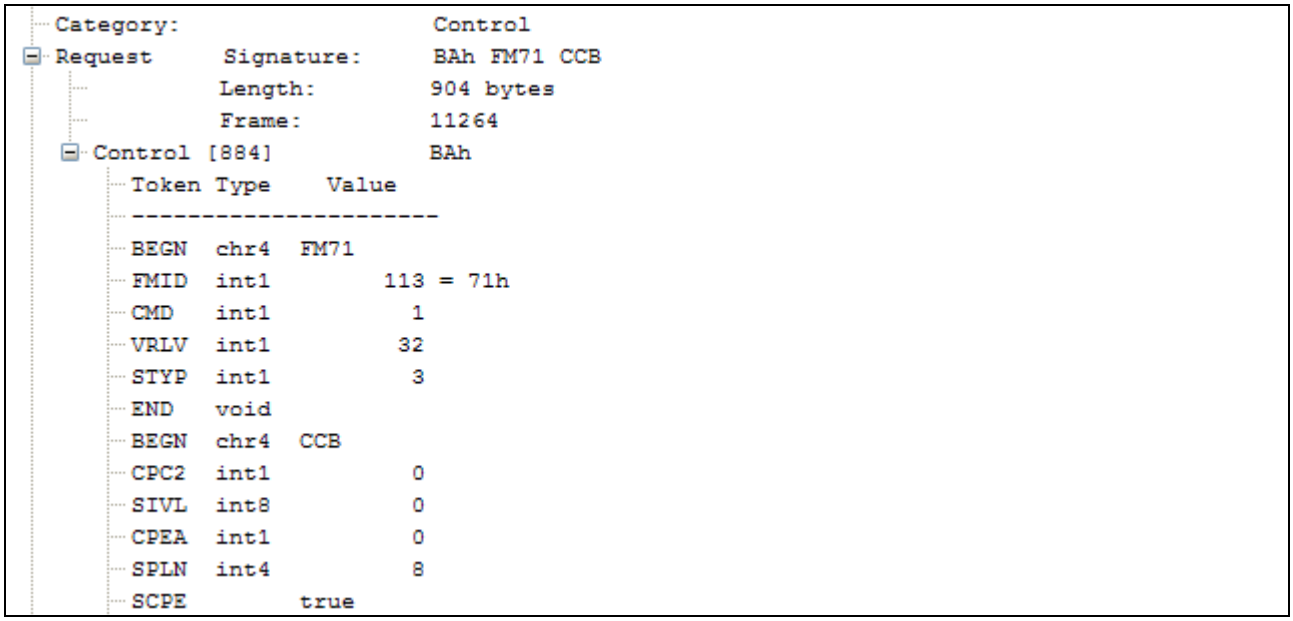
NetData decodes transactions of Tivoli's Workload Scheduler (TWS) Master Domain Manager (DM).

11.5 IBM Tivoli Director Agent

The decoder for Tivoli Director Agent recognises and characterises transactions using either TCP or UDP. The UDP version of the decoder handles transactions initiated by either client or server, and transactions running concurrently.

11.6 IBM Sterling Connect:Direct

The Sterling Connect:Direct file-transfer package is widely used for its reliability and versatility, particularly in mainframe financial systems. Prior to its purchase by Sterling Software and then by IBM it was known as Network Data Mover (NDM). NetData's decoder separates control from data-transfer transactions and presents control parameters in tables of 4-character tokens (parameter names), data types, and values. The signature for a control message begins with a hex code (such as BAh) for flags that indicate message purpose, and then lists the tokens that appear first in every BEGN-END bracket of parameters. The first such token usually appears like FM71 which indicates format number 71 (hex).



Some sessions encrypt all transferred data and most of the control messages with SSL. NetData extracts the SSL records from their Connect:Direct envelopes and describes the SSL handshakes in the same level of detail as it does for HTTPS and other protocols that use SSL.

11.7 IBM Download Director

NetData now recognises traffic of the IBM Download Director which is used by the Update Manager to download software fixes. Because data appears to flow in only direction, from port 7618, NetData doesn't attempt to characterise transactions of this protocol.

11.8 Cisco Gateway Load Balancing Protocol (GLBP)

NetData decodes multicast UDP packets using the Cisco Gateway Load Balancing Protocol (GLBP).

11.9 Spector CNE Decoder

NetData detects and partially decodes the control-centre and data-vault transactions of Spector CNE, software used to monitor and investigate user activity. Transactions with the control-centre server use UDP port 16768, and transactions with the data vault use TCP port 16769. NetData can provide meaning for only a few parameters in the message headers, but because it measures response times the activity of this application can be plotted on charts for correlation with the activity of web transactions that are being monitored.

A monitored workstation runs a Recorder program that is installed secretly and records all user activity with screen snapshots. The recorded information is compressed and encrypted, written to disk and uploaded to the data vault every 15 minutes or so. Could Recorder activity degrade workstation performance?

11.10 Linux Audit Logging

For its solution to the needs of SIEM (Security Information and Event Management) Linux provides an audit framework for logging system calls and other events. An introduction can be found in

> https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sec-understanding_audit_log_files

All logged messages have a common header format containing a timestamp, node name and daemon name, as in

```
Jan 23 15:37:42 <node name> audispd:
```

The format of a message body depends on the daemon to which it is addressed. The daemon *audispd*, an audit event multiplexor, is started by the audit daemon to get events and distribute them to child programs that want to analyse events in near real time. Its messages contain multiple name-value pairs normally conveyed in text strings rather than binary codes.

Every message has at least three fields conveying the node's domain name, the message type and a combined timestamp and message ID, as in NetData's description of a SYSCALL type:

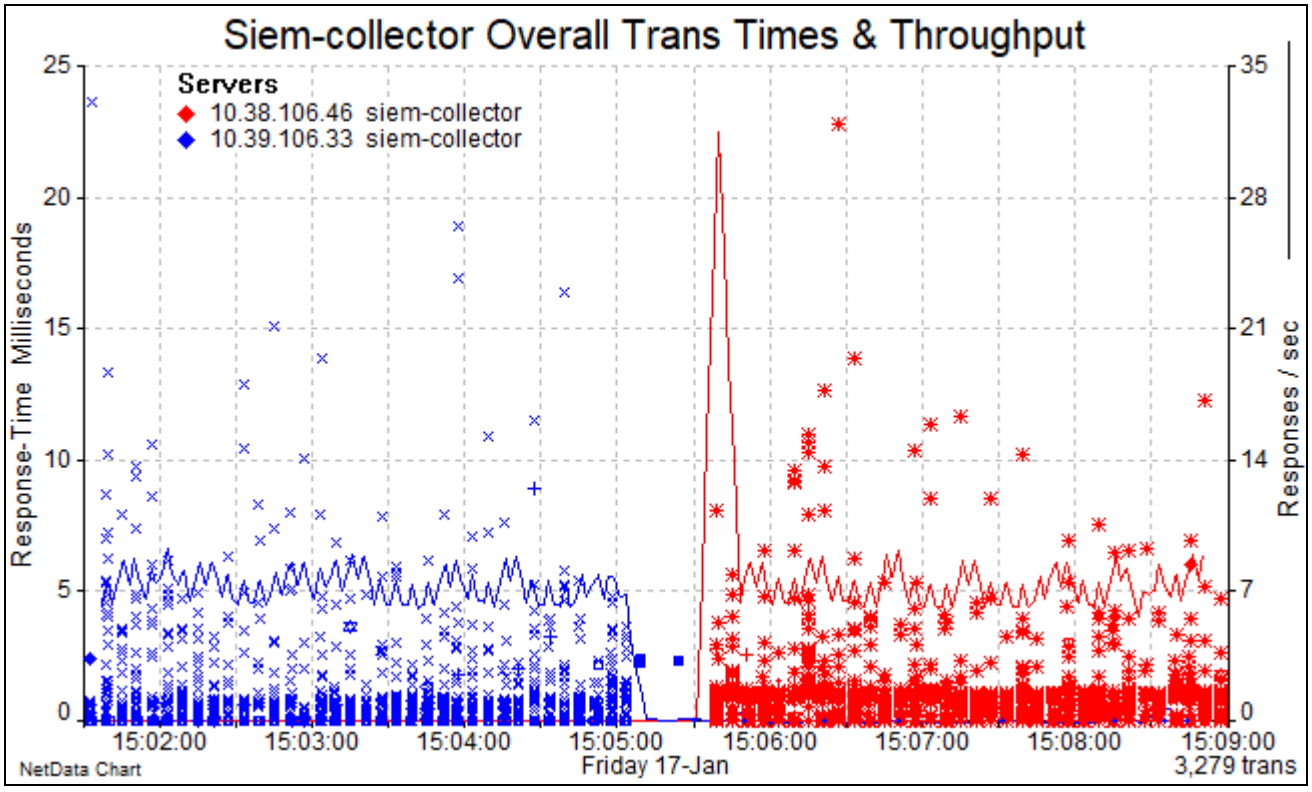
```
Jan 17 15:06:27          <node name> audispd:
                    SYSCALL=sys_mprotect (10) "udapi_slave" "delete"
               yes
    node=                <domain name>
    type=                SYSCALL
    msg=                 audit(1579233987.340:123474224):
    arch=                40000003
    syscall=             10
    per=                 400000
    success=             yes
    exit=                0
    a0=                  ffb9e00c
    a1=                  0
    a2=                  0
    a3=                  0
    comm=                "udapi_slave"
    exe=                 "/opt/LE/UniData/bin/udapi_slave"
    subj=          unconfined_u:unconfined_r:unconfined_t:s0-
s0:c0.c1023
    key=                 "delete"
```

NetData displays the field names and values in a table of two columns and augments the message header with the values for type, syscall code, comm, key and success. NetData translates syscall codes to names such as *sys_mprotect* (memory protection) for code 10. These selected field values also form the request signature of transactions that NetData records to characterise events.

A SYSCALL message is augmented with other messages of different types such as CWD, PATH and EOE that all carry the same msg value and provide additional information about the system call. EOE indicates that a group of related messages is complete. NetData aggregates all the related messages in a single transaction whose request-message time spans the interval from the first to the last message in the group. NetData extracts message IDs to appear in the key-data columns of both the transaction and packet tables.

The four parameters to a system call, a0 to a3, are also extracted into the key-data columns:

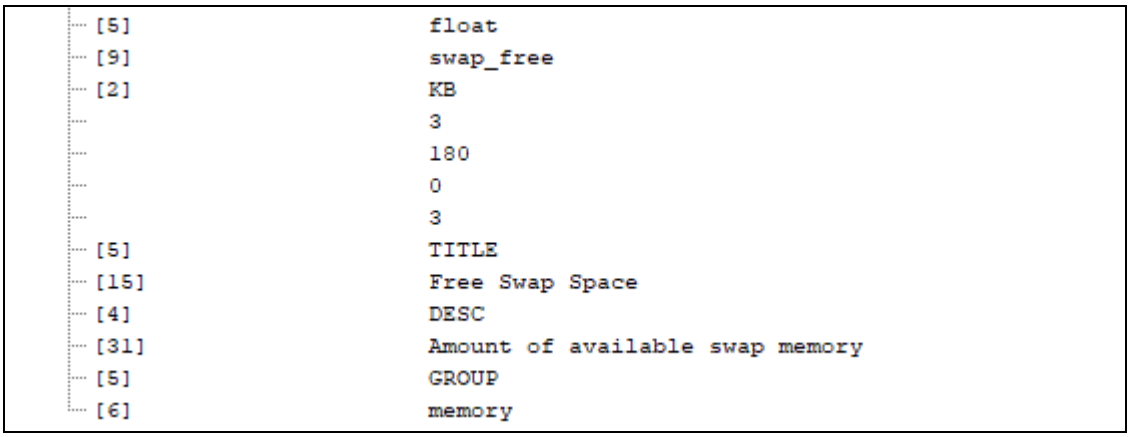
	All Time	Key Data
_mprotect (10) "udtsort" "delete" no; CWD; PATH PARENT; EOE--no resp expected	0.0000	a=8061d40,4ec4ec4f,2,1 123474192 (4)
_mprotect (10) "udtsort" "delete" no; CWD; PATH PARENT; EOE--no resp expected	0.0005	a=8061d40,4ec4ec4f,3,1 123474193 (4)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...		a=ff901adc,0,35,a2625bc 123474194 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0003	a=ff9019dc,0,6,0 123474195 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0002	a=ffb9e07c,0,1c,901d9fc 123474196 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0000	a=ff9019dc,0,0,0 123474197 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0008	a=ffb9df7c,0,a,0 123474198 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0010	a=ffb9e09c,0,0,0 123474199 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0015	a=ffb9e00c,0,0,0 123474200 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0011	a=ffb9e09c,0,0,0 123474201 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0021	a=ffb9e09c,0,0,0 123474202 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0010	a=ffb9e00c,0,0,0 123474203 (5)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...	0.0010	a=ffb9e09c,0,0,0 123474204 (5)
_mprotect (10) "udtsort" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE--no r...	0.0002	a=8061d40,8241,3,0 123474205 (5)
_mprotect (10) "udtsort" "delete" no; CWD; PATH PARENT; EOE--no resp expected	0.0000	a=8061d40,4ec4ec4f,2,1 123474206 (4)
_mprotect (10) "udtsort" "delete" no; CWD; PATH PARENT; EOE--no resp expected	0.0007	a=8061d40,4ec4ec4f,3,1 123474207 (4)
_mprotect (10) "udapi_slave" "delete" yes; CWD; PATH PARENT; PATH DELETE; EOE...		a=ffb9e19c,0,35,9040b9c 123474208 (5)

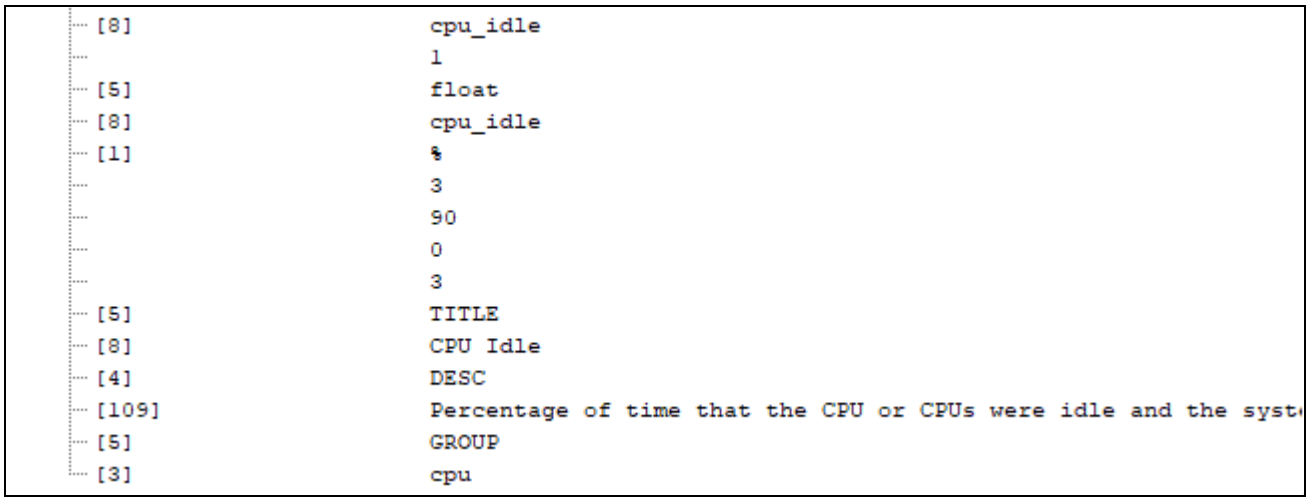


Logged event messages appear here in bursts at intervals of six seconds. After a pause of 30 seconds that preceded a new connection with a different collector, the event backlog generated a very high message rate.

11.11 Ganglia Monitoring System

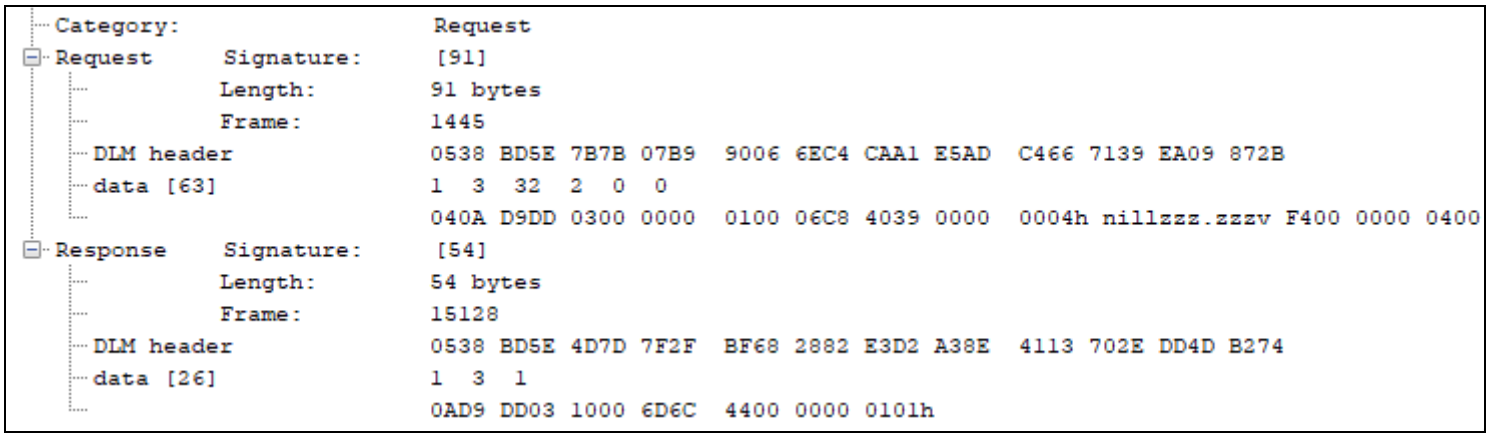
Ganglia is an open-source scalable, distributed monitoring tool for high-performance computing systems, clusters and networks. The software is used to view either live or recorded statistics covering metrics such as CPU load averages or network utilization for many nodes. NetData characterises messages sent to Ganglia daemons and tags the traffic as *gmond.* The daemon uses UDP port 8649.





11.12 Compuware Distributed License Management (DLM)

NetData characterises transactions of Compuware's Distributed License Management (DLM) server that normally use TCP port 7188.



11.13 Cisco Meraki Cloud Dashboard

NetData detects Cisco Meraki dashboard traffic that uses UDP port 9350.

11.14 Cisco EnergyWise

NetData detects traffic from devices that participate in a Cisco EnergyWise system.

11.15 Microsft Office Groove LAN Device Presence Protocol (DPP)

NetData detects traffic from devices that broadcast their presence to UDP port 1211 using the Microsft Office Groove LAN Device Presence Protocol (DPP).

11.16 Remote Management Control Protocol (RMCP)

NetData decodes UDP traffic of the Remote Management Control Protocol (RMCP) that carries requests and responses of the Intelligent Platform Management Interface (IPMI) through port 623. IPMI is an initiative of Intel that has been adopted by more than 200 computer system vendors, including IBM, Dell and HP, for remote management of PCs and other networked devices to reduce total cost of ownership.

11.17 Argent Infrastructure Monitoring

NetData decodes the traffic of Argent monitoring consoles and three Argent engines: Argent Guardian for monitoring servers; Argent SNMP for monitoring network devices and handling traps; and data-consolidation transfers.

11.18 Snare Event-Log Collection

NetData now recognises logged system events that are sent in UDP packets from Snare agents to port 6161 of a Snare server for the "System iNtrusion Analysis & Reporting Environment" of Intersect Alliance. NetData displays the contents of individual events in the Contents column of the packet table, and accumulates events in the request message of a group transaction for each Snare connection, forming event tables as in this extract:

SubmitTime	Source	User	LogType	ComputerName	ExpandedString
Tue Aug 14 11:03:39 ...	VMware View	NT AUTHORITY\S...	Warning	SRV4pyd01.central...	Error retireving user for SID S-1-5-21-6776287.
Tue Aug 14 11:03:39 ...	VMware View	NT AUTHORITY\S...	Warning	SRV4pyd01.central...	Error retireving user for SID S-1-5-21-6776287.
Tue Aug 14 11:03:39 ...	VMware View	NT AUTHORITY\S...	Warning	SRV4pyd01.central...	Error retireving user for SID S-1-5-21-6776287.
Tue Aug 14 11:03:42 ...	VMware View	NT AUTHORITY\S...	Information	SRV4pyd01.central...	(SESSION:423EC47A2A8E9EA3A386B8D4F..
Tue Aug 14 11:03:45 ...	VMware View	NT AUTHORITY\S...	Information	SRV4pyd01.central...	CONNECTED:Server:cn=0183c549-4fe7-40b.
Tue Aug 14 11:03:46 ...	VMware View	NT AUTHORITY\S...	Warning	SRV4pyd01.central...	(SESSION:423EC47A2A8E9EA3A386B8D4F..
Tue Aug 14 11:03:46 ...	VMware View	NT AUTHORITY\S...	Warning	SRV4pyd01.central...	(SESSION:423EC47A2A8E9EA3A386B8D4F..
Tue Aug 14 11:03:46 ...	VMware View	NT AUTHORITY\S...	Warning	SRV4pyd01.central...	(SESSION:423EC47A2A8E9EA3A386B8D4F..
Tue Aug 14 11:03:46 ...	VMware View	NT AUTHORITY\S...	Information	SRV4pyd01.central...	PENDING:Server:cn=390b3a7c-cfe7-4997-9..
Tue Aug 14 11:03:47 ...	VMware View	NT AUTHORITY\S...	Information	SRV4pyd01.central...	CONNECTED:Server:cn=261b54db-33b4-41..
Tue Aug 14 11:03:49 ...	VMware View	NT AUTHORITY\S...	Warning	SRV4pyd01.central...	Error retireving user for SID S-1-5-21-6776287.
Tue Aug 14 11:03:49 ...	VMware View	NT AUTHORITY\S...	Warning	SRV4pyd01.central...	Error retireving user for SID S-1-5-21-6776287.

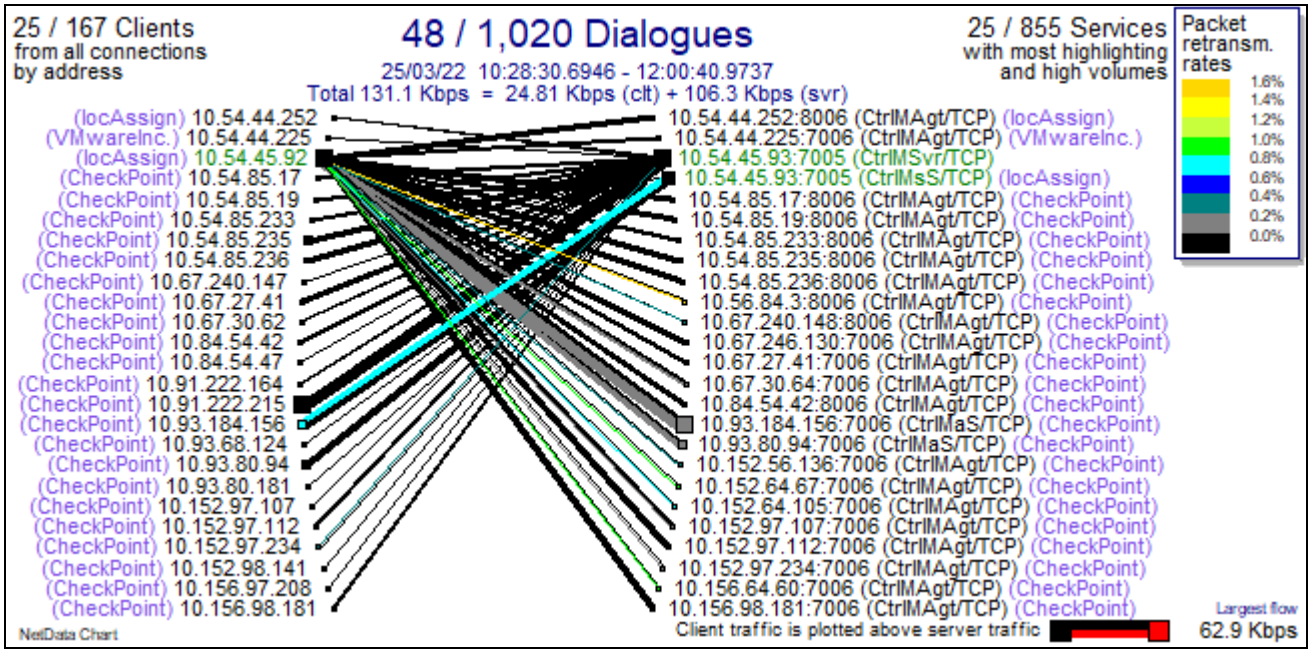
11.19 Nagios NSClient++ Windows Monitor

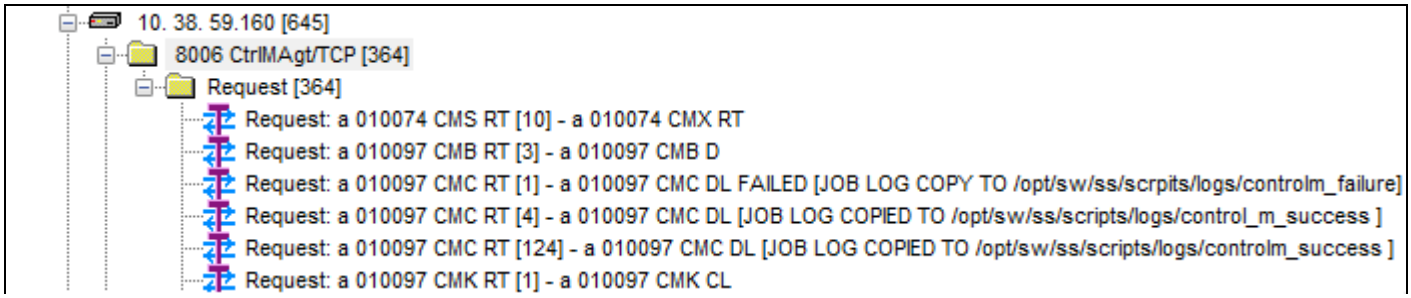
NetData can characterise the transactions of Nagios open-source software that monitors the Windows operating system in remote machines that run NSClient++. This traffic usually uses TCP port 12489 on the Windows machines.

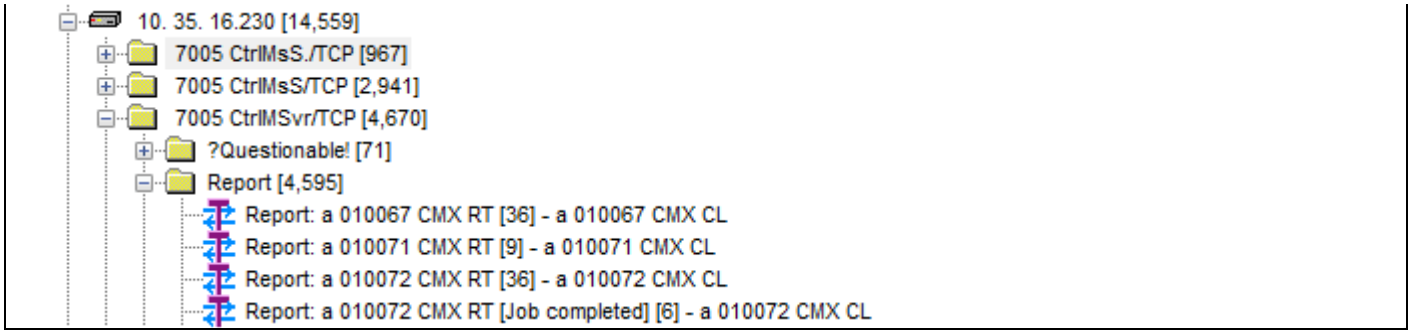
11.20 BMC Control-M Workflow Manager

The BMC Control-M workflow management system involves a server and many agents as depicted in the dialogue chart below. The server sends requests to agents through their TCP port 7006, and agents send reports in round-trips to the server through its TCP port 7005. These two port numbers are defaults and similar agent numbers are often allocated, such as 7016 and 8006. NetData recognises Control-M traffic if the port numbers end in 5 or 6.

Connections may be either encrypted or in clear and NetData handles both through the one port if necessary.







All messages in clear are rendered entirely in ASCII code. They begin with the two letters ‘a ‘ and two length indicators. The header portion contains no formatting information to identify such fields as a timestamp and a message index number that starts with 01 for each transaction. The optional

data portion contains some field-type codes such as F, C and U with occasional length indicators. In NetData's transaction description such metadata appears in the first column.

```
Protocols:                 BMC Control-M workflow agent / Transmission Control Protocol
Category:                  Request
Request     Signature:     a 010098 CMC RT
            Length:        279 bytes
            Frame:         333738
   Function                a 010098 CMC RT   2022/03/25 11:38:53.09
                           2833604884730000
      index                01
           [39]            collections-bat.prod.srv.distribution.com
           [09]            cxprodctm
           [02]            OS
           [00]
      data [220]           030159
         [F032]            E:\Logs\Collections\Batch\cntrlm
                           3m6rf00001
         [10]              k9wadhoc03
         [10]              k9wadhoc03
         [C15]             SRV-CTRLM-TALLY
                           0517100
         [U039]            collections-bat.prod.srv.distribution.com
                           00000P0000000000 0000

Response    Signature:     a 010099 CMC DL FAILED [JOB LOG COPY TO E:\Logs\Collections\Batch\cntrlm]
                           [Error: The requested operation could not be completed due to a file
                              system limitation]
            Length:        363 bytes
            Frame:         336281
   Function                a 010099 CMC DL   2022/03/25 11:39:30.09
                           2833604884730000
      index                02
           [39]            collections-bat.prod.srv.distribution.com
           [09]            cxprodctm
           [03]            WIN
           [00]
      data [253]           22024220221139ct3m6rf0012220JOB LOG COPY TO
                              E:\Logs\Collections\Batch\cntrlm
      FAILED               Cannot copyysout of order 3m6rf run 00001 to
                              E:\Logs\Collections\Batch\cntrlm.
      Error:               The requested operation could not be completed due to a file system
                              limitation
```

Transaction-request and -response signatures are formed from key elements in the message headers. The first phrase in quotes and any warning or error descriptions are appended to their respective signature so that significant transactions can be found in the transaction-class tree.

11.21 Microsoft High Performance Computing (HPC)

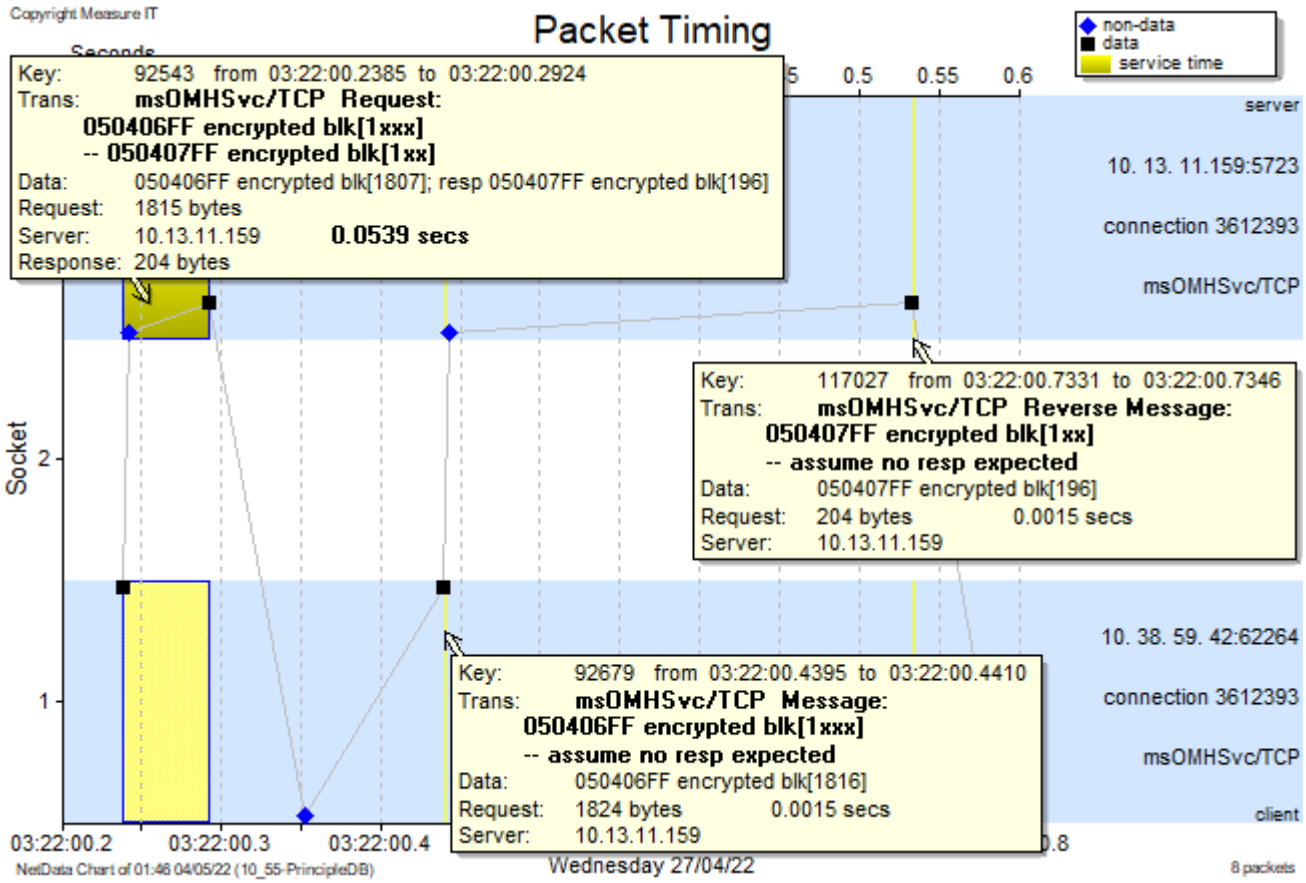
NetData detects and decodes as far as possible the many protocols associated with Microsoft's High Performance Computing (HPC) solution for cluster computing. The protocols allow access to a Management Service, a System Definition Model (SDM) Service and a Job Scheduler Service (JSS) on the head node, from compute nodes, from a cluster manager, and from command-line tools. Another protocol handles communication from the JSS on the head node to the Node Manager on compute nodes. Altogether NetData presently distinguishes 10 different HPC protocols.

11.22 Microsoft Operations Manager Health Service

NetData reconstructs transactions of the Operations Manager Health Service (OMHS), and decodes their messages as far as encryption allows.

The data in OMHS messages is usually encrypted and the messages don't always form request-response pairs initiated by the client. However, NetData characterises what appear to be round-trips whether initiated by client or server, and unidirectional flows in either direction, without responses.

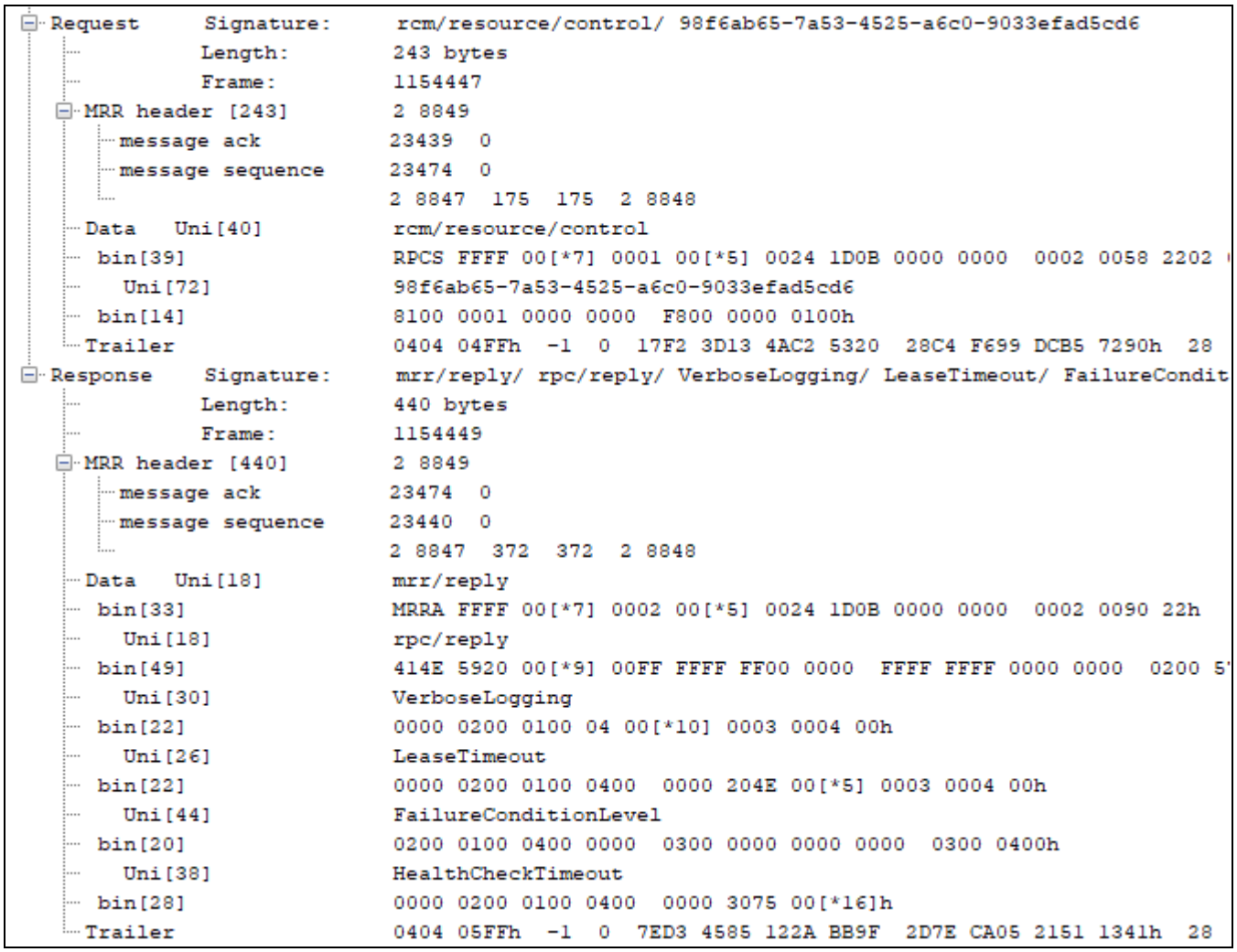
The chart below characterises three such pseudo transactions. The first is a standard round-trip from the client; the next two transactions comprise independent messages, first from the client and then from the server.



11.23 Microsoft Cluster Service and Multicast Request Reply

Microsoft software provides extensive cluster services to support high-availability monitoring and failover for groups of independent computers running Windows applications. The first sign of a cluster service on a network is likely to be an exchange of heartbeat messages between pairs of nodes using UDP port 3343 for both source and destination. Heartbeat messages expect a heartbeat response and both client and server initiate heartbeat round-trips independently. The times of these round-trips can be plotted on the performance chart.

Most of the information passed between nodes uses a communication procedure called *Multicast Request Reply (MRR)* that is like an RPC but can issue a request to multiple recipients and get responses from all of them. To ensure that the communication is reliable it uses TCP, and if a TCP connection breaks it is automatically restored. However, to ensure reliable delivery when a connection is restored, MRR adds its own message sequence numbers and acknowledgements, and they can be seen in the message headers of NetData's transaction descriptions:



To display message payloads NetData lists the interleaved portions of binary code (mostly in hex) and Unicode text strings. The sequences of text strings are adopted as the request and response signatures that characterise different transaction types.

Messages are usually conveyed in IPv6 frames between TCP port 3343 on one node and an ephemeral port on the other node. Request messages may be issued by either node. The IPv6 frames are then encapsulated in IPv4 packets and conveyed between UDP ports 3343 on both nodes, like the heartbeat messages. As it does with all other encapsulations, NetData records the outer-header MAC addresses and UDP socket details in the frame-description column of the packet table:

```
Source:              LL0:2Df9:AAB8:761E:9583:55809
Destination:         LL0:F4C3:E0D7:28FB:315C:3343
Frame description:   outer MAC 0010DB-FF901B -> 005056-98EE7C;
                     UDP 10.79.59.48:3343 -> 10.78.59.4257:3343
```

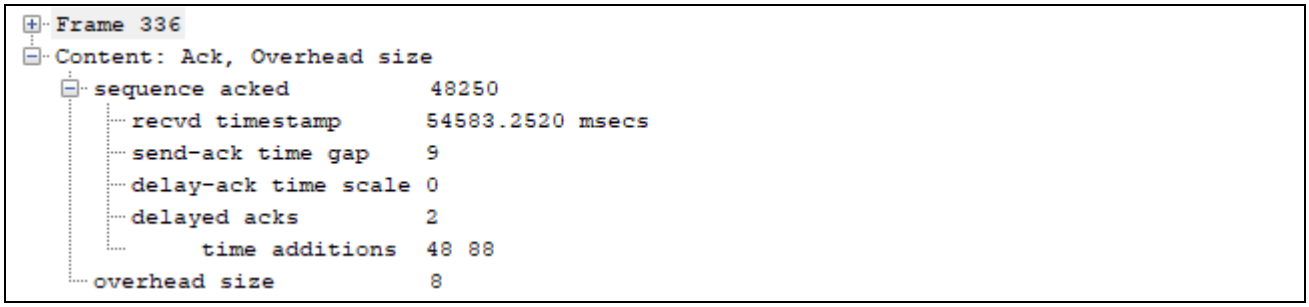
NetData tags the UDP heartbeat traffic as ‘clustNet’ and the encapsulated MRR traffic as ‘msMRR’. Some UDP packets encapsulate IPv4 multicast name-resolution frames (LLMNR) to UDP port 5355.

11.24 Microsoft Remote Desktop on UDP with Extended Transport

When controlling a remote desktop Microsoft Remote Desktop Protocol (RDP) usually establishes two connections with port 3389 on the remote machine, one connection using TCP and the other using an extended transport protocol on UDP. The payload on both connections is encrypted with TLS 1.2 (SSL 3.3).

The extended transport protocol, tagged by NetData as 'RDPeT2', is a second version specified in the Microsoft document [MS_RDPEUDP2]. It provides functions of lost-data recovery and flow control similar to TCP but is more efficient and avoids some of TCP's weaknesses. Its header can be understood by constructing it in three steps. The first two bytes specify a window size and in a set of six flags specify what options follow: an acknowledgement with a timestamp and delay information; an overhead-size indicator derived from the average length of headers preceding received data payloads; delayed-ack information; the lowest unacknowledged sequence number ('ack of acks'); an ack vector (providing the functions of a SACK); and the data payload preceded by a channel sequence number.

The second step prefixes the header with a single byte that handles ambiguities with very short packets and in future may indicate a higher-level packet type. The final step is to change byte order, swapping bytes one and eight.



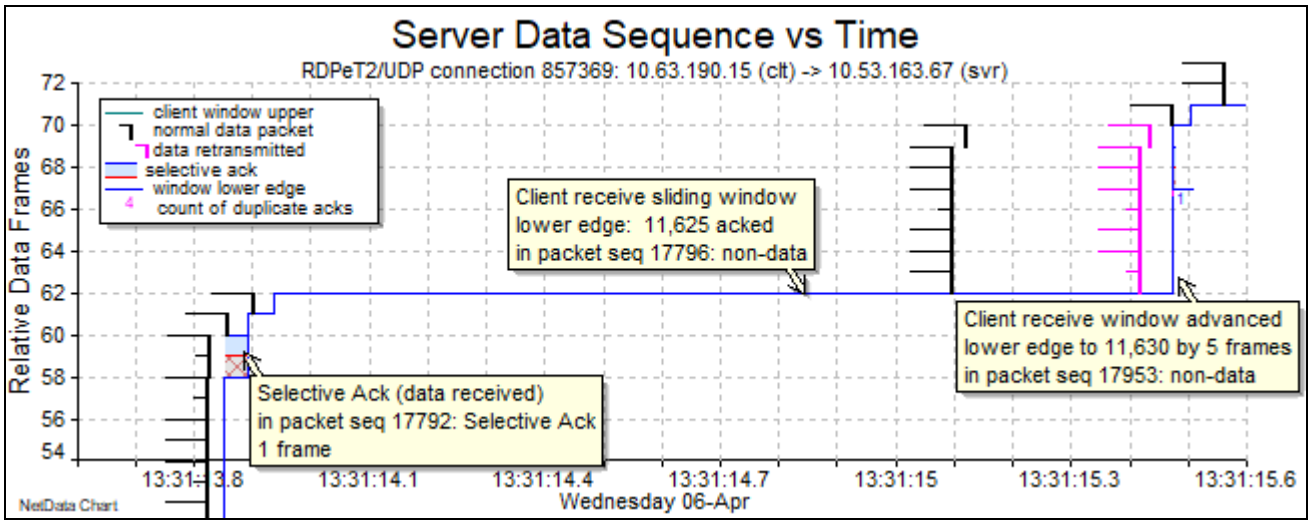
Packets with data carry two sequence numbers: a transmitted-packet number that always increments by one, for detection of packet losses; and a channel- or data-sequence number that identifies the original data segment should it be retransmitted.

Packet-sequence and ack numbers appear in the Context column of the packet table, preserving the normal Data Seq and Data Ack columns for data-sequence numbers, the numbers that can be plotted on the flow chart. The data-ack number is calculated by NetData from the ack number in the transport header and taking into account the most recent, relevant difference between packet- and data-sequence numbers, and adding one to identify the next expected data-sequence number.

IP ID	Data Seq	Data Ack	AppType	Data	Blks	Function	Context	Content
58752	47635		RDsktPS./RDPeT2	1205	3	Data, Delay ack info;...	48250	delayed acks max 74\ ...
26572		47636	RDPeT2			Ack, Overhead size	ack48250	sequence acked 4763...
58755	47636		RDsktPS./RDPeT2	1230	1	Data; AppData	48251	AppData[42]
58756	47637		RDsktPS./RDPeT2	703	2	Data; AppData (2)	48252	AppData[1628]\ AppD...
26573	6345	47638	RDsktPS./RDPeT2	215	1	Ack, Data, Overhea...	7146; ack48252	sequence acked 4763...
58759		6346	RDPeT2			Ack, Overhead size	ack7146	sequence acked 6346...
58761	47638		RDsktPS./RDPeT2	1043	2	Data; AppData (2)	48253	AppData[42]\ AppData...
26574		47639	RDPeT2			Ack	ack48253	sequence acked 4763...
58762	47639		RDsktPS./RDPeT2	1230	1	Data; AppData	48254	AppData[42]

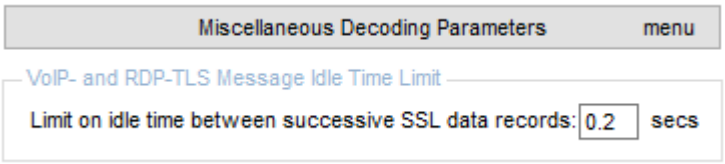
The resulting sliding-window graphs on the flow chart look like TCP sliding windows apart from the fact that sequence numbers refer to whole data segments rather than individual bytes.

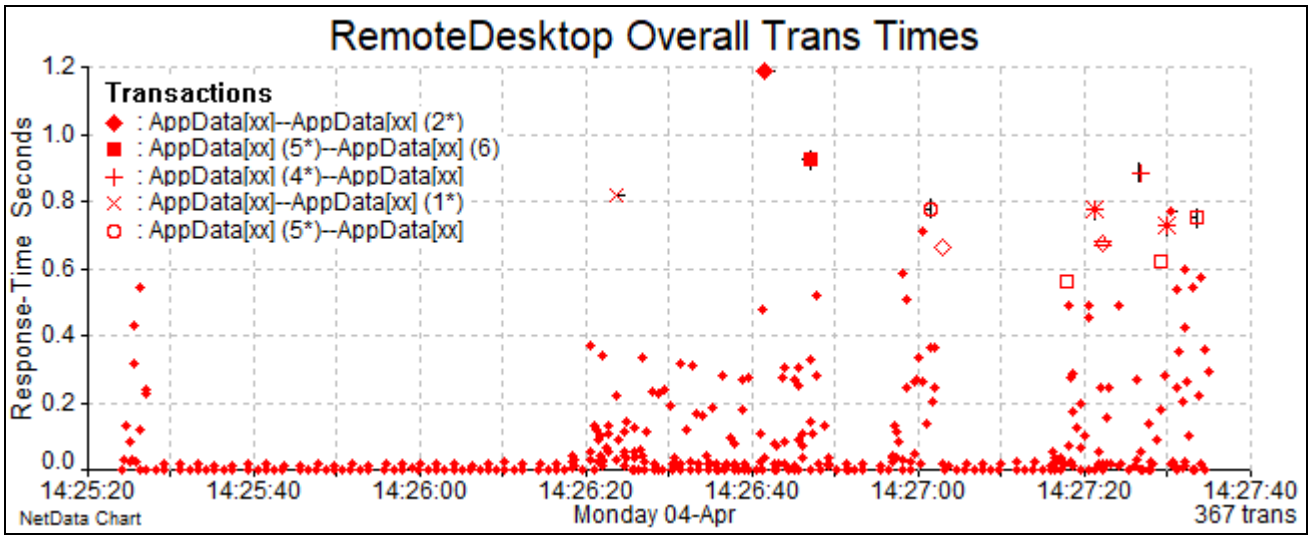
Retransmitted data segments are highlighted and the information in ack vectors is displayed like TCP selective acks.

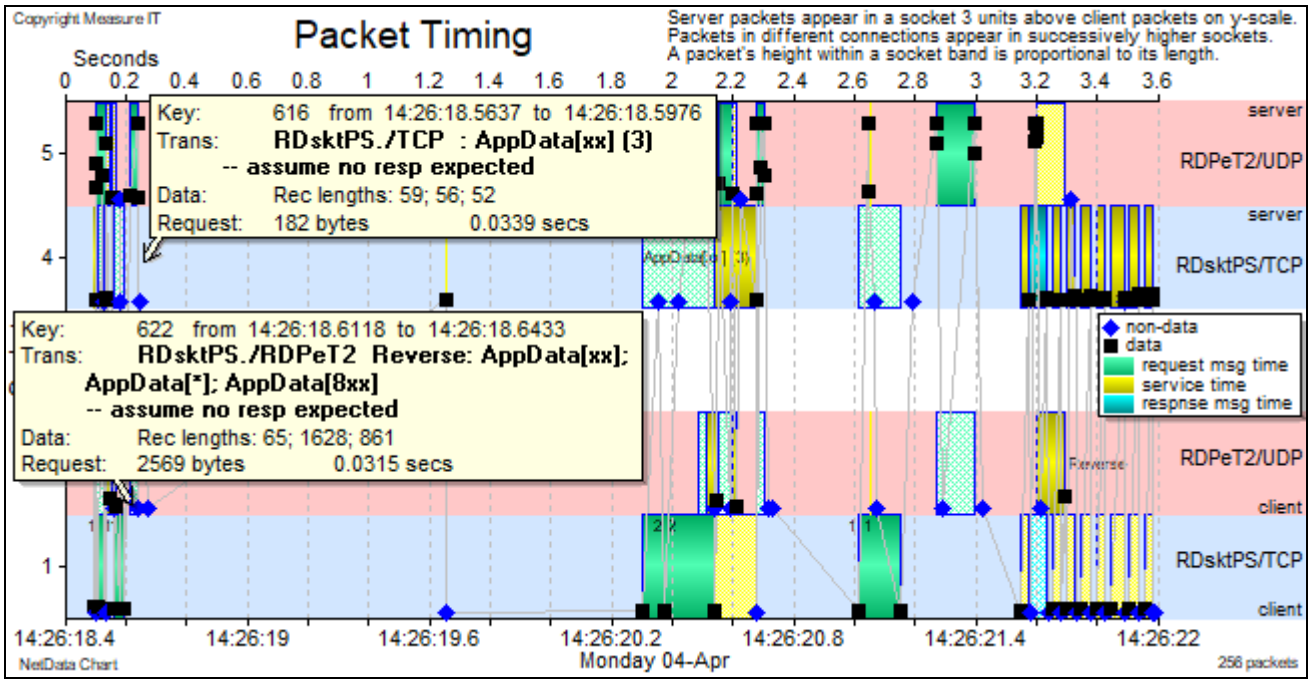


NetData unpacks every header and characterises pseudo transactions derived from flows of TLS records. Not all RDP packet sequences form pairs of request and response messages but this TLS decoder characterises some sustained data flows in one direction only, and exchanges that appear to be round-trips, whether initiated by the client or the server. Such a pseudo transaction is terminated at every occurrence of an idle period longer that 200 ms. This scheme reveals patterns of RDP activity on both the performance and timing charts for correlation with other system behaviour, but avoids presenting pseudo transactions with misleadingly-large response times.

The transaction-controlling idle period can be changed in a miscellaneous decoding parameter under the heading ‘VoIP- and RDP-TLS Message Idle-Time Limit’ on the Decoding page of controls:

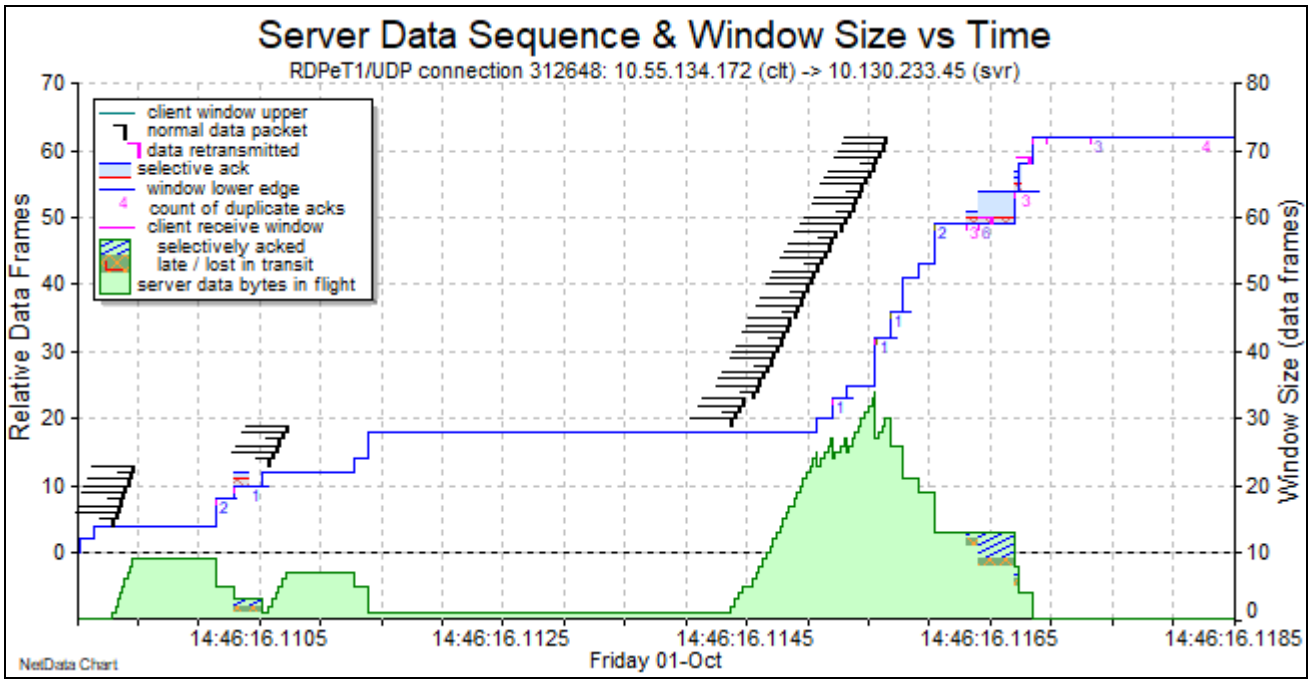






This packet-timing chart compares the activity on both the TCP (grey band) and UDP (pink) connections. The two popups descibe data flows without a response ('assume no response expected'). As in this example most of the apparent round-trips and flows without responses on the UDP connection are initiated by the server rather than the client ('Reverse: AppData[xx]').

NetData's decoder for the first version of the extended transport protocol, tagged by NetData as 'RDPeT1', has been upgraded to the same standard as the RDPeT2 decoder. The green area on the window-size chart below plots the number of data packets in flight, waiting for acknowledgement, and the sliding window shows how the server retransmits data segments when prompted by ack vectors from the client.



11.25 Flexera (ex Globetrotter) Software FlexLM Licence Manager

NetData characterises and measures transactions handled by the Flexera (ex Globetrotter) Software FlexLM licence-manager daemon.

11.26 Citrix ICA Session Reliability

NetData detects Citrix ICA traffic to port 2598. This traffic uses the Citrix Session Reliability feature that re-opens TCP connections that fail.

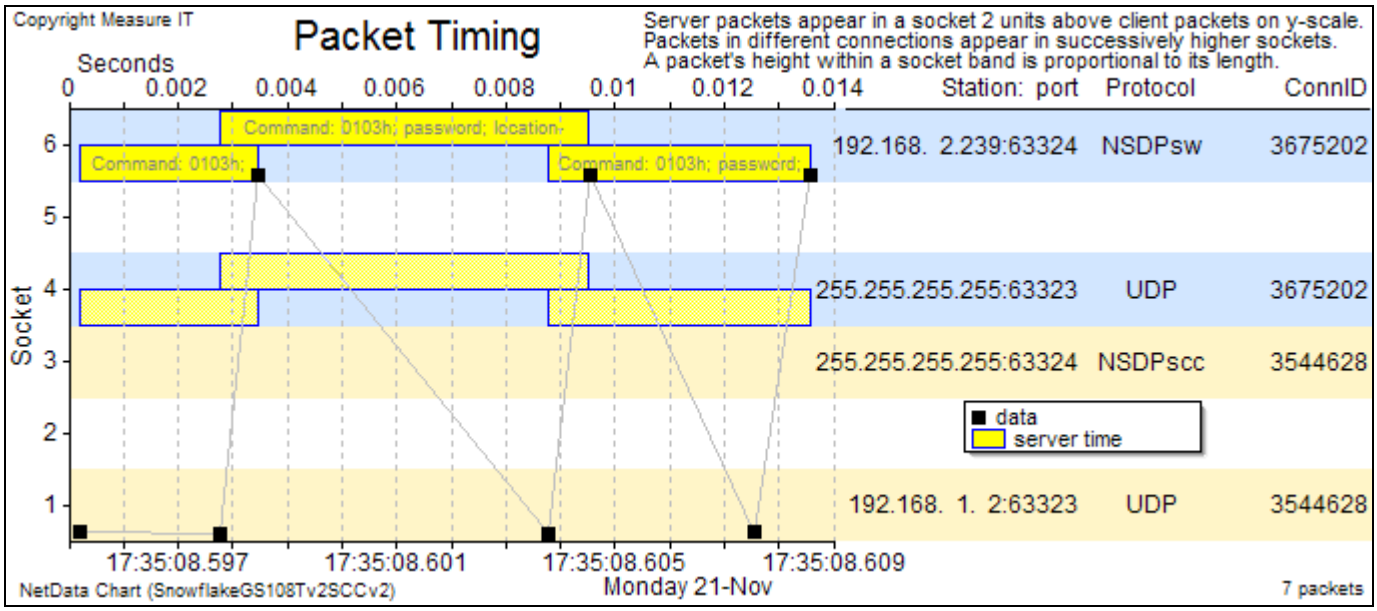
11.27 Netgear Switch Discovery Protocol

The Netgear Switch (or Smart) Discovery Protocol (NSDP) allows Netgear's Smart Control Centre (SCC) to discover and communicate with Netgear switches without knowledge of their IP addresses. The SCC broadcasts UDP probes or commands from port 63323 to port 63324, and relevant switches respond with UDP broadcasts from port 63324 to port 63323. NetData decodes the broadcast packets and matches responses with their probes or commands to characterise the transactions. The response times are unlikely to be significant, but it is useful to know the purpose of these IP broadcasts and to diagnose any problems in the management system.

A sixth colour – orange – has been added for packets in the packets table, to identify broadcast packets that serve as responses rather than requests.

	Time Of Day	Seq	Source	Destination	Len	Hdr	Net	Tspt	ConnID	AppType	Data	Function
■	17:55:34.1283	124	SNOWFLAKE:63323	Broadcast:63324	86	46	IP4	UDP	661022	NSDPscc	40	Request 0101h
■	17:55:34.1309	125	192.168.1.239:63324	Broadcast:63323	90	46	IP4	UDP	398377	NSDPsw	44	Response 0102h
■	17:55:37.0761	126	SNOWFLAKE:63323	Broadcast:63324	134	46	IP4	UDP	661022	NSDPscc	88	Probe 0101h
■	17:55:37.0794	127	192.168.1.239:63324	Broadcast:63323	216	46	IP4	UDP	398377	NSDPsw	170	Response 0102h
■	17:56:34.7886	128	SNOWFLAKE:63323	Broadcast:63324	123	46	IP4	UDP	661022	NSDPscc	77	Command 0103h
■	17:56:34.7914	129	192.168.1.239:63324	Broadcast:63323	82	46	IP4	UDP	398377	NSDPsw	36	Response 0104h
■	17:56:34.7917	130	SNOWFLAKE:63323	Broadcast:63324	108	46	IP4	UDP	661022	NSDPscc	62	Command 0103h
■	17:56:34.797	131	192.168.1.239:63324	Broadcast:63323	82	46	IP4	UDP	398377	NSDPsw	36	Response 0104h

If the user submits many configuration changes at the same time CSS splits them into separate but concurrent broadcast round-trips with successive sequence numbers, as NetData shows in the following chart.



However, when changing the switch's IP address, CSS sets the switch address, gateway address and subnet mask together in the one round-trip.

11.28 IBM and SAS Grid Computing

NetData now decodes the traffic of some protocols used to manage grid computing, specifically those used in the SAS Intelligent Platform for enterprise analytics, and those associated with IBM's Platform Load Sharing Facility (LSF). The SAS protocols usually encrypt all the TCP payload except for a four-byte length indicator at the start of each message block and NetData relies on the use of default port numbers to detect these protocols confidently. The IBM protocol messages, however, are in clear and will be recognised irrespective of their port numbers.

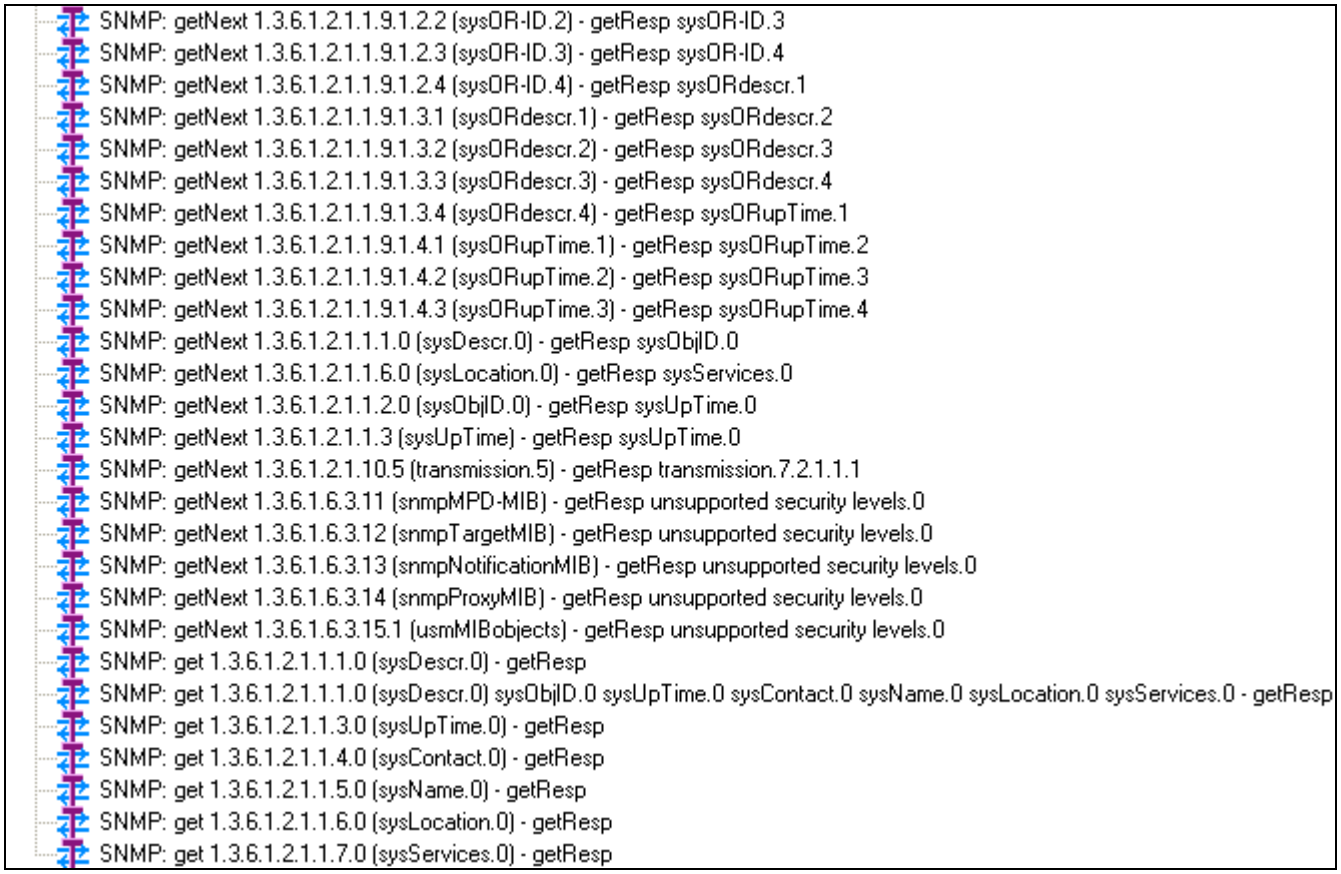
Large response messages in the LSF protocol are split into many blocks with simplified headers that are easy to recognise because they contain two fields of 4 and 2 bytes whose respective contents in hex notation are BADC0FFE and F00D.

Default Port Numbers	NetData Tag	Description
6881	LSFmbD	Load Sharing Facility Master Batch Daemon (MBD) for job scheduling
6882	LSFsbD	Load Sharing Facility Slave Batch Daemon (SBD) for job execution
7969	LSF_LIM	LSF Load Information Manager (LIM) (both UDP and TCP)
8561	SASmetad	SAS Intelligence Platform Metadata Server
8581-8584	SASspawn	SAS Intelligence Platform Object Spawner
8591	SASwspc	SAS Intelligence Platform Workspace Server

11.29 Simple Network Management Protocol (SNMP)

The SNMP (Simple Network Management Protocol) decoder represents Object IDs by the conventional dotted-decimal notation and for many objects adds a name in parentheses for part or all of the objects. Object IDs and names that are the subject of Get and Set requests are included in transaction request and response signatures, and the values of objects are recorded as key data to appear in the transaction table's Data column.

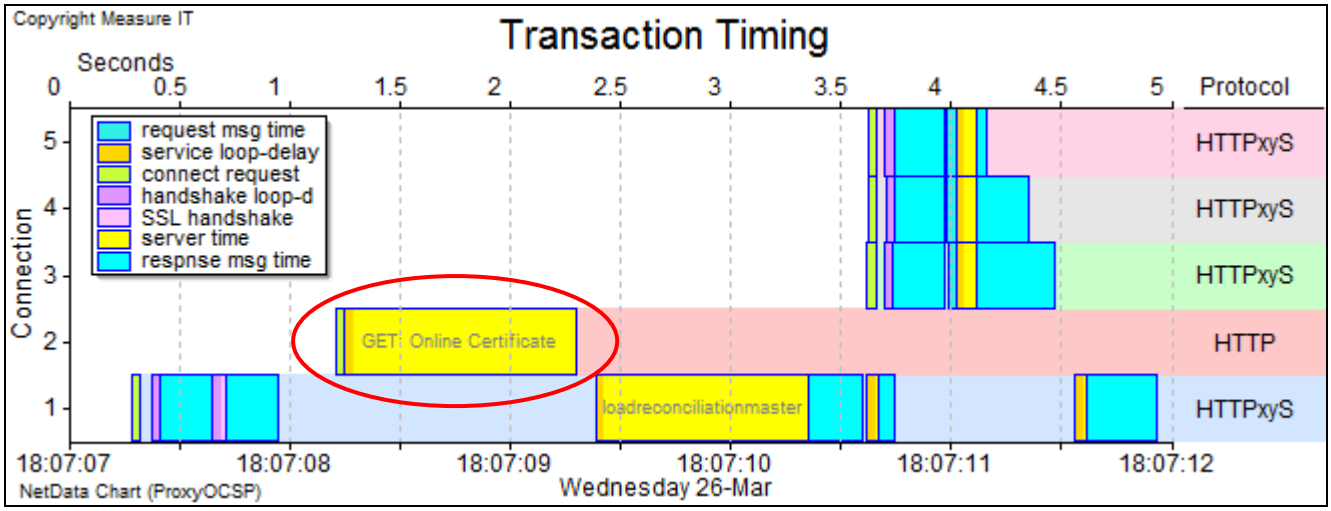
It is possible to select in the transaction tree the Get requests for a particular object, load those Get transactions, and plot the object values against time-of-day on the performance chart. Network error rates or performance events can be correlated with any measurements recorded in a MIB in any device on the network, provided a management tool reads the measurements at regular intervals while the traffic is captured. If an already-installed network management system can't poll the required objects there are free MIB browsers that will poll objects and plot graphs of measurements in real time.



11.30 Online Certificate Status Protocol

When a computer relies on a digital certificate received from another system it should at regular intervals check whether the certificate has been revoked by the issuing Certificate Authority (CA). Checks were originally made by requesting from the CA its Certificate Revocation List (CRL). The preferred method now is to send to the CA or its agent a query that refers to a particular certificate, using the Online Certificate Status Protocol (OCSP). The response to such a query repeats the certificate's identification, indicates the status of the certificate, appends an authenticating signature, and appends the responder's certificate.

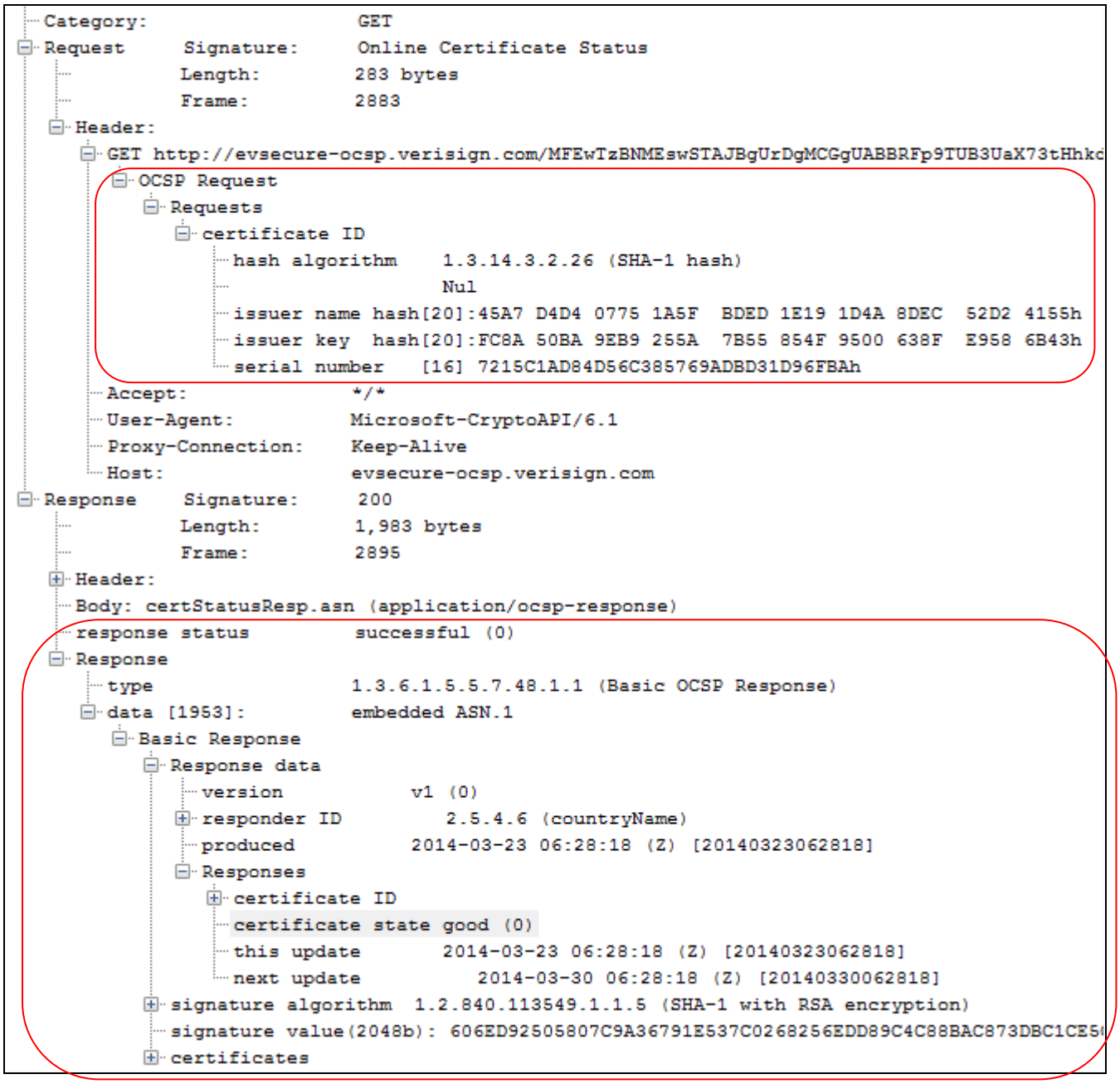
NetData decodes the requests and responses in OCSP/HTTP transactions. The OCSP messages are encoded in the Distinguished Encoding Rules (DER) of Abstract Syntax Notation One (ASN.1), as are digital certificates and revocation lists. Response messages are always conveyed as an HTTP body. Large request messages are conveyed as the bodies of HTTP Post requests, but short requests are further encoded in base-64 (for conversion to text), then converted to a URL (replacing each reserved character with a percent sign and a pair of hexadecimal digits), and conveyed as the URL of an HTTP Get request.



This chart plots the loading of a web page that required five supporting files. The transaction began with a new connection through a proxy server and two handshake round-trips to open a secure session, but before the page file could be requested the browser checked the status of the server's certificate by sending an OCSP request to the CA. In this case certificate checking held up the page request by 1.5 seconds.

The panel below displays part of NetData's description of an OCSP transaction with Verisign. The request message in this case was conveyed in the URL and its description appears within the header description and below the URL. For the transaction's request signature NetData has replaced the long URL with the phrase 'Online Certificate Status'.

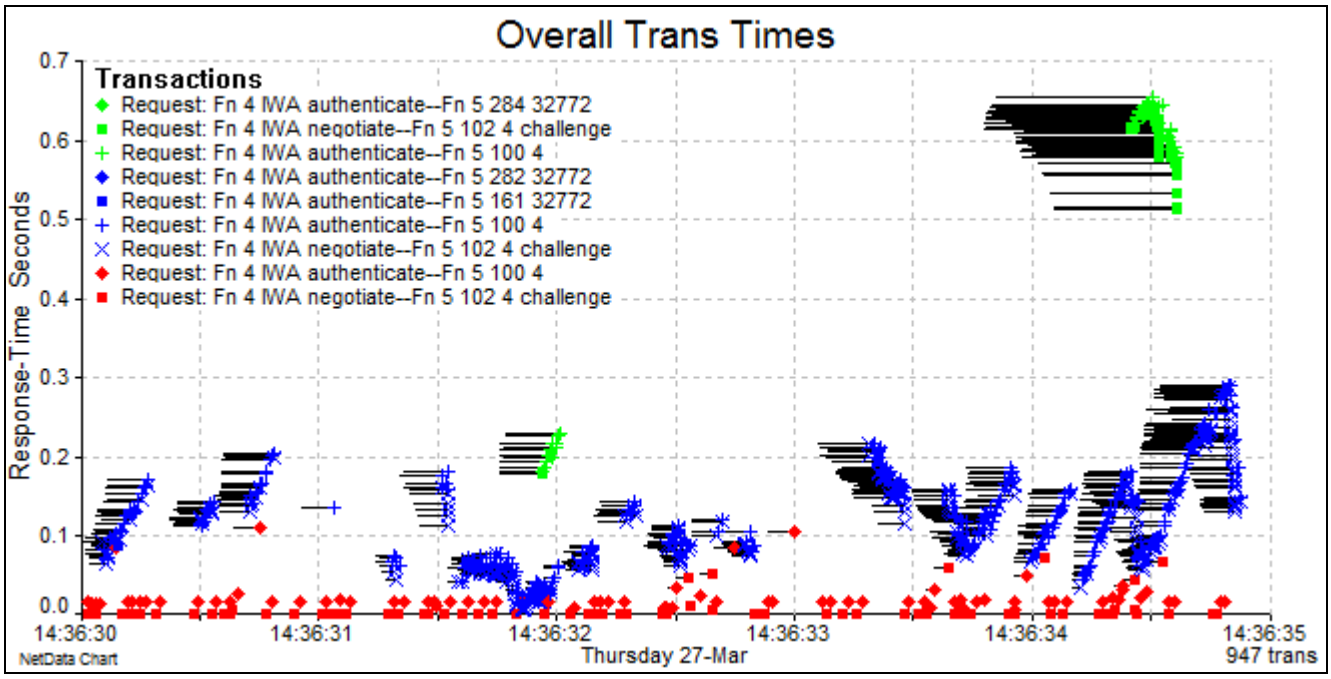
The certificate being checked is identified primarily by its serial number and any ambiguity is avoided by also including hashes of the issuer's distinguished name and public key. The first part of the response indicates whether the responder was able to handle the request successfully, and the status of each listed certificate – good, revoked or unknown – is indicated by a field immediately following the certificate's identification.


Category: GET
Request Signature: Online Certificate Status
Length: 283 bytes
Frame: 2883
Header:
GET http://evsecure-ocsp.verisign.com/MFEwTzBNMEswSTAJBgUrDgMCGgUABBRFp9TUB3UaX73tHhkd
OCSP Request
Requests
certificate ID
hash algorithm 1.3.14.3.2.26 (SHA-1 hash)
Nul
issuer name hash[20]:45A7 D4D4 0775 1A5F BDED 1E19 1D4A 8DEC 52D2 4155h
issuer key hash[20]:FC8A 50BA 9EB9 255A 7B55 854F 9500 638F E958 6B43h
serial number [16] 7215C1AD84D56C385769ADBD31D96FBAh
Accept: */*
User-Agent: Microsoft-CryptoAPI/6.1
Proxy-Connection: Keep-Alive
Host: evsecure-ocsp.verisign.com
Response Signature: 200
Length: 1,983 bytes
Frame: 2895
Header:
Body: certStatusResp.asn (application/ocsp-response)
response status successful (0)
Response
type 1.3.6.1.5.5.7.48.1.1 (Basic OCSP Response)
data [1953]: embedded ASN.1
Basic Response
Response data
version v1 (0)
responder ID 2.5.4.6 (countryName)
produced 2014-03-23 06:28:18 (Z) [20140323062818]
Responses
certificate ID
certificate state good (0)
this update 2014-03-23 06:28:18 (Z) [20140323062818]
next update 2014-03-30 06:28:18 (Z) [20140330062818]
signature algorithm 1.2.840.113549.1.1.5 (SHA-1 with RSA encryption)
signature value(2048b): 606ED92505807C9A36791E537C0268256EDD89C4C88BAC873DBC1CE5
certificates

11.31 BlueCoat Authentication and Authorization Agent (BCAAA)

NetData decodes authentication transactions of the BlueCoat Authentication and Authorization Agent (BCAAA). These transactions are multiplexed over TCP connections that use the default port number 16101.

Authentication strings are coded in base-64 for transmission as text in transaction messages. NetData converts the base-64 texts to their original strings, and decodes them further if they use NTLMSSP (NT LAN Manager Security Support Provider). The host name is extracted from challenge messages, and user details are extracted from authenticate messages, for display in the data column of the transaction table.



This chart reveals signs of stress in BlueCoat authentication agents arising from bursts of authentication requests referring to the same user, and identifies what are incomplete or invalid authenticate messages because they produced abnormal responses such as ‘Fn 5 284 32772’.

Each authentication requires two round-trips – the server (agent) issues a challenge in the response of the first trip, and receives the responding authenticate message in the second trip. The server relates the trips by an 8-byte sequence number at the start of the relevant messages. NetData displays part of these numbers, in hex, in the data column of the transaction table. It also characterises the reverse trips, from server challenge to client response, as Reverse transactions.

The 8-byte challenge word is displayed in the UserID column of the transaction table, to help correlate front-end and backend authentication transactions.

11.32 Citrix NetScaler Metric Exchange Protocol

The Metric Exchange Protocol (MEP) is used by Citrix NetScaler appliances to exchange site metrics, network metrics, and persistence information to other sites participating in Global Server Load Balancing (GSLB). MEP traffic and command propagation is handled by TCP port 3011 in clear, or port 3009 when using SSL. NetData recognises this traffic and measures the response times of site-metric polling transactions that are initiated at one-second intervals by both the client and the server. Persistence and network-metric information is exchanged with messages ‘pushed’ at 5-second intervals.

11.33 Citrix NetScaler High-Availability Services

NetData measures the response times of transactions involved in synchronising configurations for Citrix NetScaler high-availability systems. This traffic normally uses TCP port 3008 when using SSL and port 3010 in clear. It has also been seen using port numbers 5000, 5001 and 5002.

11.34 TACACS+

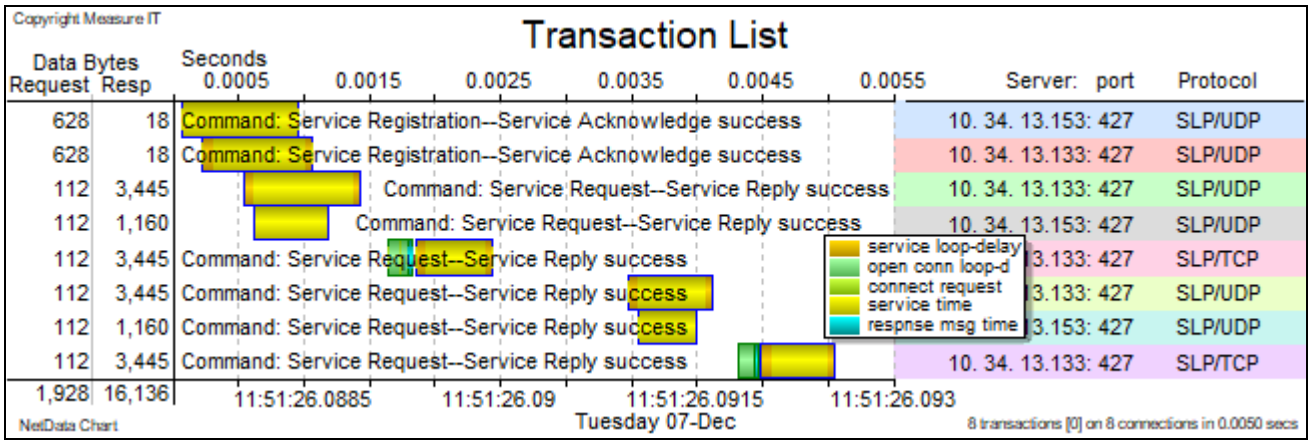
The original Terminal Access Controller Access-Control System (TACACS) was developed by BBN (Bolt, Beranek and Newman) and assigned port 49. Cisco's Plus version, TACACS+, is a quite different system although it also uses port 49 and provides functions for authentication, authorisation and accounting. Message bodies are normally encrypted but NetData decodes the 12-byte headers and measures response times.

11.35 Novell Service Location Protocol (SLP)

The Novell Service Location Protocol (SLP) provides a scalable framework for the discovery and selection of network services. It largely removes the need for static configuration in large and complex systems of services. Its primary specification is found in RFC 2608, and vendor extensions are discussed further in RFC 3224.

The protocol runs on either TCP or UDP and uses port 427. Many requests and responses may run concurrently because all messages except multicast advertisements have a transaction ID that relates responses to their requests and accommodates retransmissions of UDP requests when they don't receive a response.

NetData fully parses all eleven types of messages. The waterfall chart below illustrates a burst of Service Registration and Service Request transactions from one client to two servers, using both UDP and TCP: The TCP transactions are preceded by connection setups.



12 Naming Systems

12.1 Link-Local Multicast Name Resolution (LLMNR)

NetData detects and decodes Link-Local Multicast Name Resolution (LLMNR) queries (RFC 4795) using UDP or TCP port 5355. Queries may be unicast, or multicast to 224.0.0.252

LLMNR messages use the same format as DNS queries (RFC 1035).

12.2 Multicast DNS

The Multicast DNS decoder respects the differences with unicast DNS. From the IETF Internet-Draft:

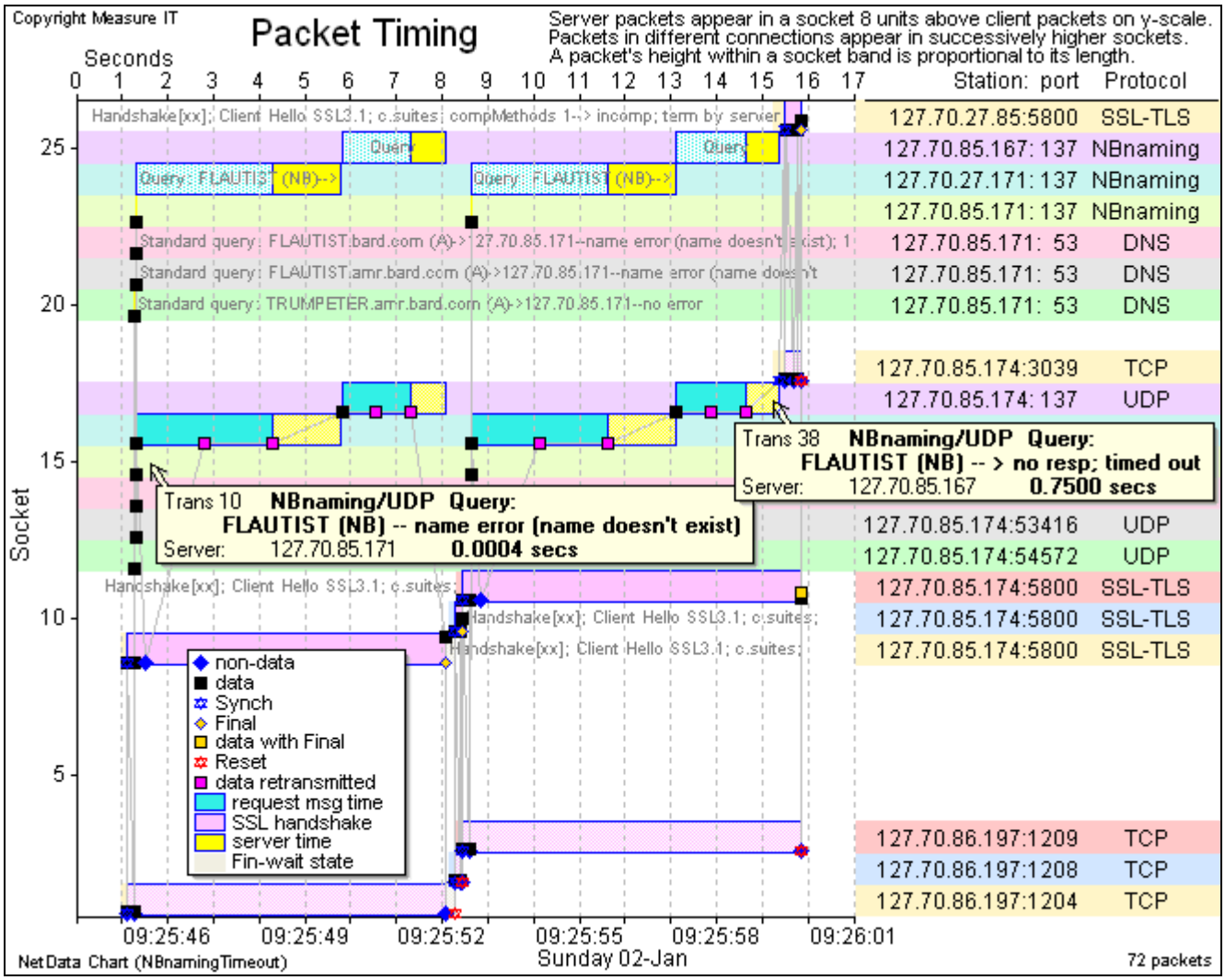
> *Multicast DNS (mDNS) provides the ability to perform DNS-like operations on the local link in the absence of any conventional unicast DNS server. In addition, mDNS designates a portion of the DNS namespace to be free for local use, without the need to pay any annual fee, and without the need to set up delegations or otherwise configure a conventional DNS server to answer for those names.*

NetData displays the mDNS flag that in a query record indicates that a unicast response is requested (making it a QU rather than a QM query), and in a resource record indicates that the name cache is to be flushed. One advantage of multicasting is that many devices on the local network can be updated with few packets, and unsolicited responses are sometimes sent to announce changes. Requests can include many query records and known answer records, while responses don't include query records. NetData assumes that requests and responses for the one device are not pipelined (as occurs with unicast queries when the server is slow), and tags repeated queries or responses as retransmissions only when their contents are identical.

12.3 NetBIOS Name Resolution

NetData's decoder for the NetBIOS (NB) name-resolution protocol has been revised and now characterises queries and responses as transactions. When there is no response to a query NetData measures the timeout between successive retransmissions and assumes that the query transaction terminates at the end of a timeout after the last retransmission.

Timed-out queries are illustrated in the following chart of a failing application. Two attempts to establish a secure session (7-second pink bars at bottom of chart) were frustrated by the application server's inability to resolve the name of a backend server. After trying DNS the server issued NB-naming queries. One name service indicated that the name didn't exist, and another service didn't respond. The query was repeated twice at 1.5-second intervals, causing a total delay of 4.5 seconds, and then the query was broadcast three times at intervals of 0.75 seconds.



Although the chart appears cluttered it reveals many aspects of the application failure: the succession of attempts to resolve a name in different ways, and query retransmissions without response. Because the backend query bars account for virtually all of each front-end transaction's time this one chart characterises the problem's key elements.

NetData now decodes NB name queries as thoroughly as for DNS queries.

12.4 JBOSS HA Pooled Invoker (Port 4446) and RMI Naming Service (1099)

NetData characterises and measures JBOSS naming-service and invoker transactions. The messages of both protocols take the form of serialised Java objects, and NetData extracts the names of significant objects to server in transaction signatures.

13 Security

13.1 LDAP Logical Expressions

The search filter of an LDAP search request is specified with the Basic Encoding Rules (BER) of ASN.1 (Abstract Syntax Notation) and NetData displays the filter within the tree structure of the transaction's request and response ASN messages.

Then tree structure of the search filter becomes more readable when converted to a conventional logical expression and included in the transaction's key-data field. The logical expression is then generalised by replacing constant values with asterisks for inclusion in the transaction's request signature to define the transaction's type.

The tree structure presents the logical operators AND and OR in the form of function calls as in the following example:

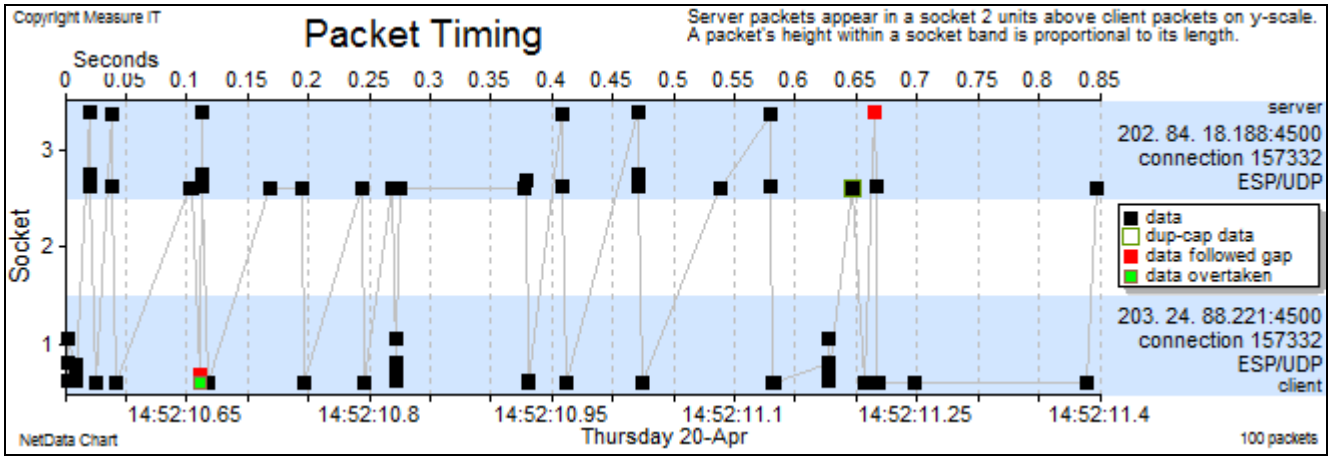
```
AND (objectclass==ironRole,
     OR (AND (NOT Present:ironHost,
              cn==defaults),
         ironHost==ALL,
         ironHost==us-grr-bkx003))
```

```
Protocols:                  Lightweight Directory Access Protocol (LDAP)
                            / Transmission Control Protocol
Key data:                   DC=AP,DC=ironblade,DC=net ((objectclass==ironRole) AND ((NOT
    (Present:ironHost) AND (cn==defaults)) OR (ironHost==ALL) OR (ironHost==us-grr-bkx003)));
    objectClass; cn; ironCommand; ironHost; ironUser; ironOption; ironRunAs; ironRunAsUser;
    ironRunAsGroup; ironNotBefore; ironNotAfter; ironOrder; uSNChanged
Category:                   LDAP
Request     Signature:      SearchRequest DC=* DC=* DC=* ((objectclass==*) AND ((NOT
                            (Present:ironHost) AND (cn==*)) OR (ironHost==*) OR
                            (ironHost==*)))  objectClass  cn  ironCommand  ironHost  ironUser
                            ironOption  ironRunAs  ironRunAsUser  ironRunAsGroup
                            ironNotBefore  ironNotAfter  ironOrder  uSNChanged
            Length:         545 bytes
            Frame:          1
   LDAPmsg
      ID#                   14
      SearchRequest
         baseObject         DC=AP,DC=ironblade,DC=net
         scope              wholeSubtree (2)
         derefAliases       never (0)
         sizeLimit          0
         timeLimit          0
         typesOnly          False
         AND
            EqualityMatch ==
               type              objectclass
               value             ironRole
            OR
               AND
                  NOT
                     Present:          ironHost
                  EqualityMatch ==
                     type              cn
                     value             defaults
               EqualityMatch ==
                  type              ironHost
                  value             ALL
               EqualityMatch ==
                  type              ironHost
                  value             us-grr-bkx003
```

13.2 Internet Protocol Security (IPsec)

NetData's IPsec decoder has been upgraded for both the Authentication Header (AH) and Encapsulating Security Payload (ESP) protocols. NetData splits AH content between the Network Layer Options column (next IP protocol, header length and authentication data) and the Context column (SPI and sequence number). The ESP SPI (Security Parameter Index) is also displayed in the Context column, but the ESP sequence number is displayed in the Data Sequence column – normally occupied by TCP sequence numbers.

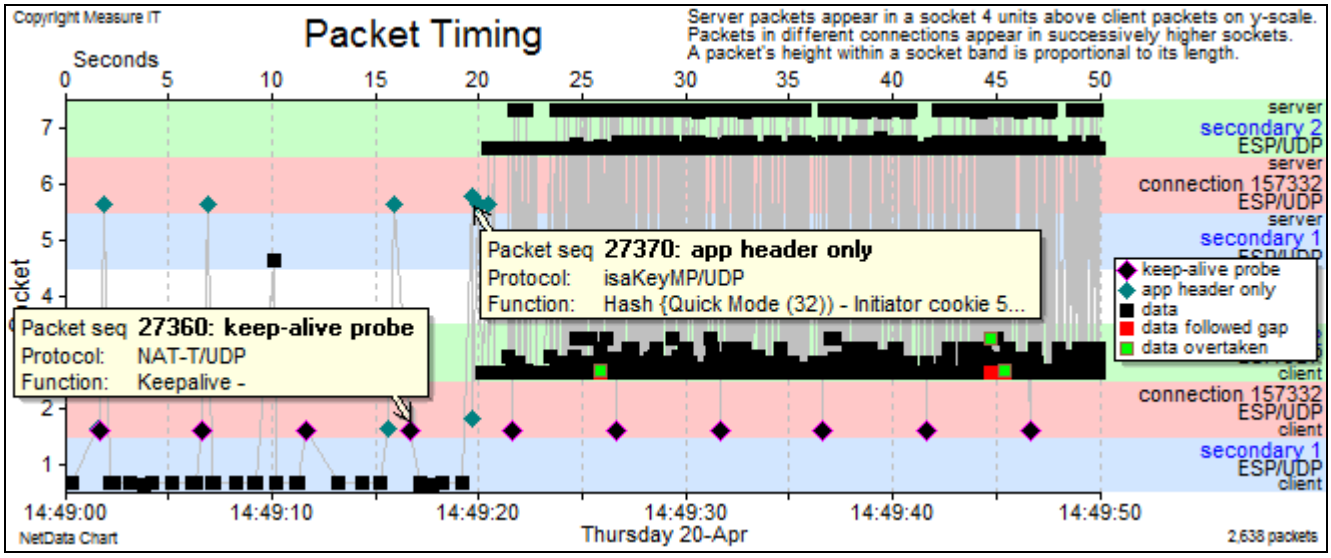
NetData tracks ESP sequence numbers with a notional 64-frame window, flagging packets arriving out of order and leaving a sequence gap (marked on the timing chart by red squares), packets overtaken and filling a gap (green squares), and packets with repeated sequence numbers (open dark-green squares).



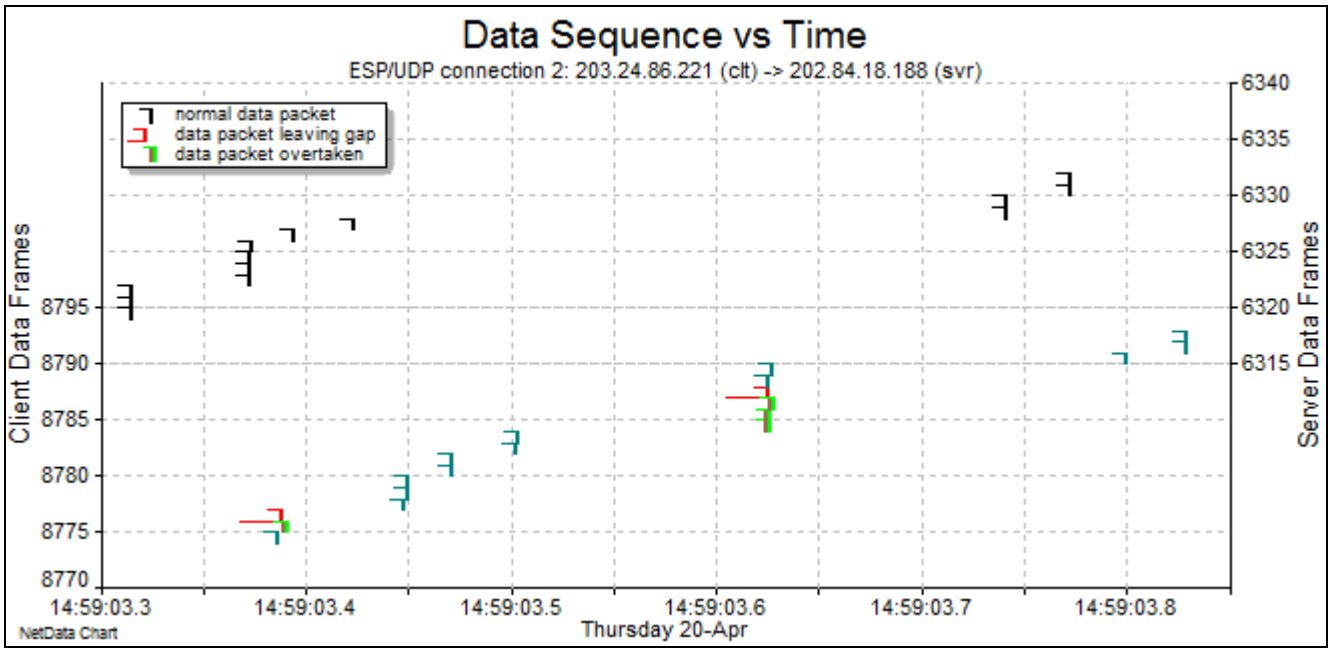
This chart of NAT-T traffic confirms that packets from the server were occasionally lost, leaving an ESP sequence gap that wasn't filled (although the lost data would have been recovered by TCP in packets with different sequence numbers). The red markers for some client packets also indicate sequence gaps, but the charts showed that all the gaps were quickly filled by packets marked with green squares – in other words, packets were not lost between the client and sniffer, but their order had occasionally changed.

Packets with ESP sequence numbers can be displayed on the data-sequence chart. The vertical strips representing packets are plotted in the usual way against the time-of-day scale at the bottom and the sequence-number scale at the left. Strip colours highlight the occurrence of sequence gaps and gap-filling packets. Although TCP headers can't be seen in the ESP tunnel mode, TCP is still likely to be the transport protocol that controls flow and recovers lost data. Unfortunately, network devices along the path can see only the outer IP header and an optional UDP header (added for the NAT-T protocol) and must regard the packets as connectionless. Networks with alternative paths are therefore more likely to deliver IPsec packets out of order, and the consequent duplicate acks and send-window halving may degrade throughput. Patterns of overtaken packets, overlaid with a data-throughput graph, help to characterise such performance problems.

An IPsec ESP tunnel-mode connection may be split into several sub-connections identified by their SPIs. NetData assigns to each sub-connection a secondary connection ID that indicates the order in which different SPIs are found in the traffic. For example, packets using the second pair of SPIs are assigned a secondary connection ID of 2. The packets of an individual IPsec stream can be separated into their distinct sub-connections by choosing 'Plot Related Secondary Connections' from the chart's context menu. NAT-T keepalive packets and ISA Key Management packets are not associated with a particular ESP stream and are plotted on the bands of the parent connection (ID 157332 in this case):

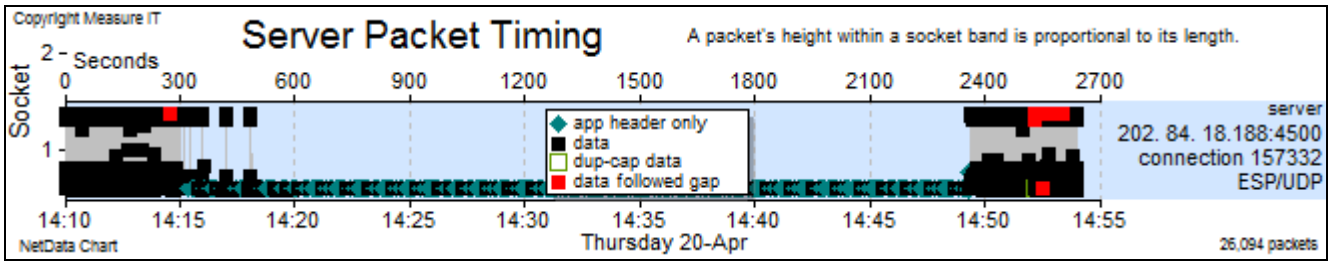


Because packet streams with different SPIs have different ESP sequence numbers, an individual sub-connection should be selected for the data-sequence chart by choosing 'Plot Flow Chart of This Connection' from the timing chart's context menu.

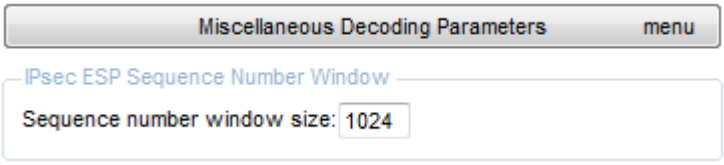


This chart of NAT-T traffic highlights the occurrence of overtaken packets in the streams of packets using the second pair of SPIs.

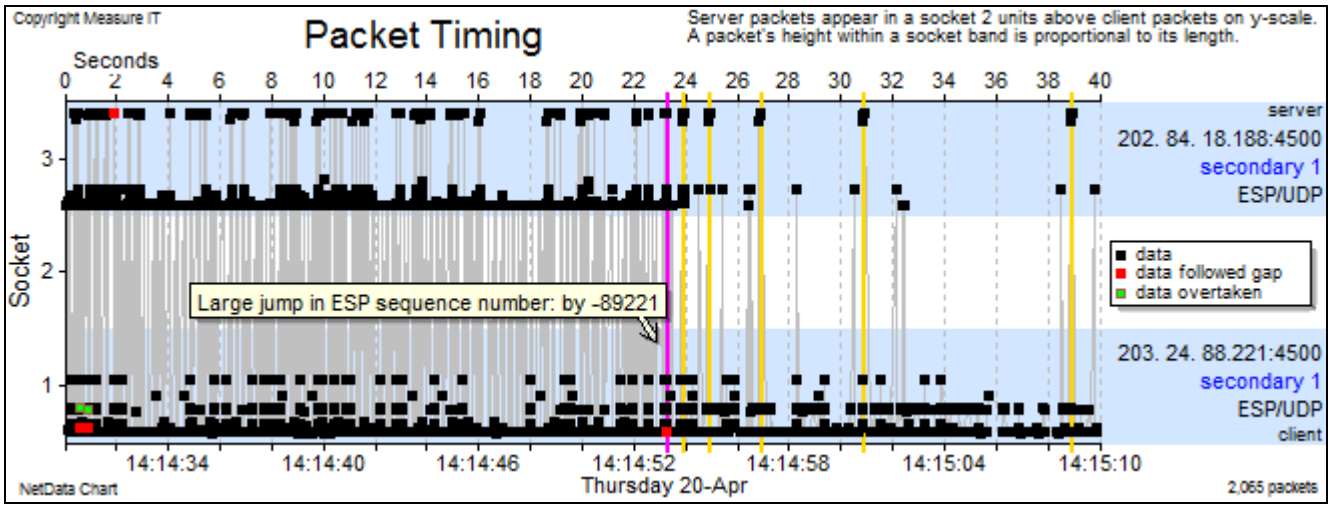
Although the TCP/IP headers and payload are all usually encrypted in IPsec traffic, NetData's tracking of ESP sequence numbers and graphical presentation can shed valuable light on network behaviour that may cause serious performance problems. The charts also provide insight to application chattiness and message lengths. The presence of red squares (sequence gaps) and the absence of green squares (gap fillers) among the server packets indicates server packet loss in this capture:



On the Decoding page of controls, in the menu of miscellaneous parameters, is an option to change NetData's sequence-number window size from the default size of 64. A change would be appropriate if the monitored VPN equipment was configured with a large window size, such as 1024, to avoid erroneous detection of replay attacks.



NetData detects ESP sequence numbers that lie outside the window and records them as network events. When loaded from the database they are plotted automatically on the timing chart as pink vertical stripes, and their occurrence is described by their pop-ups, as in this chart:



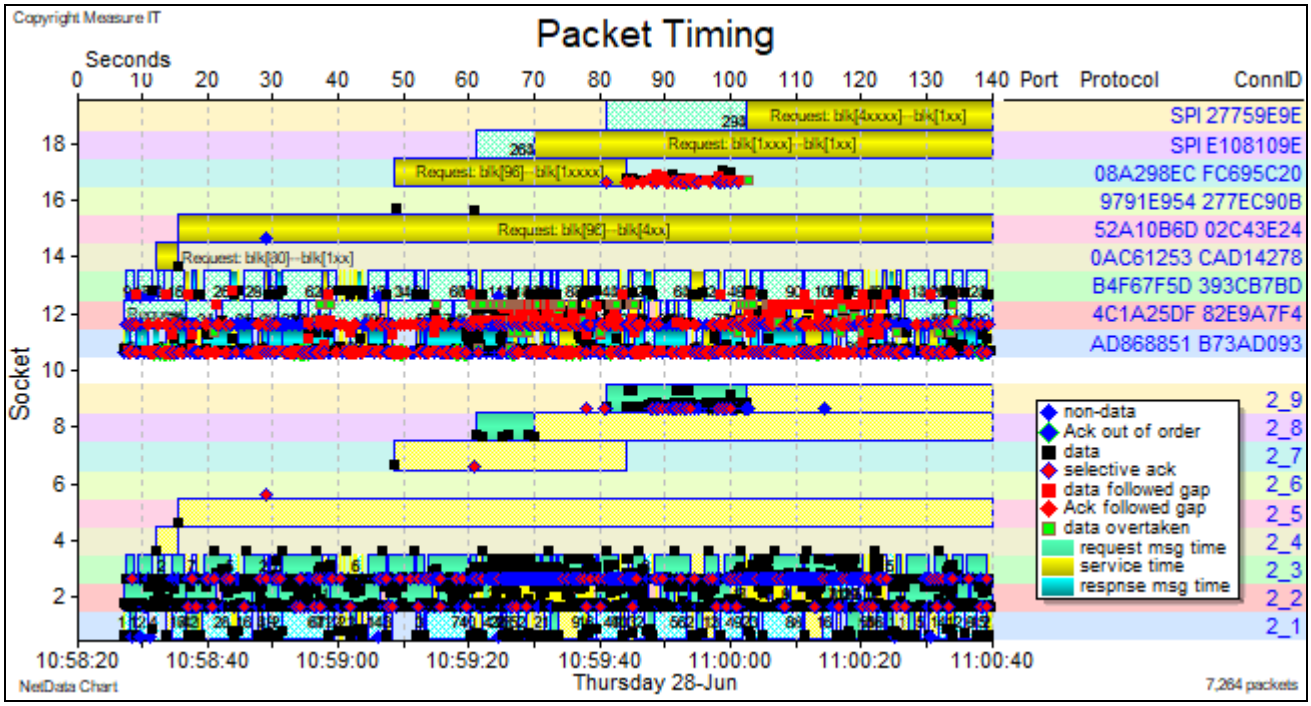
In this case the large sequence number jump *was* treated as a replay attack and the receiving device stopped forwarding client packets to their destination. Confirmation of the resulting break in the underlying TCP connection is provided by the appearance of similar bursts of server packets at time intervals that double in magnitude. This behaviour is consistent with repeated TCP retransmissions after timeouts and is recognised by NetData which records the timeouts as a sequence of network events displayed as orange stripes with optional bands:

13.3 Matching IPsec Data Streams in Multiplexed Connections

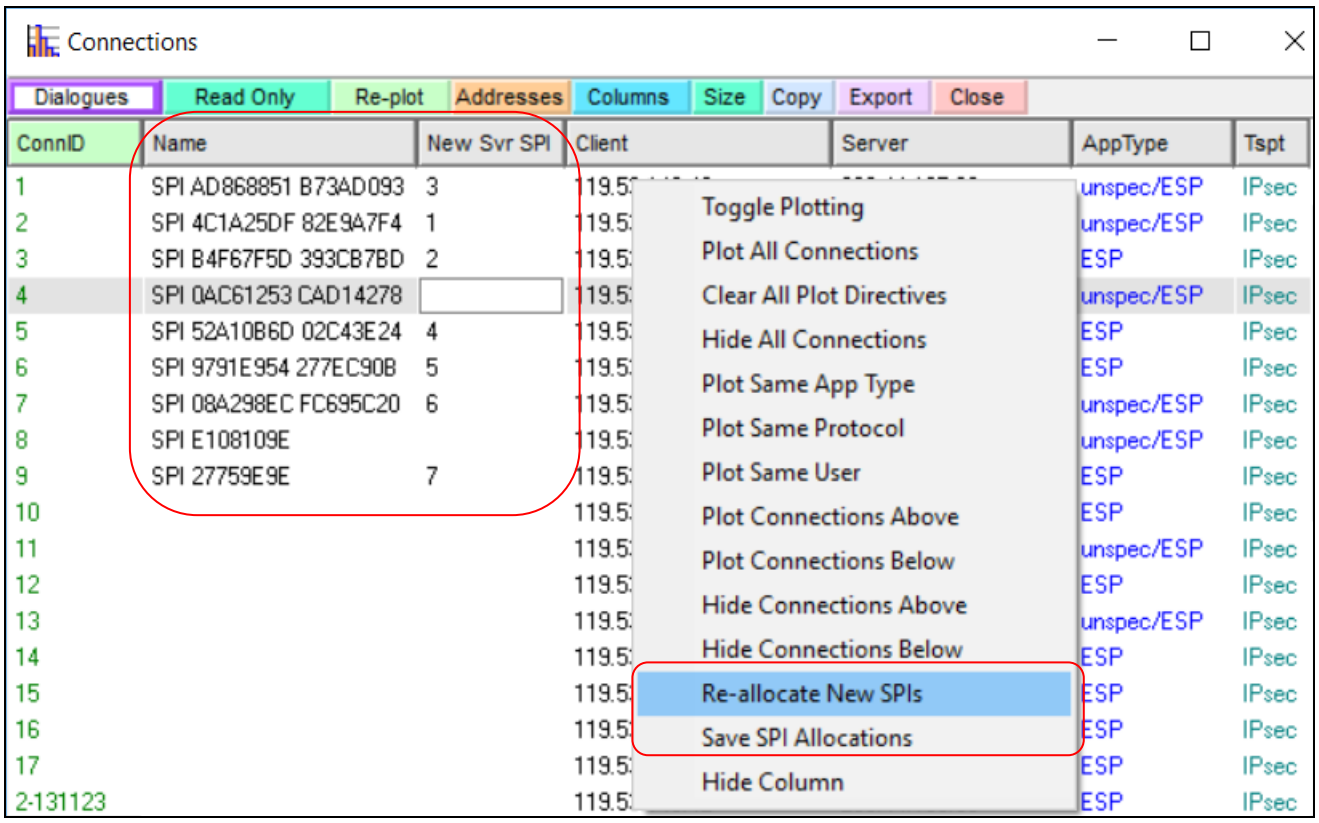
IPsec dialogues using Encapsulating Security Payload (ESP) identify secondary (multiplexed) packet streams with a 4-byte Security Profile Index (SPI), and include a 4-byte sequence number that relates to the SPI. NetData tracks the sequence numbers to highlight sequence gaps and gap-filling packets to indicate packets that were probably lost or arrived out of order. NetData also attempts to categorise packets as either data packets, acknowledgements (non-data) or Selective Acks according to their lengths and on the assumption that the underlying transport is TCP. Furthermore, it sends data packets to the generic decoder for unspecified traffic (tagged 'unspec') and characterises transactions as request-response pairs on the assumption that there is no further multiplexing within secondary connections. The latter assumption is not correct in the general case, and the characterised transactions may be invalid.

In the absence of significant multiplexing in secondary connections, NetData's charts can reveal performance issues. However, NetData has no way of deciding how client and server streams should be associated in pairs to define secondary connections; NetData simply allocates streams to secondary connections in the order that they appear in the captured traffic; and, if there are many secondary connections, the initial allocations are likely to be incorrect.

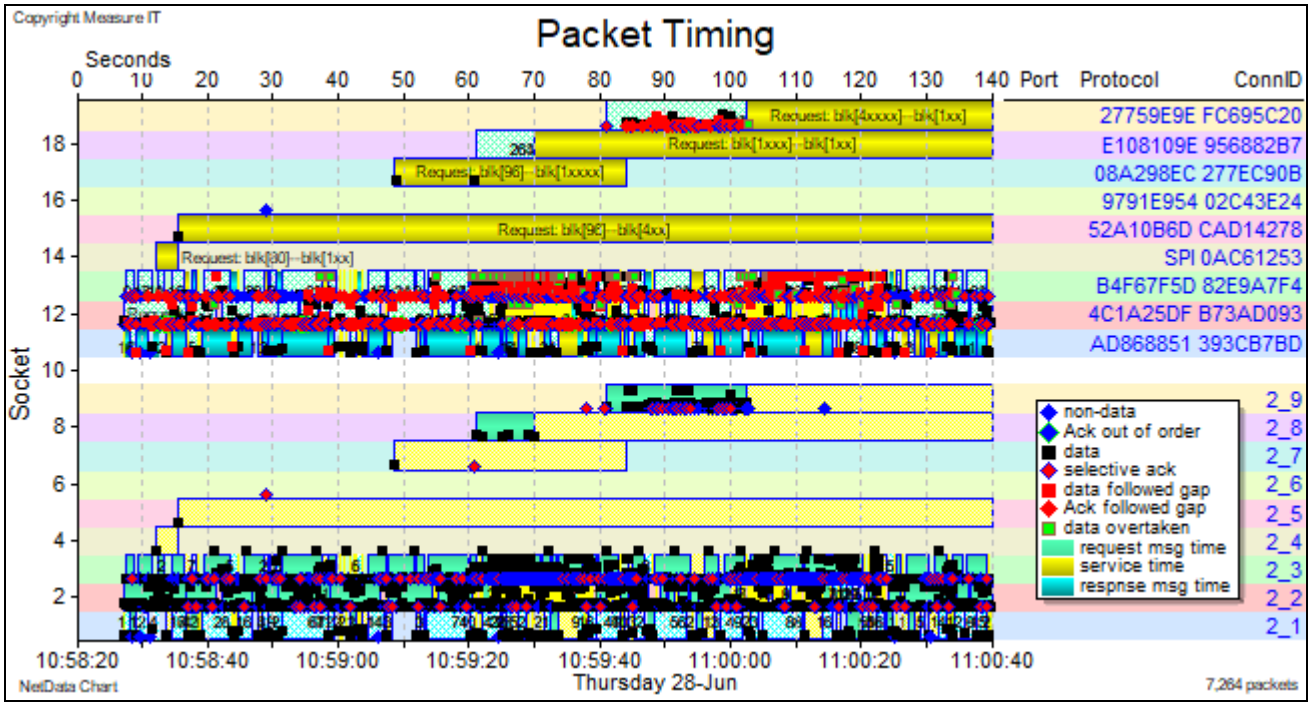
Correct pairing of client and server packet streams requires manual intervention in the connection table that supports the timing chart, as in the following example.



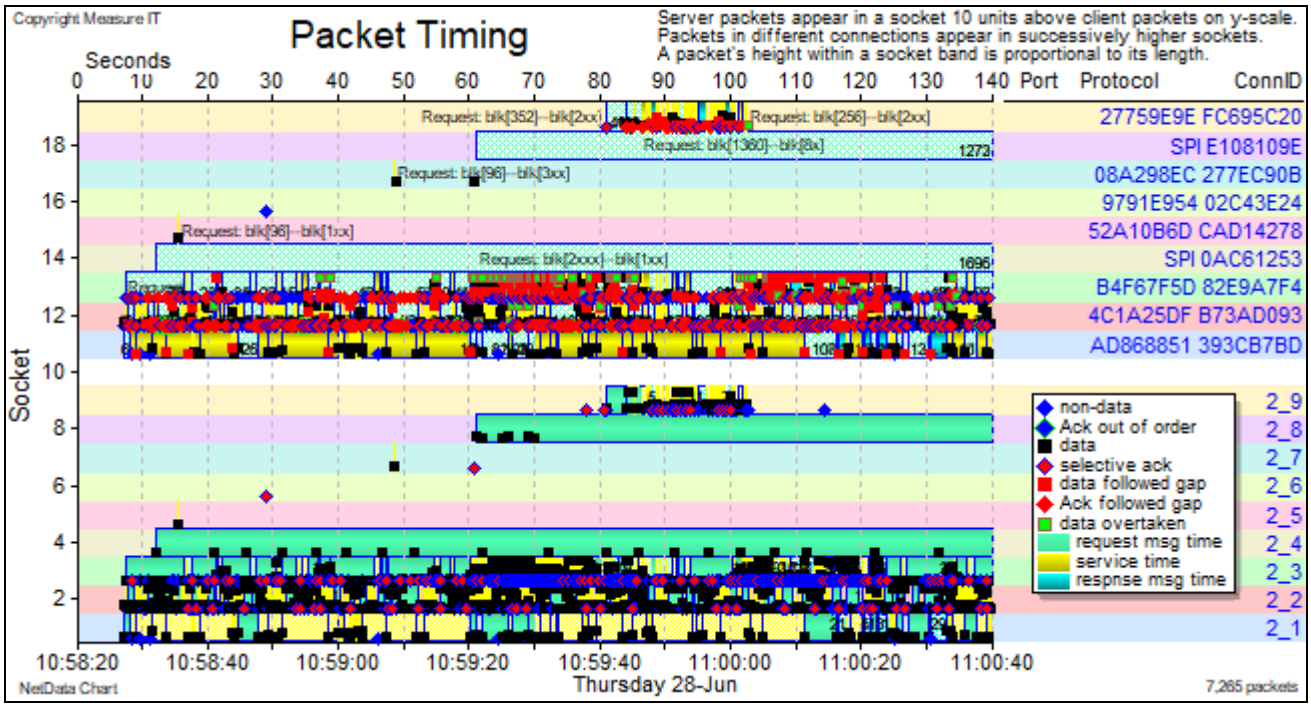
This timing chart from a super project displays nine secondary connections with IDs from 2_1 to 2_9. The blue labels on the server bands contain the client and server SPIs of the secondary connections, in hex format. None of the pairings on this chart are correct. For instance, the burst of server packets on socket band 17 (connection 2_7) clearly matches the burst of client packets on band 9 (connection 2_9). Connection 9 can be corrected by allocating it the server stream from connection 7, and this is specified in the 'New Svr SPI' column of the supporting connection table. In addition to revealing this column, it is useful to reveal the preceding column labelled 'Name' which displays the client and server SPIs of the secondary connections. Re-allocations of server SPIs can be specified by connection IDs or by SPIs in hex format, as in the table below. Any number of changes can be made in a single step.



After specifying new server SPI allocations, choose from the table's context menu to 'Re-allocate New SPIs', and NetData will regenerate the timing chart to show the effect of the changes:

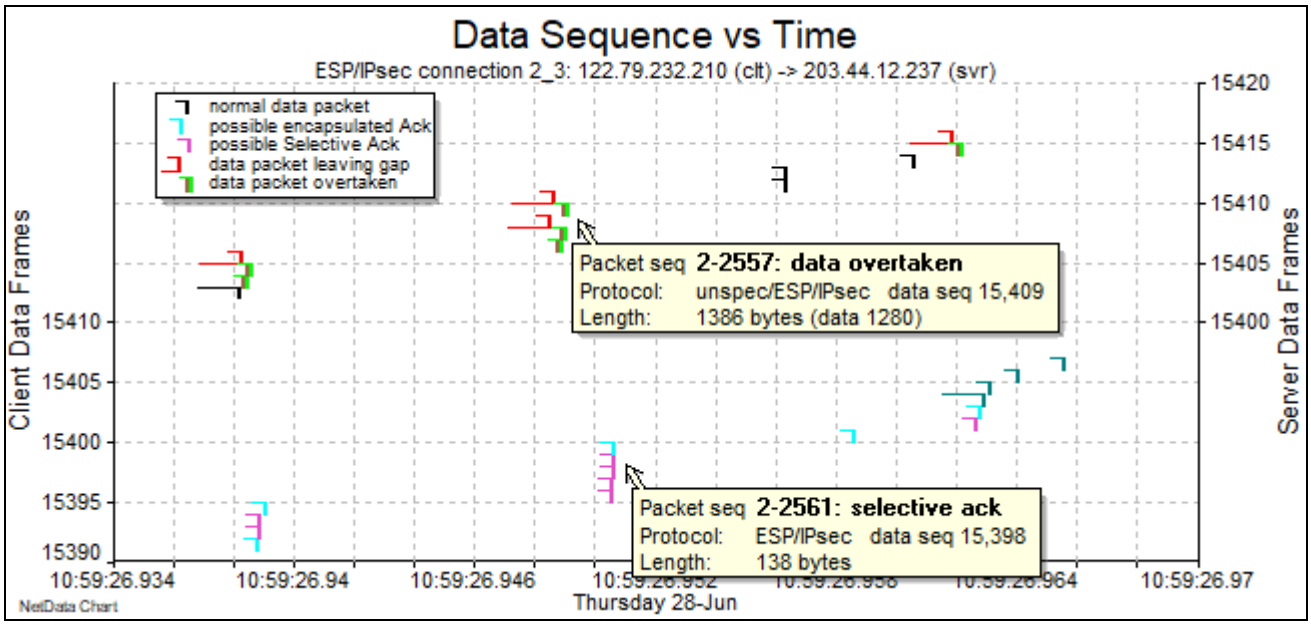


When the SPI pairings appear satisfactory, choose from the context menu to 'Save SPI Allocations'. The new pairings are written to a dedicated text file in the project folder, and the next time the traffic is analysed NetData will adopt the new pairings and characterise transactions in the new secondary connections:



13.3.1 Investigating Throughput Issues with IPsec Traffic

When client and server packet streams are correctly paired in secondary connections, their data-flow charts can be particularly useful in shedding light on why data throughput is abnormally low, as in this chart from secondary connection 3 of the second capture sequence of the above super project:



Six server data packets were overtaken – arrived out of order – and probably generated seven selective acks as indicated by the pink strips. Three of the four selective acks in the centre of the chart could have been regarded by the server as duplicate acks and prompted both a retransmission and halving of its congestion-avoidance window.

13.4 Datagram Transport Layer Security (DTLS)

NetData now decodes Datagram Transport Layer Security (DTLS) traffic. DTLS (RFC 4347 and RFC 6347) is designed to encrypt UDP traffic between applications using a protocol that is as close as possible to TLS 1.1 or 1.2. Because UDP doesn't guarantee delivery of all packets in the correct order, DTLS must use a block cipher that doesn't depend on preceding record blocks.

DTLS record headers include an epoch and sequence number. The epoch is incremented and the sequence number is reset to zero by every change-cipher record. The sequence number serves two purposes, as in IPsec ESP: for the receiver to detect retransmissions, records out of order or records missing; and guard against replay attacks. NetData tracks sequence numbers as it does for ESP: it maintains a window of recent packets to detect retransmissions, gap-filling packets, and invalid sequence numbers that may indicate a replay attack; and it flags packets involved in sequence abnormalities.

The sender has no mechanism for retransmitting data records, but it has timeout and retransmission functions to ensure that all handshake messages are correctly received.

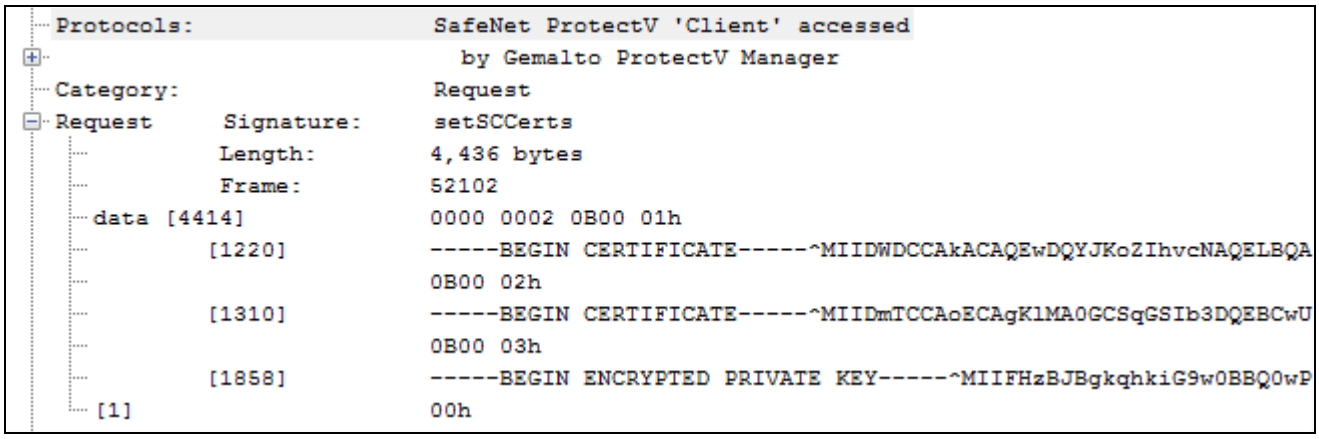
Some handshake messages such as certificate chains can be very long but DTLS does not depend on IP fragmentation. Instead, it provides its own, similar scheme and includes fragment-offset and -length indicators in every handshake-message header. Application data records must be small enough to fit in the largest packet without fragmentation.

The data-sequence (flow) chart plots packet strips according to their sequence numbers, as it does for IPsec ESP packets.

13.5 Gemalto SafeNet ProtectV Manager

It is not practical to give users access to virtual instances of servers in the cloud for pre-boot authentication (PBA), that is, to provide user IDs and passwords to unlock encrypted hard disks and other devices prior to the start of new sessions. Instead, various firms provide key management systems that utilise cloud-provider tools and APIs to handle PBA and other power-up and power-down functions. One such firm is Gemalto, and NetData decodes the transactions of its ProtectV key manager (PVMgr). NetData tags its various types of TLS dialogues: with a PostGreSQL database server (usually using TCP port 5432); from a SOAP API in PV client and user machines (to ProtectV port 8080); between primary and secondary ProtectV Mangers (port 7080) in high-availability systems; and with a SafeNet KeySecure server (port 9000) that handles interactions with the user environment.

NetData decodes transactions from the PVMgr to TCP ports 9090 (in clear) and 9093 (using TLS) of what might be thousands of ProtectV 'clients'. Digital certificates encoded in Base64 are distributed to clients through their port 9090:



NetData decodes a SysLog protocol variant (to UDP ports 514 or 5514) that the PVMgr uses to create an audit trail:

Request Strt	Description	Data
14:09:17.707474	Log Event: <14>- pvm.system...	2017-08-24 04:09:49,560 - Checking peer: vsphere$protectv@50062e59-dd7b-8e2d-d2b7-0213...
14:09:39.223321	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:11,076 - User: m044712, called API: getInstanceGroup, with Parameters: ['tes...
14:09:53.482511	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:25,335 - User: m044712, called API: listInstances, with Parameters: ['vsphere',...
14:10:07.90301	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:39,755 - User: attender, called API: resumeUnattendedReboot, with Paramete...
14:10:10.896064	Log Event: <10>- pvm.system...	2017-08-24 04:10:42,748 - UR has been resumed for the machine vsphere$protectv@898c58a...
14:10:10.958902	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:42,811 - User: m044712, called API: getJobDetails, with Parameters: [['9223']]
14:10:13.672815	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:45,525 - User: l090666, called API: getFirstRunNextStep
14:10:13.733391	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:45,586 - User: l092901, called API: runTcpcheck, with Parameters: ['10.41.13...
14:10:13.980224	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:45,833 - User: ibm, called API: getJobDetails, with Parameters: [['9223']]
14:10:14.043094	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:45,895 - User: ibm, called API: getJobDetails, with Parameters: [['9226']]
14:10:14.2748	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:46,127 - User: ibm, called API: getJobDetails, with Parameters: [['9228']]
14:10:14.335802	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:46,188 - User: m044712, called API: getJobDetails, with Parameters: [['9226']]
14:10:14.397278	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:46,250 - User: m044712, called API: getJobDetails, with Parameters: [['9224']]
14:10:14.453276	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:46,306 - User: m044712, called API: getJobDetails, with Parameters: [['9228']]
14:10:14.594792	Log Event: <14>- pvm.audit--...	2017-08-24 04:10:46,447 - User: l092901, called API: getInstanceStatus, with Parameters: ['vsp...
14:11:14.911123	Log Event: <14>- pvm.audit--...	2017-08-24 04:11:46,763 - User: attender, called API: resumeUnattendedReboot, with Paramete...
14:11:17.981478	Log Event: <10>- pvm.system...	2017-08-24 04:11:49,834 - UR has been resumed for the machine vsphere$protectv@69467f7c...
14:11:18.090177	Log Event: <14>- pvm.audit--...	2017-08-24 04:11:49,942 - User: m044712, called API: getJobDetails, with Parameters: [['9223']]

13.6 IBM Tivoli Directory Server (ITDS)

IBM Tivoli Directory Server is based on LDAP and uses SSL to encrypt its traffic through port 3660. NetData tags such traffic as ITDS and decodes it like other SSL (TLS) traffic.

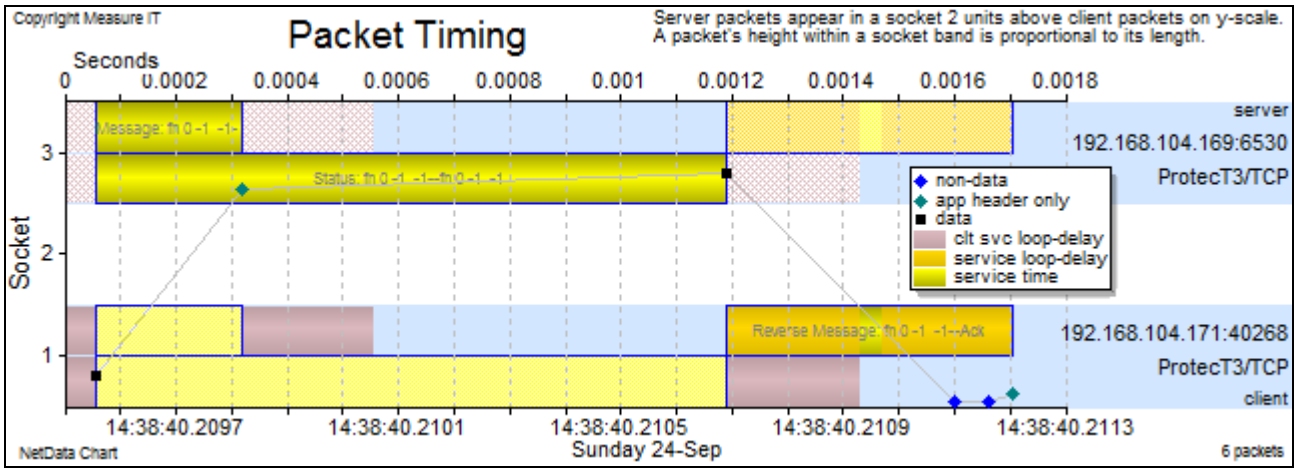
14 Backup Systems

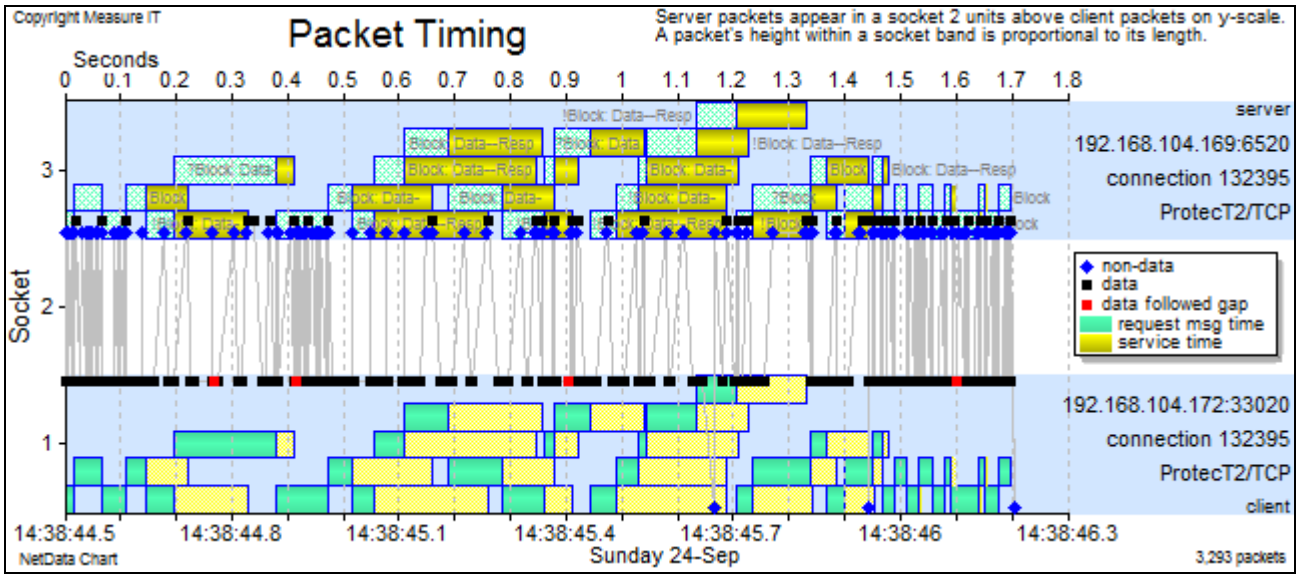
14.1 IBM ProtecTIER System Storage Deduplication and Replication

A family of IBM ProtecTIER appliances provides deduplication and replication solutions, sitting between backup servers that might run, say, NetBackup or Tivoli Storage Manager, and data repositories in the form of disk arrays. A replication grid can be formed by interconnecting ProtecTIER single nodes or two-node clusters at different sites. NetData now decodes ProtecTIER data traffic.

Deduplication and replication services may use ports 6520, 6530, 6540, 6550 and 6560, but NetData has so far been tested only with the traffic of ports 6520 and 6530. Message header formats appear to be the same across the different services, but the traffic patterns are quite different. The traffic of port 6520, tagged by NetData as *ProtecT2*, transfers data in large blocks – of one megabyte, typically – as fast as possible, whereas the traffic of port 6530 (tagged *ProtecT3*) exchanges short messages (typically 396 bytes) at regular intervals of either one second or a tenth of a second. Every data block or message receives an application acknowledgement, and NetData records the time of every round-trip. The resulting charts of response times can reveal evidence of stress in the ProtecTIER appliances.

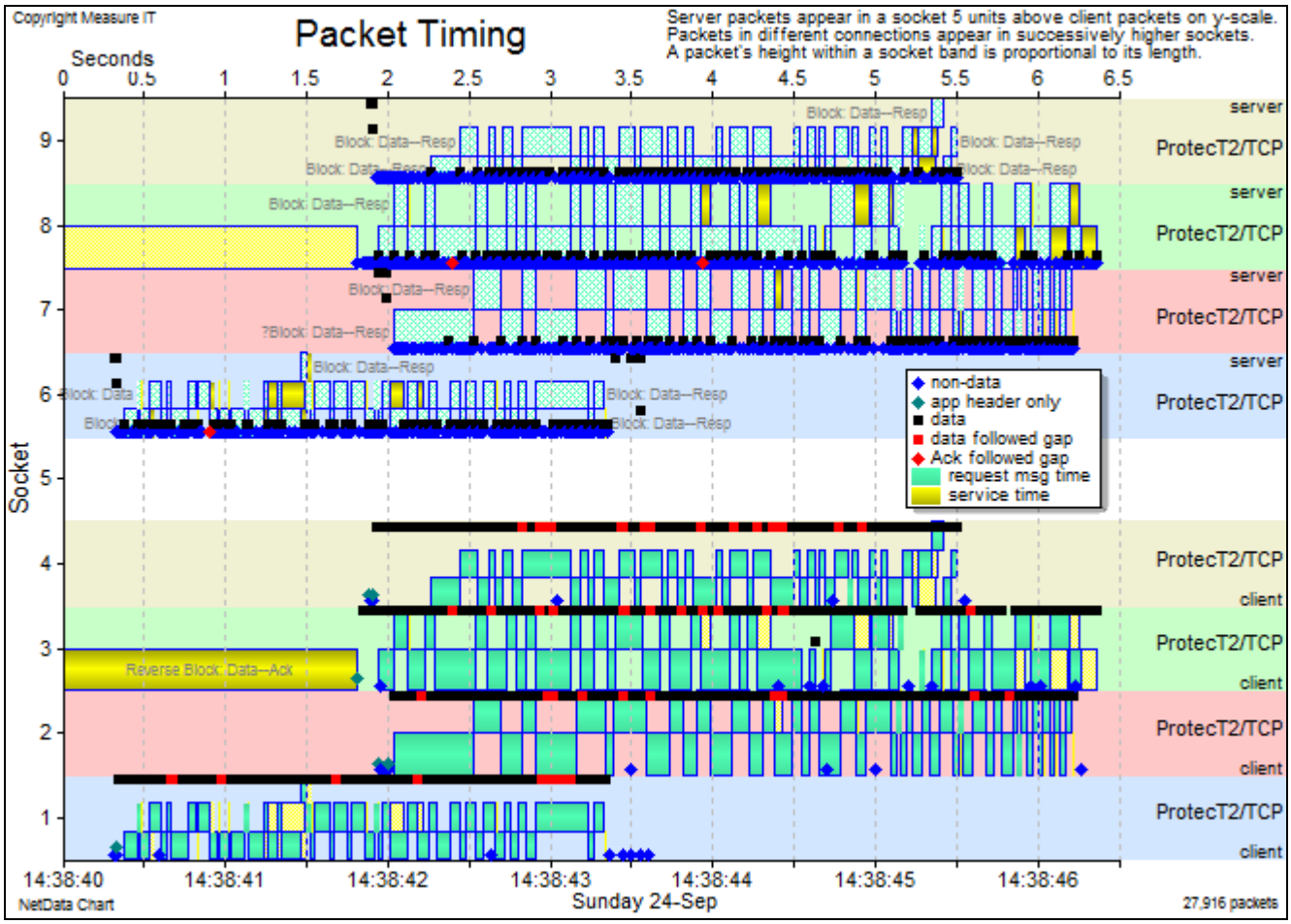
A typical exchange of ProtecT3 messages involves four data packets and three response-time measurements:



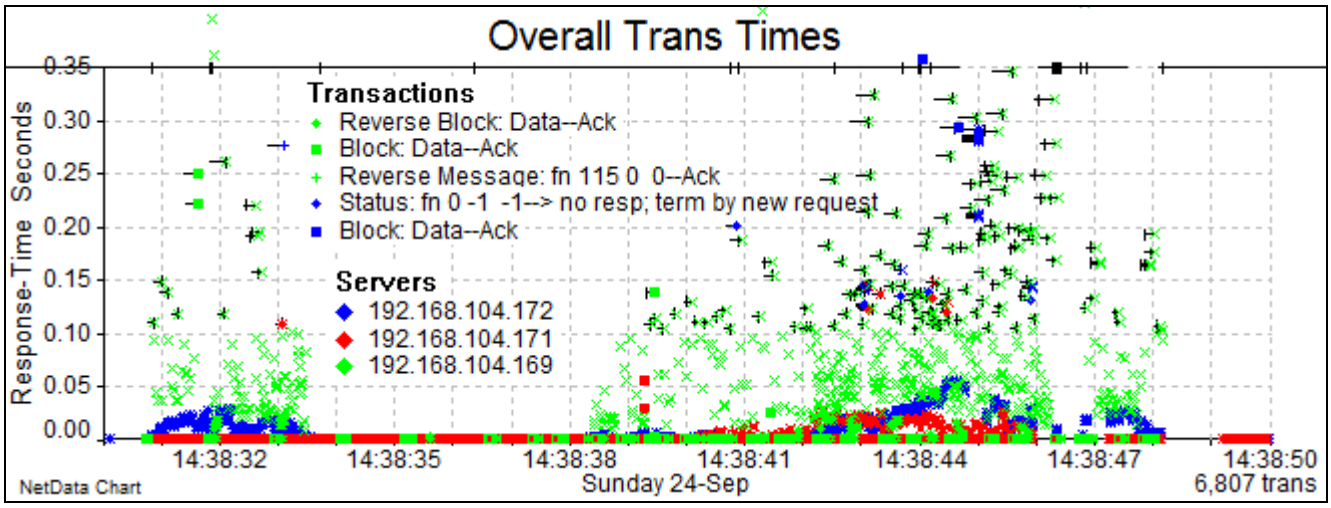


ProtecT2 traffic often involves a continuous stream of one-megabyte data blocks from client to server, as indicated by the blue-green bars on the bottom band of the above chart. Each data block receives a small data response, and the yellow bars indicate the response time between the end of a data block and its response packet (of 136 data bytes). Bars are always stacked at least two high because one response time overlaps the data transfer time of the next block; the higher stack and longer bars in the middle of the chart, however, indicate a period of severe stress in the receiving appliance.

The following chart illustrates some of the connections conveying streams of data between appliances. The ProtecTIER protocol may employ multiple streams to use all the available bandwidth when the throughput of a single connection might be limited by its flow-control parameters. In these circumstances, however, delayed acks and other signs of stress in the timing chart of a single connection may be of no consequence provided the flow is bandwidth limited.



The blue and red rising – or 'wavy' – bands of response-time markers on the following performance chart indicate varying stress in two of the ProtecTIER appliances.



14.2 Acronis

NetData characterises and measures transactions of Acronis True Image Agents, and Acronis Recovery for MS SQL Server and Exchange.

14.3 CommVault

NetData characterises and measures transactions of CommVault backup software.

14.4 Symantec NetBackup Utility and Client Daemons

NetData decodes the traffic of a Symantec NetBackup server that uses port 13724 (utility daemon) on the server for one connection, and port 13782 (client daemon) on the client for another connection. The decoder recognises four different types of connections to port 13724 and can parse backup control messages.

14.5 NetApp snapMirror and snapVault

NetApp (formerly Network Appliance Inc) specialises in storage solutions and its Data ONTAP operating system provides two types of backup service that both use TCP port 10566: snapMirror, a high-availability, lossless replcation system; and snapVault, a backup system that doesn't guarantee to preserve the most recent data. NetData decodes snapMirror and snapVault transactions. It tags the traffic of both systems as 'snpVault'.

15 Print Services

15.1 Common Unix Printer System (CUPS) Browse Protocol

NetData decodes broadcasts by printers announcing their state, capabilities and other information, to and from port 631 using UDP. This printer discovery protocol is known as the Common Unix printer System (CUPS) browse protocol and is widely used by Apple devices. TCP traffic with port 631 is generated by the quite different Internet Printing Protocol (IPP).

15.2 Canon BubbleJet Network Protocol

NetData decodes BJNP messages which use UDP ports between 8611 and 8614.

15.3 NTware UniFlow Printing Management

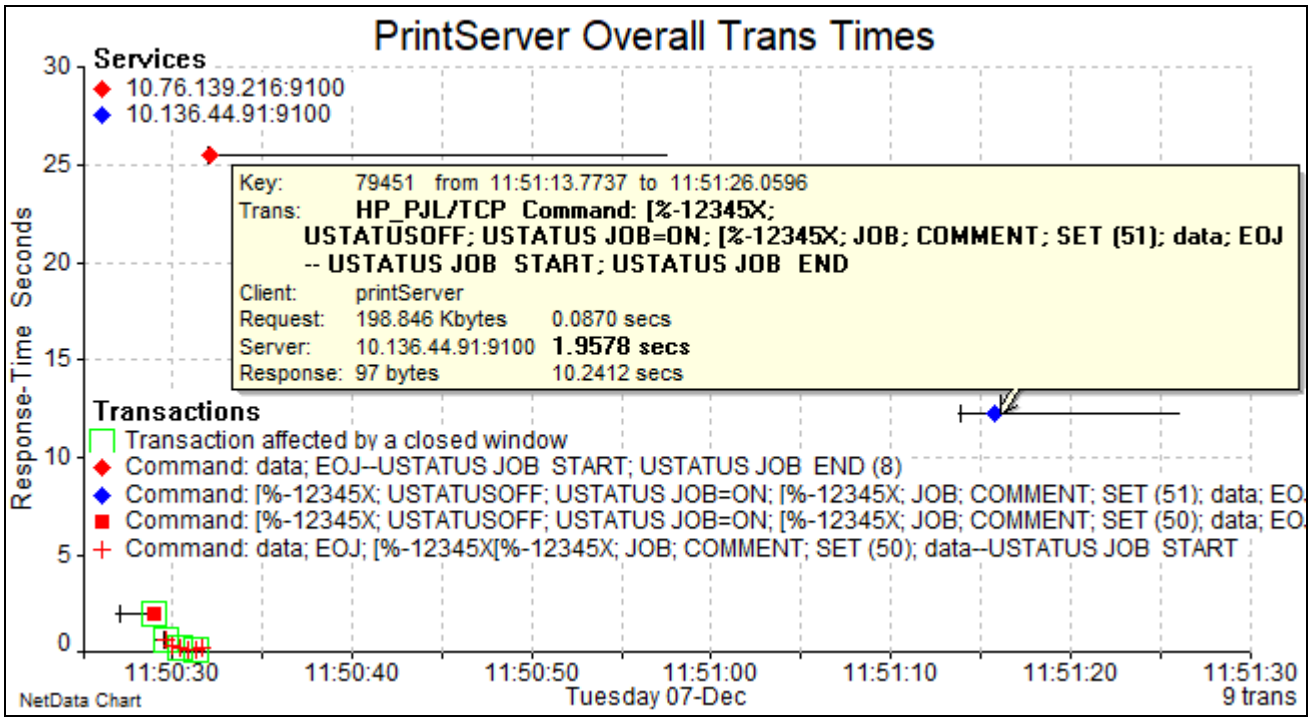
NetData recognises UDP port-mapping transactions of NT-ware's web-based print management system. If it sees a port number being advised to a client then it tags the HTTP traffic through that port as 'uFlow/HTTP'.

15.4 HP Printer Job Language

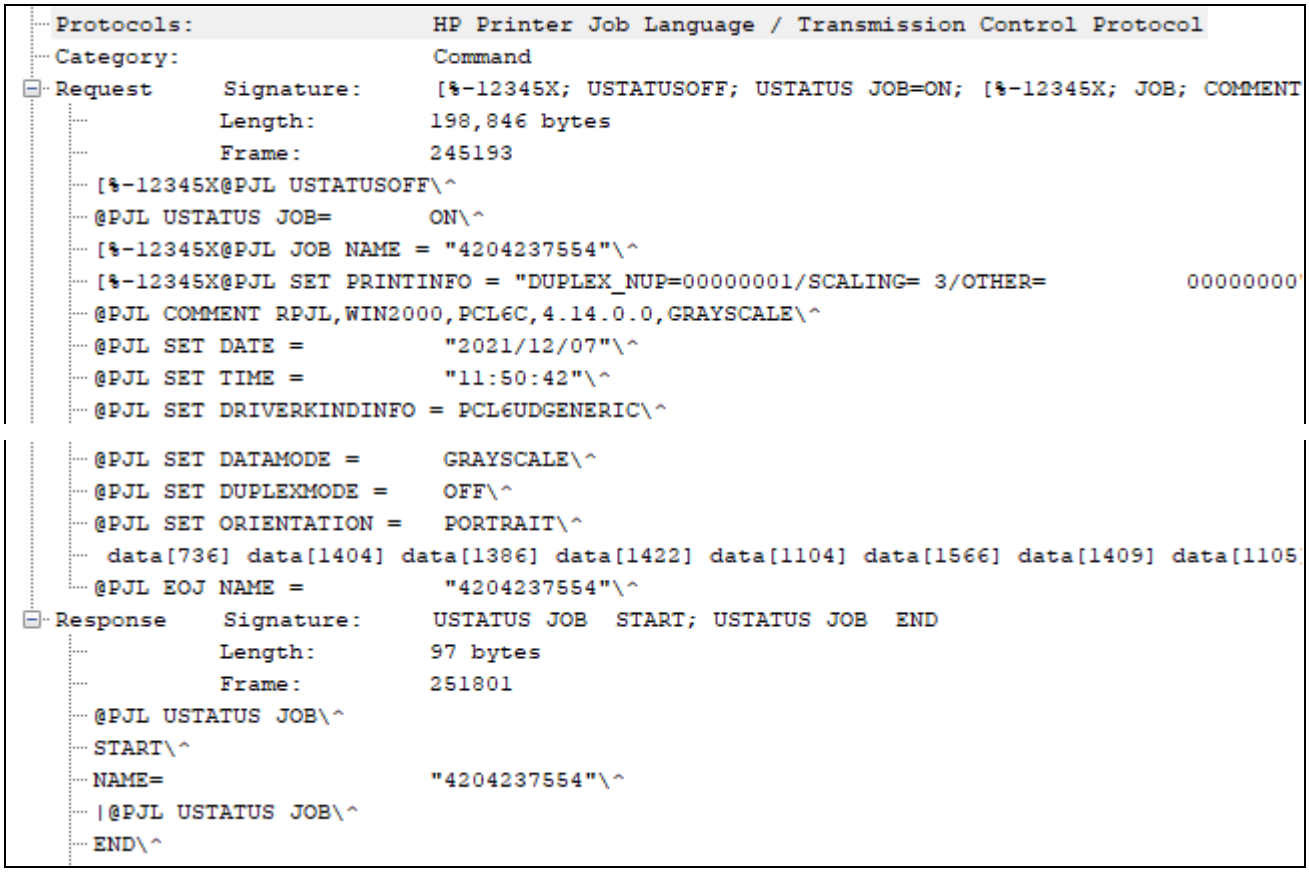
NetData decodes the traffic of printers that use the HP Printer Job Language (PJL). Marks of printer round-trips on the performance and timing charts allow this traffic to be correlated with the preliminary traffic that accesses print spoolers and distributes printer drivers.

The special string of characters [%-12345X in which '[' represent the Escape character is the Universal Exit Language (UEL) command. It allows the printer to alternate between interpreting PJL commands and printer language commands such as PCL and PostScript. All PJL jobs must begin and end with a UEL command.

Every command except UEL ends with a line feed preceded by an optional carriage return. NetData displays these two characters in its usual way as \^. Form feeds are represented by |.



A transaction description lists all the printer commands and responses:



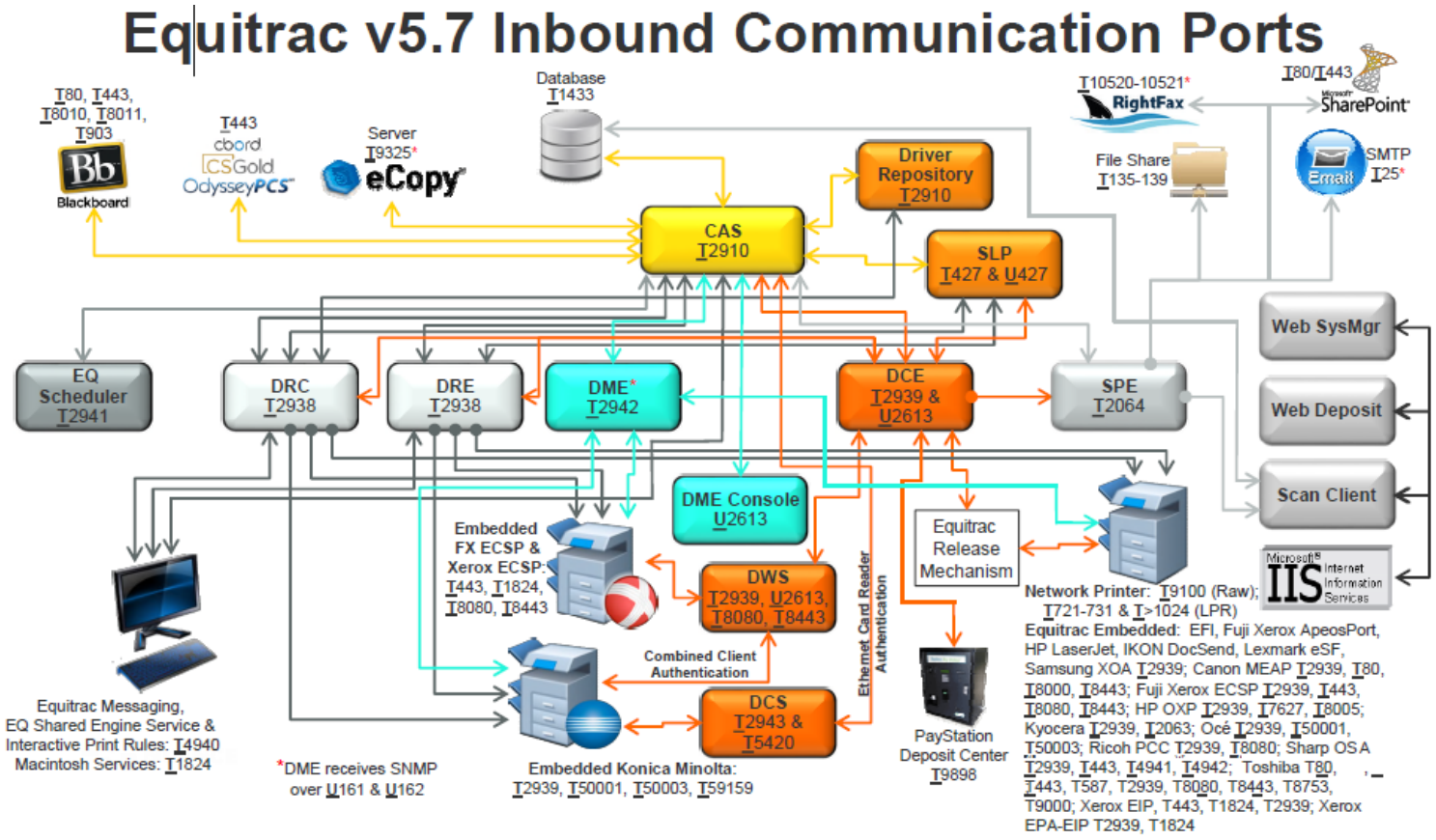
The word ‘data’ indicates an embedded block of print data with a specified length.

15.5 Equitrac Printing Management and Cost Accounting

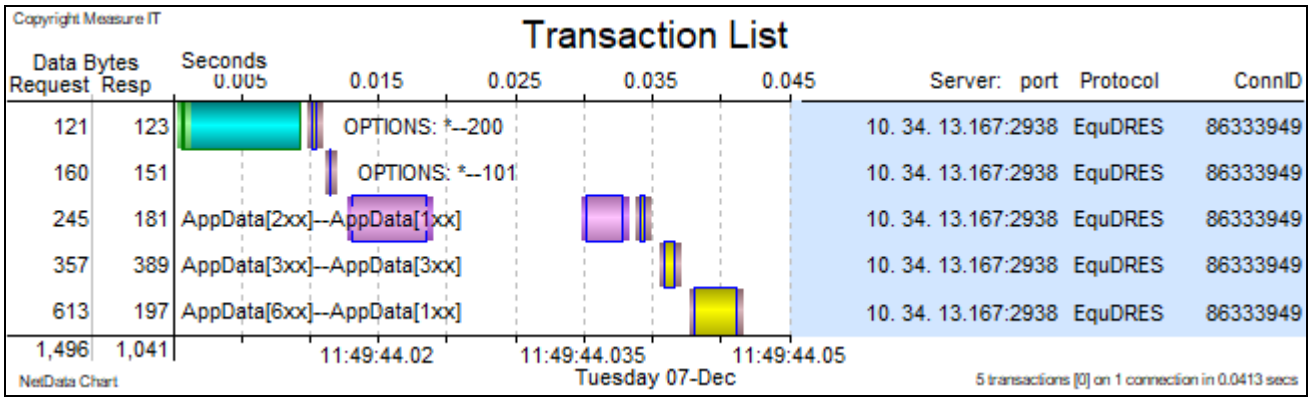
Equitrac – bought by Nuance and then Kofax - is a global company that provides document cost accounting and output management software for printers, copiers and multifunction devices in large enterprises and professional practices.

NetData identifies and labels the TLS traffic accessing various Equitrac services through specified ports:

CAS (Core Accounting Server)	Port 2910
DRE (Document Routing Engine)	Port 2938
DRC (Document Routing Client)	Port 2938
DCE **(Device** Control Engine)	Port 2939
DRE Health Check	Port 6080



When opening a connection with an Equitrac port such as 2938 the client starts with an HTTP Options command to which the server may respond with a request to change to a TLS protocol:

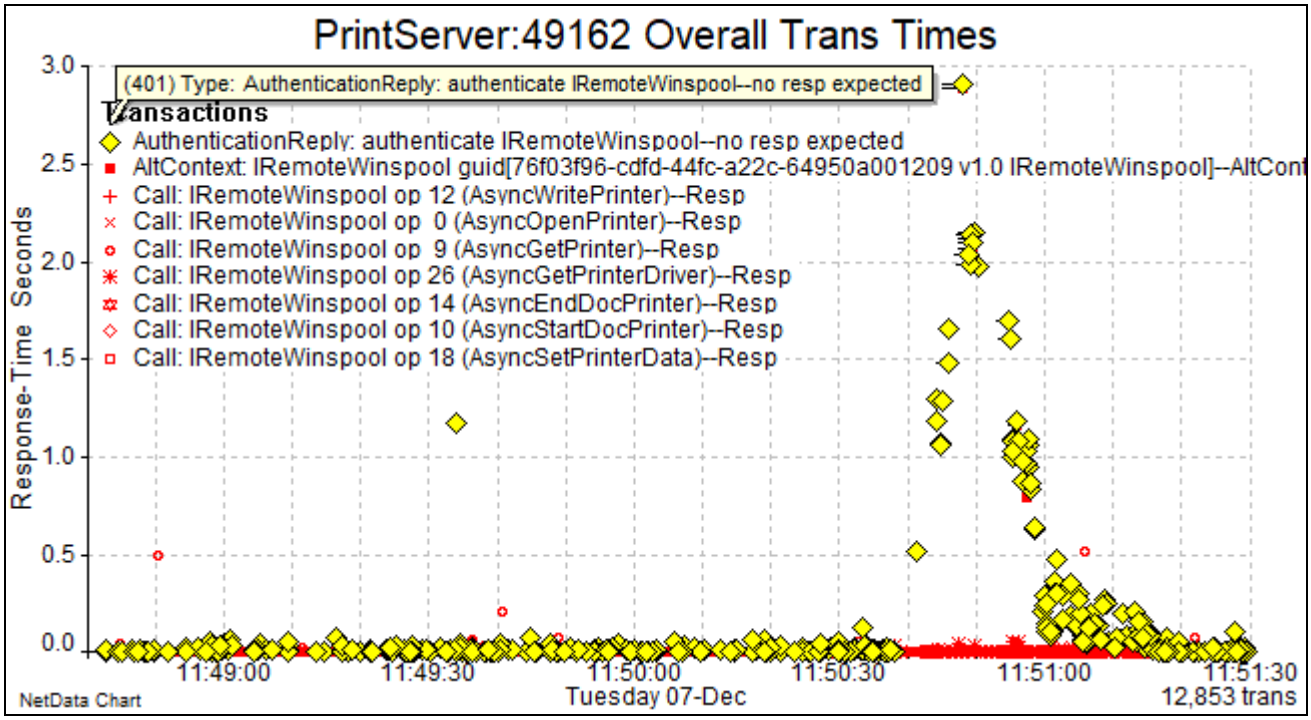


15.6 RPC Access to Spooling Services in Print Servers

Versatile print services such as follow-you printing require central printer servers whose spooler objects are accessed with RPCs (Remote Procedure Calls). NetData recognises the 77 operations of an object labelled *IRemoteWinspool* and ensures that operation names appear in the transaction signatures and in chart pop-ups. NetData fully parses the messages of Bind and AltContext transactions, including all three phases of authentications when they occur, but doesn't fully parse operation messages.

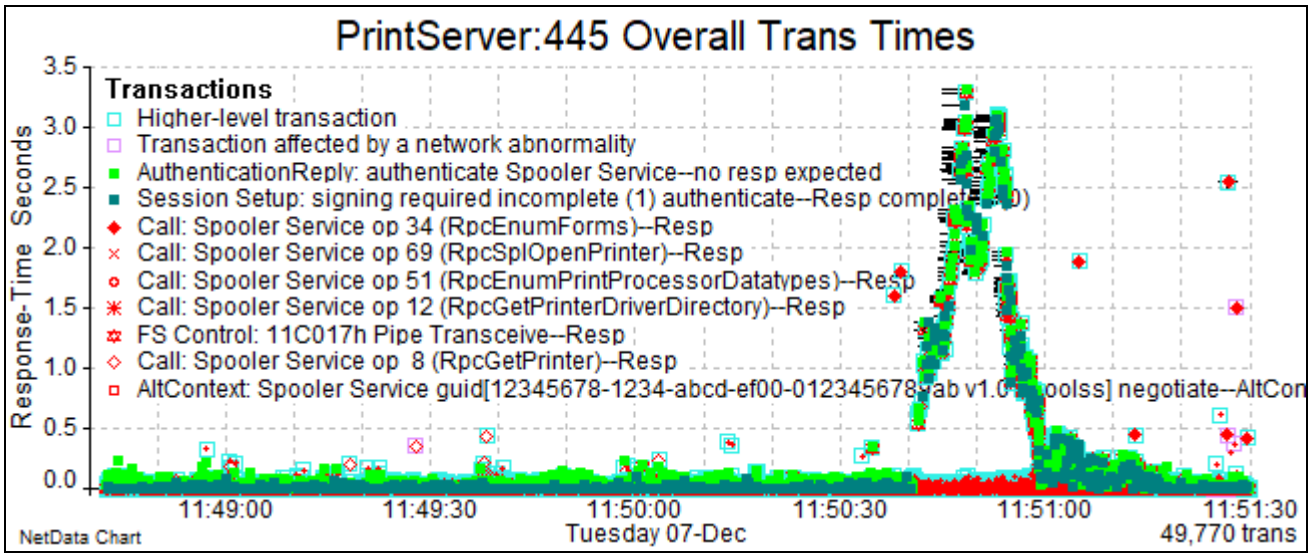
The third authentication phase is handled by a rare type of RPC client message that doesn't receive a response. Rather than assigning this transaction a zero response time NetData assumes that Phase 3 remains active until the subsequent RPC call receives a response. This artefact is useful in identifying congestion in authentication processes, particularly if authentication requires access to a backend server such as a Domain Controller (DC). An AuthenticationReply message is often combined with a subsequent call to AsyncOpenPrinter in the same packet, and the second call must wait in the server until the authentication reply has been checked.

This chart characterises a significant authentication problem that affected performance over a 30-second period.

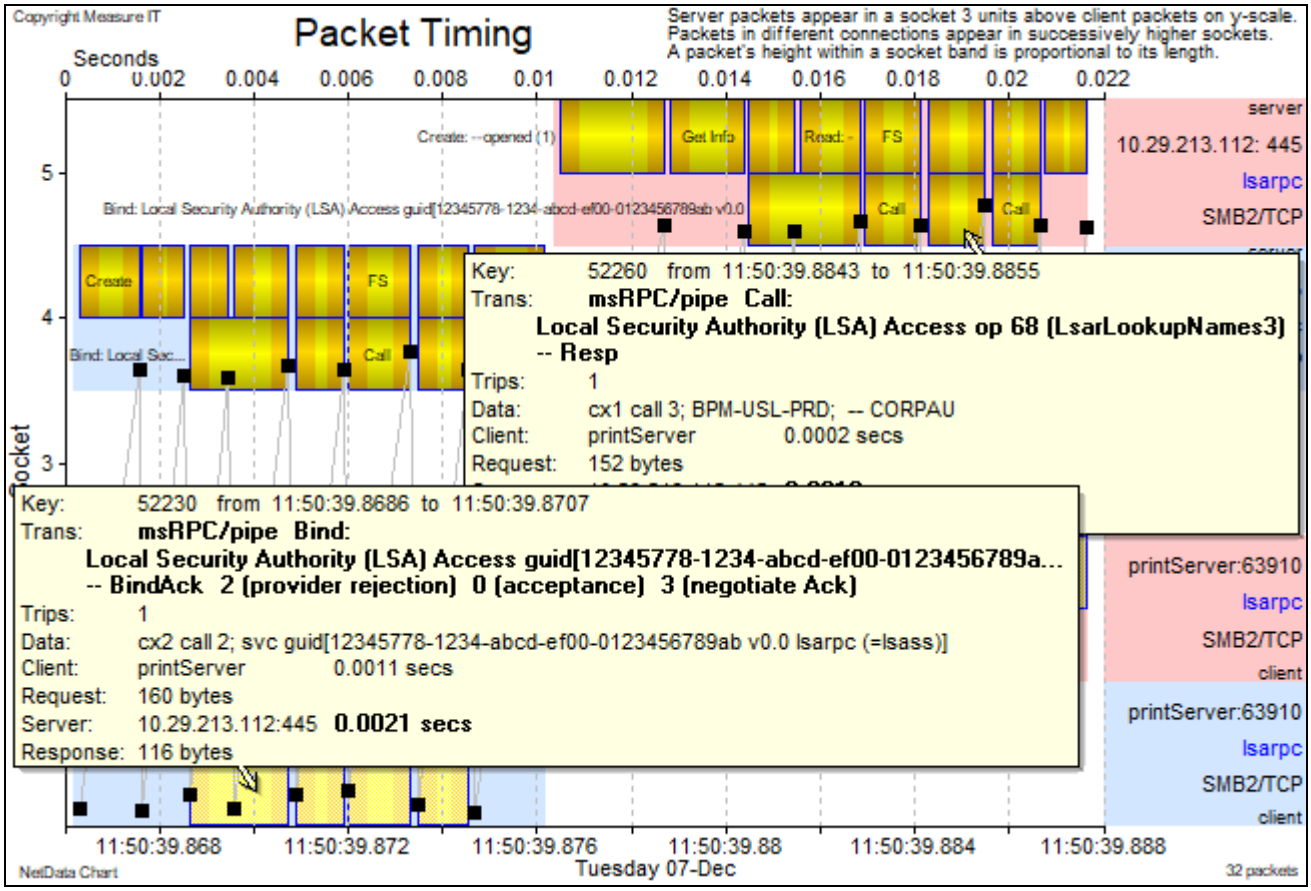


The same print server also provided a Spooler Service accessed by RPCs running through named pipes on SMB2/TCP. That service exhibited a similar authentication-performance problem over the same period and affected the three-phase NTLMSSP authentications for both SMB Session Setup and AltContext transactions.

Both charts, above and below, confirm that all authentications were subject to abnormal delays when the problem occurred.



The diagnosis found corresponding abnormal delays in RPC calls to the NetLogon Service of a domain controller and implicated RPC calls to an LSA (Local Security Authority) service. To better understand the nature of the LSA traffic the following chart splits the traffic of the relevant SMB connection into its related secondary connections that each display the transactions of a different named pipe opened to the *lsarpc* service. SMB Write, Read and Pipe-Transceive transactions conveyed the LSA RPC transactions. On each socket band of a secondary connection – i.e. named pipe – transaction bars are stacked two high: the top row displays the SMB transactions; and the bottom row displays the RPC transactions conveyed by the SMB transactions.

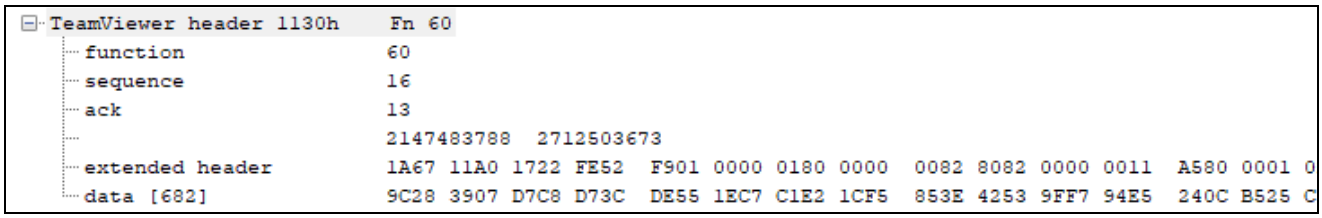


16 Remote Control

16.1 TeamViewer

TeamViewer is a cloud-based remote connectivity platform for instant remote access, remote control, and remote support. It provides secure connections from a broad range of locations without requiring a VPN, and integrates videoconferencing, secure file sharing, and screen sharing tools.

TeamViewer traffic normally uses TCP port 5938. NetData decodes only the first part of message headers, extracting a function code and message sequence numbers. It characterises transactions identified by reversals of data flow.



16.2 Secure Shell

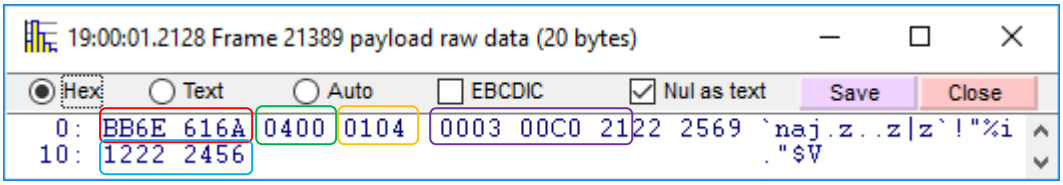
NetData recognises Secure Shell traffic on ports other than 22, provided it sees the initial part of a session when messages are not encrypted. It decodes unencrypted messages and measures response times

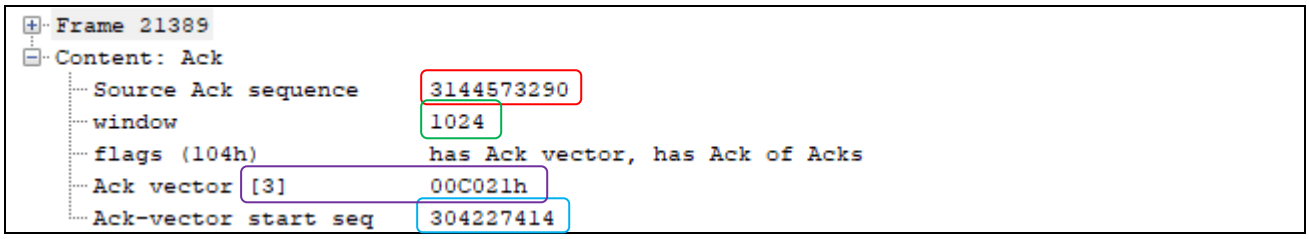
16.3 Remote Desktop Protocol: UDP Transport Extension Protocol v1

With Remote Desktop Protocol (RDP) Version 8 Microsoft introduced an option to use UDP rather than TCP for remote connections to port 3398. To make UDP sufficiently reliable to support SSL encryption it is enhanced by a UDP Transport Extension Protocol (UTEP) that has sequence numbering, data recovery, flow and congestion controls like TCP but avoiding many of TCP's weaknesses. It inserts a header that normally needs 20 bytes between the UDP header and the application payload. The protocol is described in the Microsoft document *[MS-RDPEUDP].pdf*

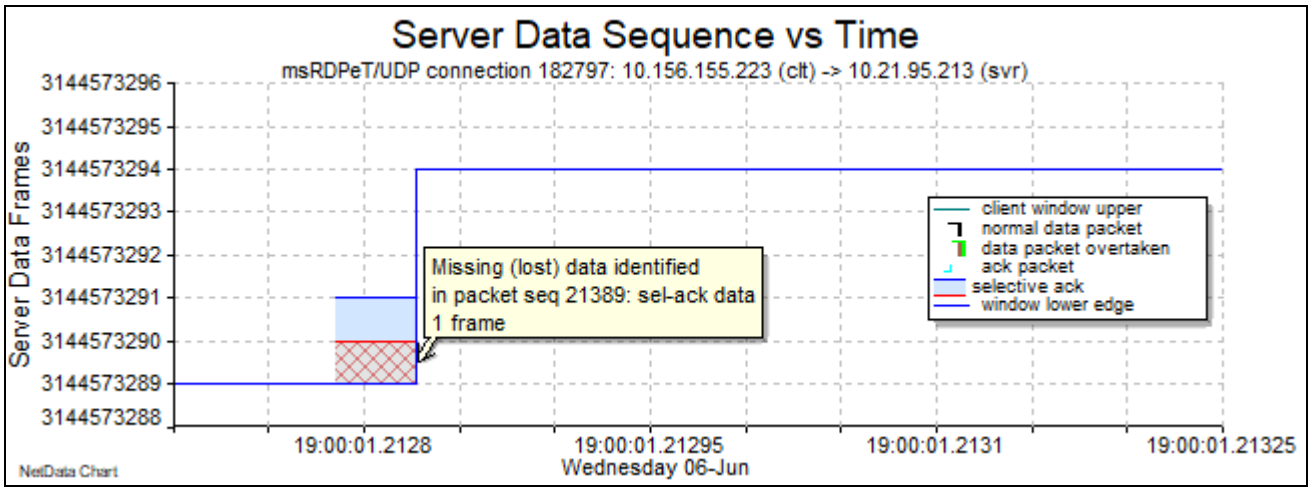
NetData fully decodes the UTEP headers, extracting packet sequence numbers, selective acknowledgement information and congestion-control flags, all of which can be displayed on the data-flow chart much like TCP connections. The only significant difference between flow charts for TCP and UTEP connections is that UTEP sequence numbers refer to whole packets that carry application data rather than data bytes.

NetData describes the following packet as an Ack because it doesn't have any payload data – only a 20-byte UTEP header. It starts with the sequence number of an acknowledged *Source* (application payload) data packet and the current receive window size.





The Ack vector specifies any sequence gaps in recently received packets, starting with the packet identified by a 'start' sequence number received as an 'Ack of Acks' in an earlier packet in the opposite direction to this ack. The vector uses run-length encoding, with a number of single-byte elements that specify alternating counts of received and missing packets. This vector specified a run of 34 received packets, one missing packet, and one received packet, which is depicted by NetData like a selective ack in a TCP packet:

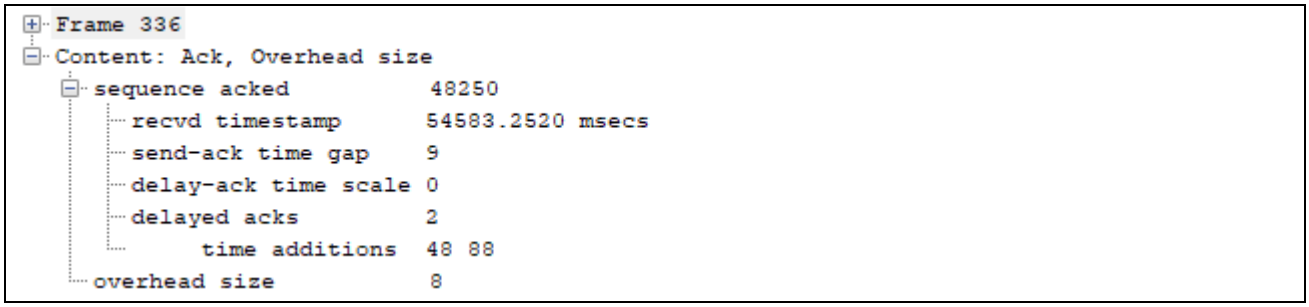


16.4 Remote Desktop on UDP with Extended Transport Version 2

When controlling a remote desktop Microsoft Remote Desktop Protocol (RDP) usually establishes two connections with port 3389 on the remote machine, one connection using TCP and the other using an extended transport protocol on UDP. The payload on both connections is encrypted with TLS 1.2 (SSL 3.3).

The extended transport protocol, tagged by NetData as 'RDPeT2', is a second version specified in the Microsoft document *[MS_RDPEUDP2].pdf*. It provides functions of lost-data recovery and flow control similar to TCP but is more efficient and avoids some of TCP's weaknesses. Its header can be understood by constructing it in three steps. The first two bytes specify a window size and in a set of six flags specify what options follow: an acknowledgement with a timestamp and delay information; an overhead-size indicator derived from the average length of headers preceding received data payloads; delayed-ack information; the lowest unacknowledged sequence number ('ack of acks'); an ack vector (providing the functions of a SACK); and the data payload preceded by a channel sequence number.

The second step prefixes the header with a single byte that handles ambiguities with very short packets and in future may indicate a higher-level packet type. The final step is to change byte order, swapping bytes one and eight.

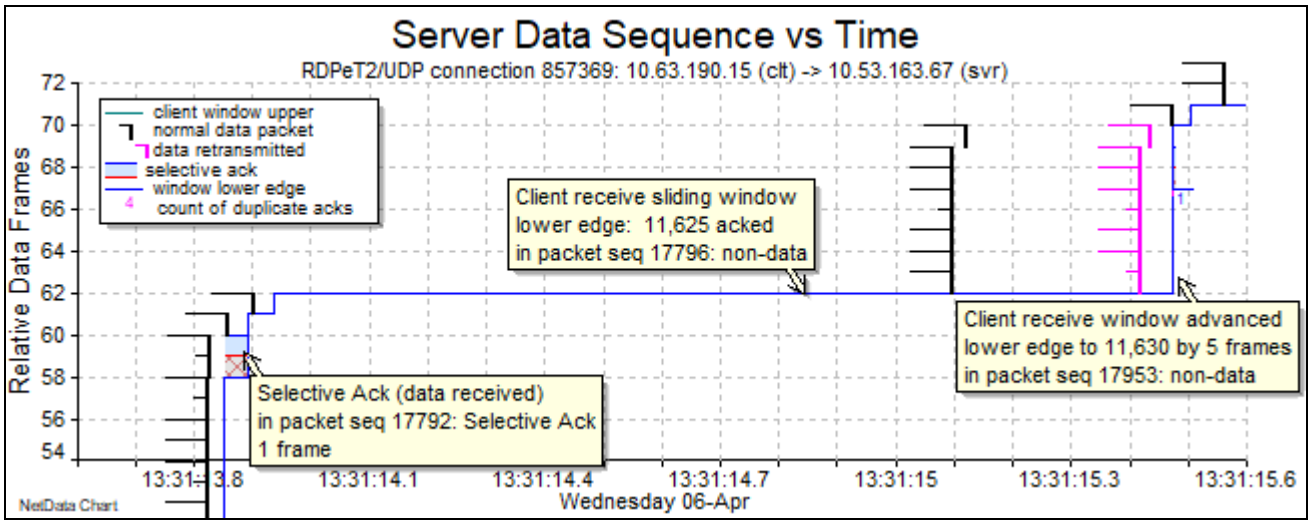


Packets with data carry two sequence numbers: a transmitted-packet number that always increments by one, for detection of packet losses; and a channel- or data-sequence number that identifies the original data segment should it be retransmitted.

Packet-sequence and ack numbers appear in the Context column of the packet table, preserving the normal Data Seq and Data Ack columns for data-sequence numbers, the numbers that can be plotted on the flow chart. The data-ack number is calculated by NetData from the ack number in the transport header and taking into account the most recent, relevant difference between packet- and data-sequence numbers, and adding one to identify the next expected data-sequence number.

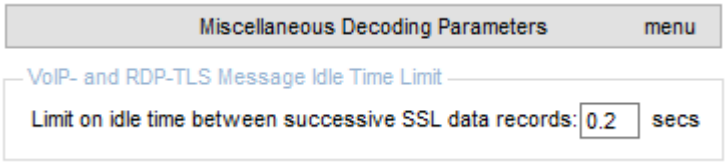
IP ID	Data Seq	Data Ack	AppType	Data	Blks	Function	Context	Content
58752	47635		RDsktPS./RDPeT2	1205	3	Data, Delay ack info;...	48250	delayed acks max 74\ ...
26572		47636	RDPeT2			Ack, Overhead size	ack48250	sequence acked 4763...
58755	47636		RDsktPS./RDPeT2	1230	1	Data; AppData	48251	AppData[42]
58756	47637		RDsktPS./RDPeT2	703	2	Data; AppData (2)	48252	AppData[1628]\ AppD...
26573	6345	47638	RDsktPS./RDPeT2	215	1	Ack, Data, Overhea...	7146; ack48252	sequence acked 4763...
58759		6346	RDPeT2			Ack, Overhead size	ack7146	sequence acked 6346...
58761	47638		RDsktPS./RDPeT2	1043	2	Data; AppData (2)	48253	AppData[42]\ AppData...
26574		47639	RDPeT2			Ack	ack48253	sequence acked 4763...
58762	47639		RDsktPS./RDPeT2	1230	1	Data; AppData	48254	AppData[42]

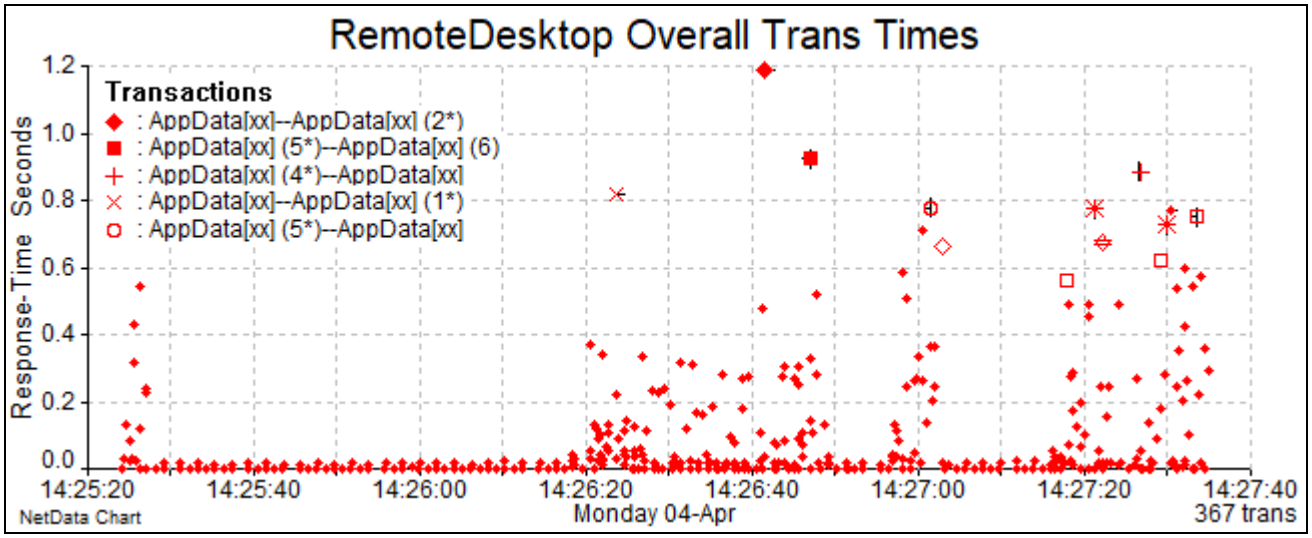
The resulting sliding-window graphs on the flow chart look like TCP sliding windows apart from the fact that sequence numbers refer to whole data segments rather than individual bytes. Retransmitted data segments are highlighted and the information in ack vectors is displayed like TCP selective acks.

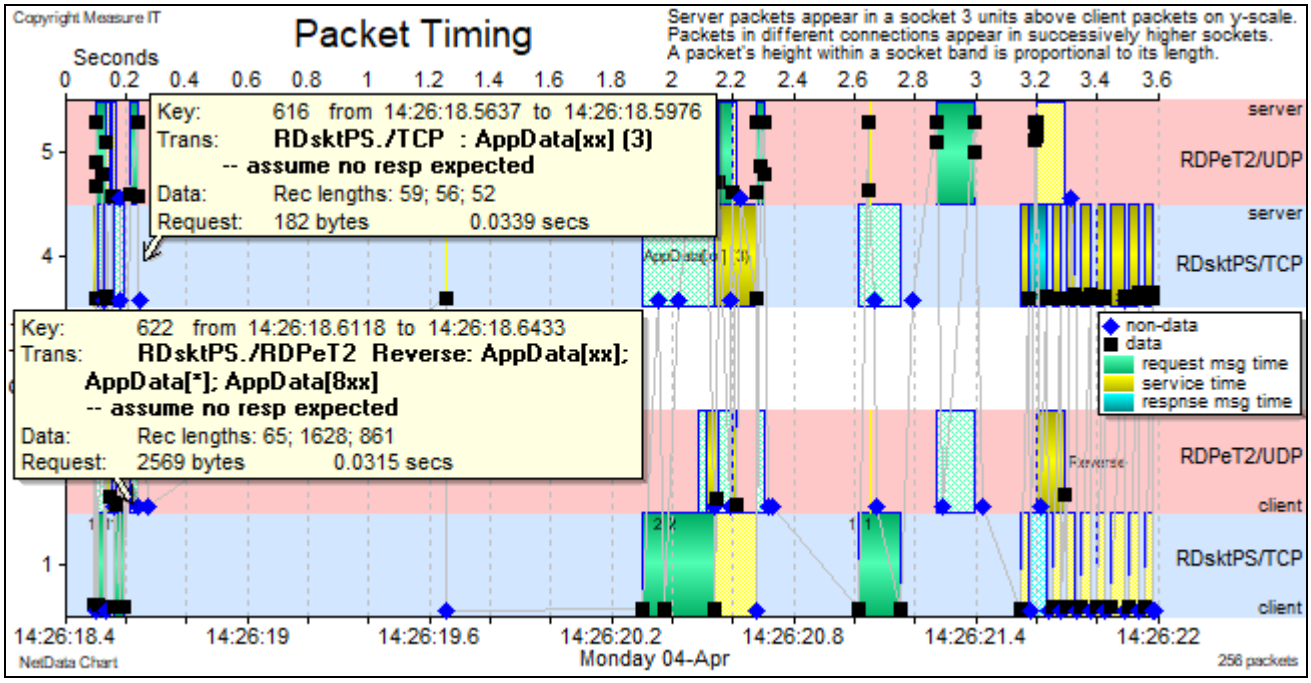


NetData unpacks every header and characterises pseudo transactions derived from flows of TLS records. Not all RDP packet sequences form pairs of request and response messages but this TLS decoder characterises some sustained data flows in one direction only, and data exchanges that appear to be round-trips, whether initiated by the client or the server. Such a pseudo transaction is terminated at every occurrence of an idle period longer that 200 ms. This scheme reveals patterns of RDP activity on both the performance and timing charts for correlation with other system behaviour, but avoids presenting pseudo transactions with misleadingly-large response times.

The transaction-controlling idle period can be changed in a miscellaneous decoding parameter under the heading ‘VoIP- and RDP-TLS Message Idle-Time Limit’ on the Decoding page of controls:

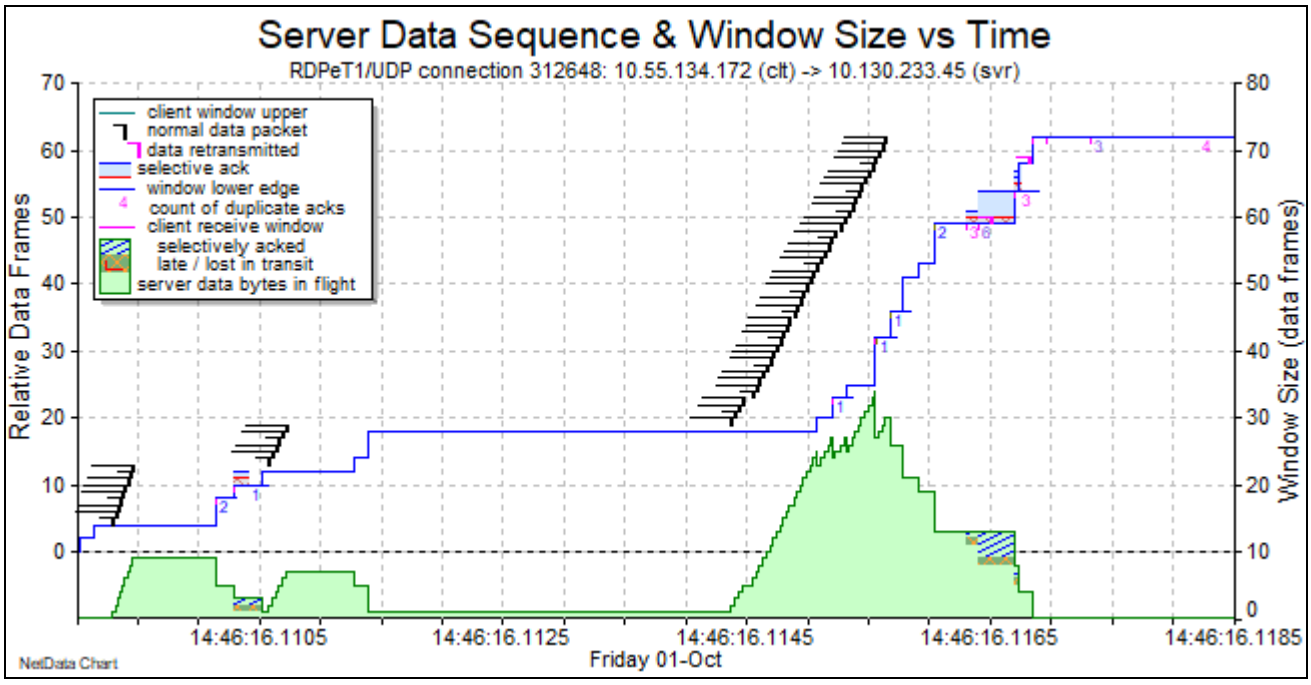






This packet-timing chart compares the activity on both the TCP (grey band) and UDP (pink) connections. The two popups descibe data flows without a response (‘assume no response expected’). As in this example most of the apparent round-trips and flows without responses on the UDP connection are initiated by the server rather than the client (‘Reverse: AppData[xx]’).

NetData’s decoder for the first version of the extended transport protocol, tagged by NetData as ‘RDPeT1’, has been upgraded to the same standard as the RDPeT2 decoder. The green area on the window-size chart below plots the number of data packets in flight, waiting for acknowledgement, and the sliding window shows how the server retransmits data segments when prompted by ack vectors from the client.



17 Virtual Private Networks and Tunnelling

17.1 Teredo Tunnelling IPv6 over UDP

NetData decodes packets of the Teredo protocol (RFC 4380) that gives IPv6 connectivity to IPv6-capable hosts that are on an IPv4 network and have no native connection to an IPv6 network.

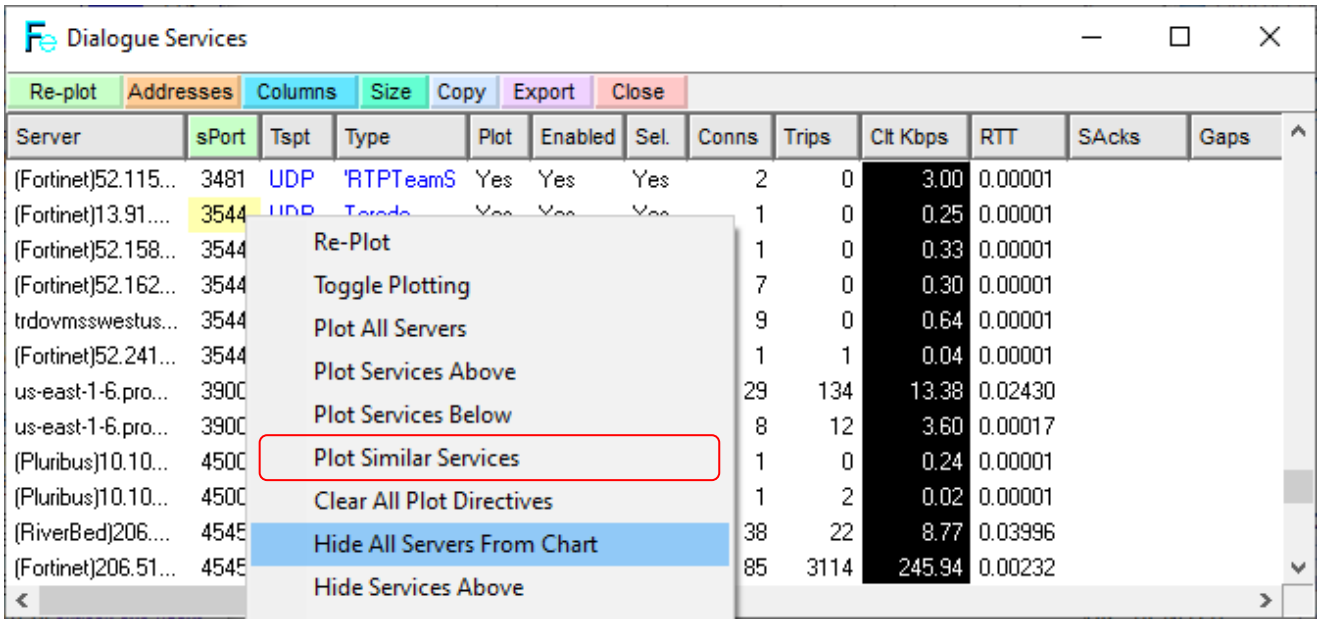
Teredo encapsulates IPv6 datagram packets within IPv4 UDP packets for routing them on the IPv4 Internet and through NAT devices. Teredo nodes elsewhere on the IPv6 network (called Teredo relays) receive the packets, remove their encapsulation, and pass them on.

A Teredo client is assigned a public IPv6 address that starts with a Teredo prefix (2001::/32) and embeds the client's IPv4 address, its port number obfuscated by inverting all its bits, and the public IPv4 of the NAT device, also obfuscated. NetData displays the embedded addresses in clear.

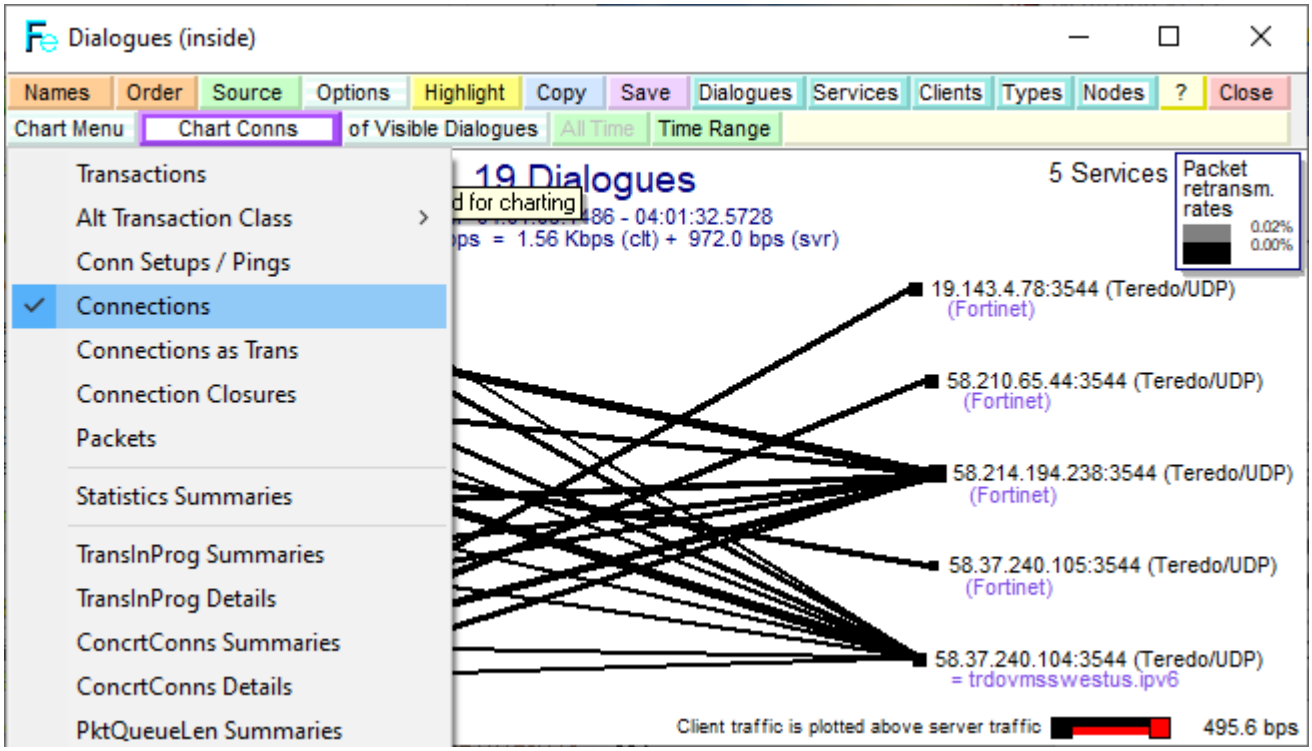
The IPv4 packets access Teredo services through UDP port 3544. NetData records the IPv4 addresses and port numbers of each packet in the frame-description column of the packet table, reserving the normal columns for the packet's IPv6 information. NetData creates two related connection records for each connection, for the IPv4 and IPv6 details respectively, but packet records refer only to the IPv6 connection record.

Both the IPv4 and IPv6 versions of each dialogue will appear on dialogue charts, but packet records for charting can be loaded only by referring to the IPv6 version. However, if the IPv4 connection records are loaded, a new option – 'Load Related Connections' – on the connection table will load all the related connection records, and they can be used to load connection packets with the option 'Load Packets of Similar Connections'.

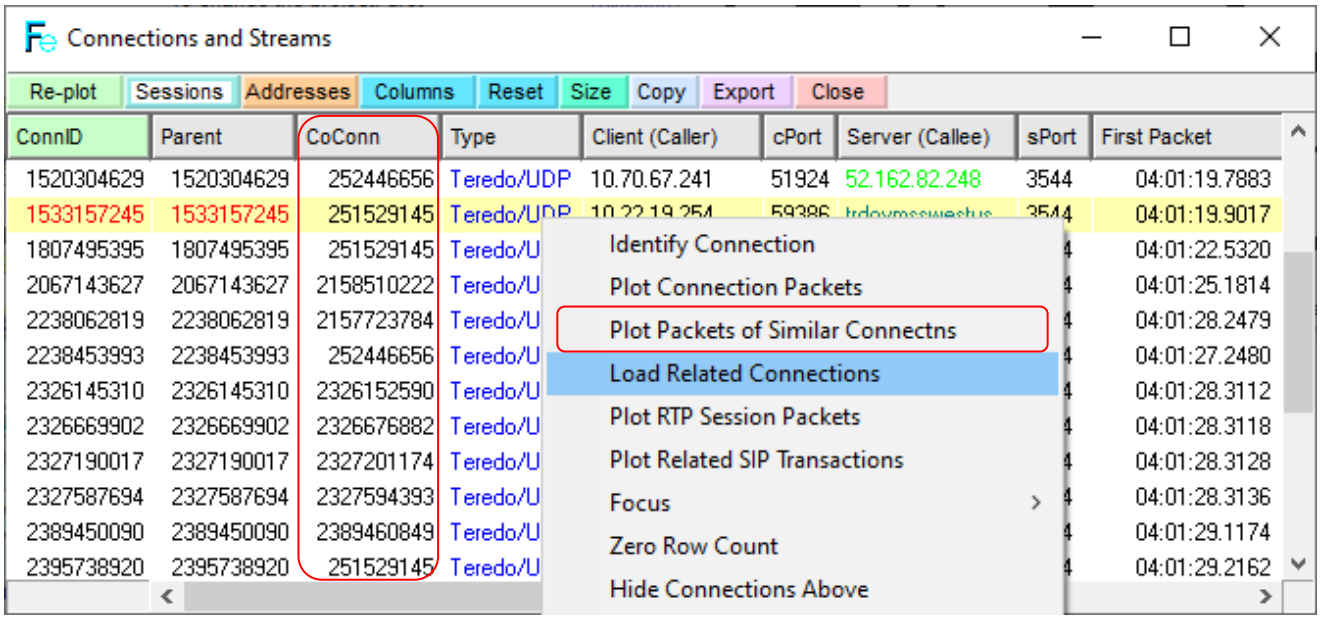
All the IPv4 Teredo connection records can be loaded from the dialogue chart. If the project has many thousands of dialogues the Teredo dialogues are unlikely to appear on the chart without severe filtering, and a filter can be applied in the Services table to display only server ports numbered 3544: sort the server-port column ('sPort') and right-click on any cell with number 3544; choose first to 'Hide All Servers From Chart' and then choose 'Plot Similar Services'; click the green button 'Re-plot'.



On the dialogue chart ensure that Connections are enabled below the Chart Menu button and click the Chart Conns button.



Right-click anywhere on the connections table and choose ‘Load Related Connections’, to load records of all the connections whose IDs appear in the CoConn column.



The loaded connection records appear in the table below.

ConnID	Parent	CoConn	Type	Client (Caller)	cPort	Server (Callee)	sPort	First Packet
323619296	323619296	323612597	IP6hOnly/IP6	2001::34A2:52...	0	2001::349E:D1...		04:01:10.52
1031282423	1031282423	1031275724	IP6hOnly/IP6	2001::34A2:52...	0	2001::D5B:945...		04:01:15.52
2326152590	2326152590	2326145310	IP6hOnly/IP6	2001::34A2:52...	0	2001::34F1:807...		04:01:28.31
2326676882	2326676882	2326669902	IP6hOnly/IP6	2001::34A2:52...	0	2001::34F1:807...		04:01:28.31
2327201174	2327201174	2327190017	IP6hOnly/IP6	2001::34A2:52...	0	2001::34F1:807...		04:01:28.31
2327594393	2327594393	2327587694	IP6hOnly/IP6	2001::34A2:52...	0	2001::34F1:807...		04:01:28.31
2389460849	2389460849	2389450090	IP6hOnly/IP6	2001::34A2:52...	0	2001::34F1:807...		04:01:29.11
251529145	251529145	2395738920	NDiscvry/ICMP6	LL0::FFFF:FFFF...	0	All routers		04:01:10.08
252446656	252446656	2238453993	NDiscvry/ICMP6	LL0::FFFF:FFFF...	0	LL0::8000:F227...		04:01:10.09
2157723784	2157723784	2238062819	NDiscvry/ICMP6	LL0::FFFF:FFFF...	0	LL0::8000:F227...		04:01:26.23
2158510222	2158510222	2067143627	NDiscvry/ICMP6	LL0::8000:FFFF...	0	All routers		04:01:26.24
251522554	251522554	252446656	Teredo/UDP	10.37.111.109	56489	52.162.82.248	3544	04:01:10.08

In the case above, the loaded IPv6 connections were of two distinct types: an ICMPv6 (Neighborhood Discovery) Router Solicitation or Advertisement; and those with an IPv6 header and no IP payload but a packet trailer of 16 bytes or more containing at least one IPv4 address and port number as in

```
0104 5764 726C 030E  0000h  192.168.42.3:52655  100.97.45.26:52655
```

The packet description below illustrates a typical sequence of Teredo headers, IPv6 header, and optional IP payload – an ICMPv6 neighbourhood discovery packet:

```
⊞ Frame 628413
⊟ Content: Router Advertisement; authentication; origin
  ⊟ Teredo authentication:
      client ID       [0]
      authentication [0]
      nonce                 4271 3958 EBBE 45E3h
      confirmation          0
    Teredo origin indicatn: 198.105.65.197:56382
    IPv6 header:            details in Frame description
  ⊟ Router Advertisement:
      current hop limit     0
      flags
      router lifetime       0 secs
      reachable time        15000 ms
      retransm timer        2000 ms
    ⊟ prefix info           2001::34A2:52F8:FF00:0:20:100/64
        prefix length       64
        flags (64)          autonomous address configuration
        valid lifetime      -1
        preferred lifetime -1
                            0
  ⊟ IPv4 addresses embedded in IPv6 addresses:
      source srvr socket    0.0.0.0:3544
                  client    52.162.82.248
      destination socket    0.0.0.0:0
                  client    0.0.0.1
```

17.2 ZeroTier One Connectivity Traffic

NetData detects traffic of an Atlassian connectivity tool ZeroTier One that uses UDP port 9993. ZeroTier One is a portable client application providing connectivity to public or private virtual networks.

17.3 WireGuard VPN Tunnels

NetData decodes the unencrypted parts of packets using the WireGuard protocol which was designed to replace both IPsec and TLS-based solutions for Virtual Private Networks (VPNs), while being more secure and easier to use. A thorough specification is provided by the PDF *WireGuard: Next Generation Kernel Network Tunnel*, by Jason A Donenfeld, at www.wireguard.com.

17.4 OpenVPN

NetData decodes openVPN traffic in which packets are encapsulated in plaintext over UDP. OpenVPN normally uses port 1194.

17.5 GPRS Tunnelling Protocol

NetData decodes traffic using the General Packet Radio Service (GPRS) Tunnelling Protocol (GTP) for user data (GTP-U). The UDP destination port for GTP user data is 2152. The lower-layer IP addresses and the GTP header details of GTP/UDP packets are recorded in the frame-description field of the packet table, leaving the normal address fields to describe the encapsulated IP packets.

17.6 Point-to-Point Protocol over Ethernet

A decoder has been added to decode headers of the PPPoE protocol that is often found in wireless access links to the Internet. It fully decodes packets during the active discovery stage, and, during the session stage, the PPPoE headers that precede the headers of other protocols such as IP and PPP-LCP.

17.7 General Routing Encapsulation

NetData can decode IP packets encapsulated by Cisco's General Routing Encapsulation (GRE) protocol for tunnelling through an IP network. The outer IP addresses and GRE flags are described in the Frame Description field of each packet, and all other packet fields describe the encapsulated packet.

17.8 Unscrambling the PPP Omelette

Prior to the advent of mobile devices such as iPhones and tablets, the Point-to-Point Protocol (PPP) was rarely seen outside dial-up lines. Its attraction for new devices lies in its ability to multiplex many bi-directional channels, with diverse protocols, over point-to-point links in almost any physical network; its support for compression and encryption in individual channels; and its suite of control protocols to configure the channels and provide authentication.

Mobile devices now commonly tunnel their PPP connections through IP networks using Cisco's GRE (Generic Routing Encapsulation) protocol over IP, together with PPTP (Point-to-Point Tunneling Protocol) connections over TCP, to create a Virtual Private Network (VPN). GRE inserts optional sequence and ack numbers which may be useful to detect packet loss and to re-order packets in case the encapsulated stream has compressed or encrypted data. The GRE header also specifies the EtherType of the protocol in the next higher layer, and a key value for that protocol. The next higher protocol is usually encapsulated PPP (EtherType 880Bh), in which case GRE's key value contains the PPP message length and call ID. The call ID determines the multiplexed channel within the GRE tunnel.

A PPTP client is usually referred to as a PPTP Access Concentrator (PAC), and the server as a PPTP Network Server (PNS).

The encapsulated-PPP header contains the EtherType of the next higher protocol, preceded by the two fixed bytes FF03h that appear like an LLC1 (Logical Link Control Type 1) or HDLC (High-level Data Link Control) header with a control byte (03) specifying unnumbered frames. Usually, the next higher protocol is either one of the several PPP control protocols that configure the channel – LCP (Link Control), IPCP (IP Control), CCP (Compression Control), or CHAP (Challenge Handshake Authentication) – or the packet conveys encapsulated data. If the data is compressed or encrypted it is likely to use one of the respective Microsoft protocols MPPC or MPPE (both using EtherType 00FDh); otherwise the next protocol is likely to be encapsulated IP (EtherType 0021h).

If the higher-layer protocol is TCP/IP, the complete stack becomes TCP/IP/PPP/GRE/IP. For recursive IP stacks like this NetData displays only the higher-layer parameters and connection ID in the normal columns, while summarising lower-layer parameters in the frame-description column. The lower-layer connection ID appears in the '2nd ID' column of the packet table, and replaces the higher-layer ID in the ConnID column of the transaction table.

The PPP control protocols employ a common format for configuration options conveyed in request and response messages. In general a Configure Request message might generate any one of four types of response: a Terminate Ack message that requests renegotiation (only by LCP); a Configure Reject message that lists rejected options; a Configure Nak message that specifies changed options; and a Configure Ack message that accepts all the requested options. The LCP protocol, for example, might progress through four round-trips, revising the requested options each time, before the negotiation is completed with a Configure Ack. While the client negotiates its options the server negotiates its own options by issuing Configure Request messages in the reverse direction and also progressing through several trips. Furthermore the various control protocols generally conduct their own negotiations, in forward and reverse directions, concurrently, and it is possible that several multiplexed channels might be configured at the same time.

When reading a table of packets it is not easy to match the various responses with their requests. Correct matching must take into account call IDs that determine the channel; which peer initiates the request; which protocol is handling the message; and a transaction identifier in the message header that distinguishes different round-trips. NetData displays the transaction identifier in the Context column of the packet table, and displays a channel ID – regarded by NetData as a secondary connection ID – in the '2nd ID' column of the packet table.

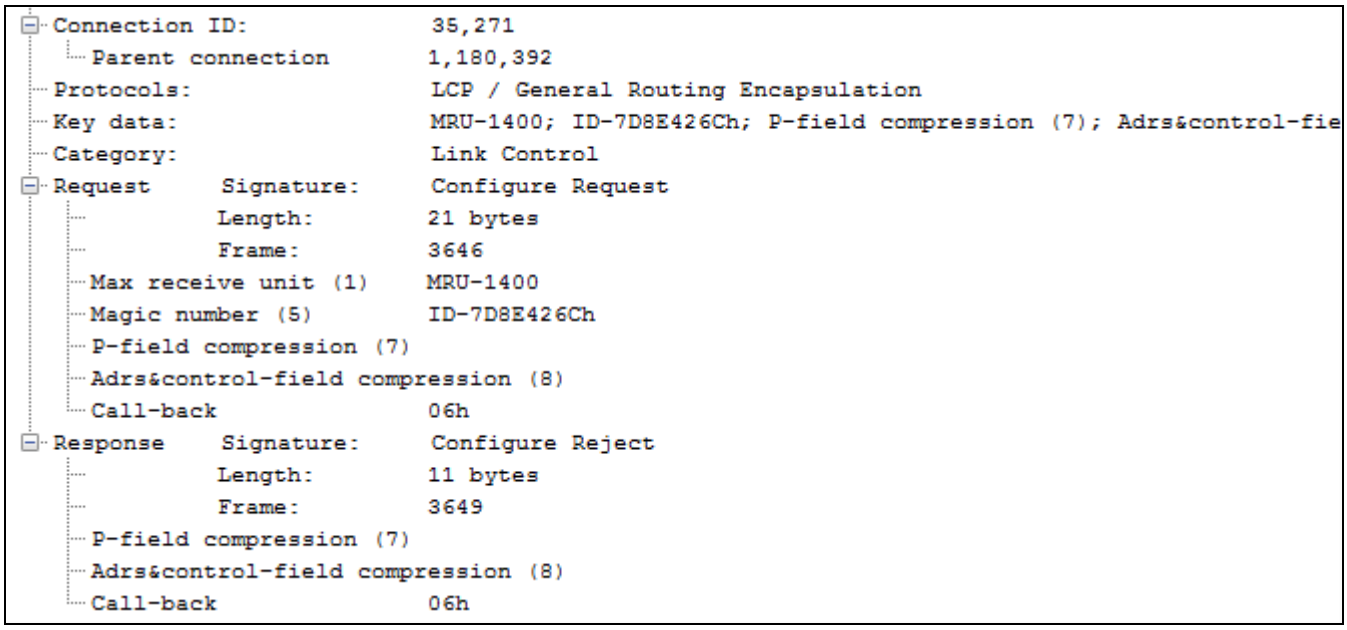
A channel ID is determined by Outgoing-Call Request and Response messages in the PPTP protocol that runs over TCP between the same pair of IP addresses as the GRE/IP tunnel. The client and server nominate unique call IDs which NetData treats somewhat like source and destination port numbers. A PPTP Outgoing-Call Request also includes a serial number which NetData adopts as the channel ID. However, if the serial number is zero, NetData generates its own ID which is a function of the two call IDs.

NetData displays the call ID of each PPP/GRE message; like a source port number, it is appended to the source IP address that appears in the Source column of the packet table. The destination's call ID is not displayed because it is not present in the packet.

The packet table below lists most of the packets involved in establishing a bidirectional channel in a GRE tunnel between a pair of IP addresses. The Data Seq and Data Ack columns display TCP sequence numbers for the PPTP connection, and GRE sequence numbers for the PPP/GRE tunnel.

The Flags column displays either TCP flags or just two flags, A and P, to indicate whether the GRE packet includes an Ack (A) or data sequence number (P). The KeyData column displays call ID and serial numbers for PPTP packets, and the values of options in PPP control messages.

More details of message contents are displayed when a packet or transaction description is requested:

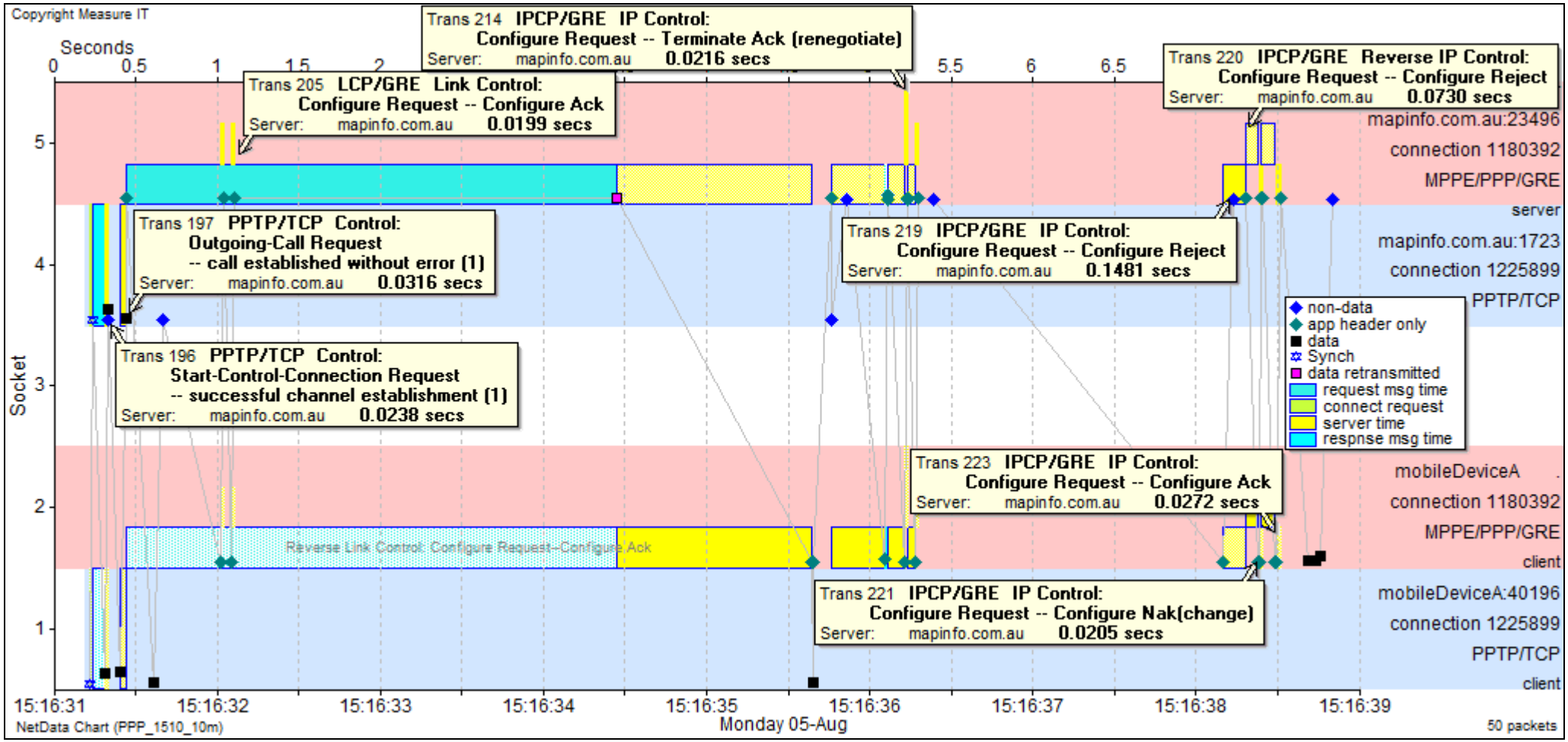


The first LCP Configure Request from the server – the first packet using the channel defined by the PPTP transaction – is often ignored and in the following case was retransmitted after 3 seconds.

	Time Of Day	Seq	Source	Destination	Tspt	Flags	Data Seq	Data Ack	ConnID	App...	Data	Function	C...	KeyData
■	15:16:31.3103	3614	mobileDeviceA:40196	mapinfo.com.au: 1723	TCP	AP	350672...	748242...	1225899	PPTP	156	Start-Control-Connection R...		
◆	15:16:31.3305	3615	mapinfo.com.au: 1723	mobileDeviceA:40196	TCP	A	748242...	350672...	1225899	PPTP				
■	15:16:31.3341	3616	mapinfo.com.au: 1723	mobileDeviceA:40196	TCP	AP	748242...	350672...	1225899	PPTP	156	successful channel establi...		
■	15:16:31.4102	3617	mobileDeviceA:40196	mapinfo.com.au: 1723	TCP	AP	350672...	748242...	1225899	PPTP	168	Outgoing-Call Request		23496, serial 35271
■	15:16:31.4417	3619	mapinfo.com.au: 1723	mobileDeviceA:40196	TCP	AP	748242...	350672...	1225899	PPTP	32	call established without err...		23496, 128
◆	15:16:31.4434	3620	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	P	0		1180392	LCP	21	Configure Request	1	Map-00000000h; PPP-CHA
■	15:16:31.6103	3627	mobileDeviceA:40196	mapinfo.com.au: 1723	TCP	AP	350672...	748242...	1225899	PPTP	24	Set Link Info		128
◆	15:16:31.6666	3628	mapinfo.com.au: 1723	mobileDeviceA:40196	TCP	A	748242...	350672...	1225899	PPTP				
◆	15:16:32.0207	3646	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	P	0		1180392	LCP	21	Configure Request	0	MRU-1400; ID-7D8E426Ch,
◆	15:16:32.0413	3649	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	AP	1	0	1180392	LCP	11	Configure Reject	0	P-field; Adrs&control-field; Ca
◆	15:16:32.0904	3656	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	AP	1	1	1180392	LCP	14	Configure Request	1	MRU-1400; ID-7D8E426Ch
◆	15:16:32.1103	3657	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	AP	2	1	1180392	LCP	14	Configure Ack	1	MRU-1400; ID-7D8E426Ch
■	15:16:34.447	3690	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	P	3		1180392	LCP	21	Configure Request	1	Map-00000000h; PPP-CHA
◆	15:16:35.6439	3706	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	AP	2	3	1180392	LCP	21	Configure Ack	1	Map-00000000h; PPP-CHA
■	15:16:35.6503	3707	mobileDeviceA:40196	mapinfo.com.au: 1723	TCP	AP	350672...	748242...	1225899	PPTP	24	Set Link Info		128
◆	15:16:35.6504	3708	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	P	3		1180392	LCP	18	Identification	2	
◆	15:16:35.6506	3709	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	P	4		1180392	LCP	25	Identification	3	
◆	15:16:35.7688	3712	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	AP	4	2	1180392	CHAP	26	Challenge	105	
◆	15:16:35.7695	3713	mapinfo.com.au: 1723	mobileDeviceA:40196	TCP	A	748242...	350672...	1225899	PPTP				
◆	15:16:35.8572	3714	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	A		4	1180392	Ack				
◆	15:16:36.0906	3723	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	AP	5	4	1180392	CHAP	58	Response	105	
◆	15:16:36.114	3724	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	AP	5	5	1180392	CHAP	63	Success	105	
◆	15:16:36.1147	3725	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	P	6		1180392	CCP	10	Configure Request	1	40-bit keys, 128-bit keys, sta
◆	15:16:36.2103	3726	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	AP	6	6	1180392	CCP	10	Configure Request	4	compress, 40-bit keys, 128-b
◆	15:16:36.2104	3727	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	P	7		1180392	IPCP	22	Configure Request	5	0.0.0.0; NBNS-0.0.0.0
◆	15:16:36.2104	3728	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	P	8		1180392	CCP	10	Configure Nak(change)	1	128-bit
◆	15:16:36.2312	3729	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	AP	7	8	1180392	CCP	10	Configure Nak(change)	4	128-bit
◆	15:16:36.232	3730	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	P	8		1180392	IPCP	4	Terminate Ack	5	
◆	15:16:36.233	3731	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	P	9		1180392	CCP	10	Configure Request	2	128-bit keys, stateless (new
◆	15:16:36.2805	3732	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	AP	9	9	1180392	CCP	10	Configure Request	6	128-bit keys, stateless (new
◆	15:16:36.2806	3733	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	P	10		1180392	CCP	10	Configure Ack	2	128-bit
◆	15:16:36.3012	3734	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	AP	10	9	1180392	CCP	10	Configure Ack	6	128-bit
◆	15:16:36.3924	3735	mapinfo.com.au:234...	mobileDeviceA	PPP/GRE	A		10	1180392	Ack				
◆	15:16:38.1601	3750	mobileDeviceA: 128	mapinfo.com.au	PPP/GRE	P	11		1180392	IPCP	22	Configure Request	7	0.0.0.0; NBNS-0.0.0.0

	Request Strt	Type	Description	End Rsp	ConnID	Data
•	15:16:31.3103	PPTP/TCP	Control: Start-Control-Connection Request--successful ch...	0.0238	1225899	
•	15:16:31.4102	PPTP/TCP	Control: Outgoing-Call Request--call established without e...	0.0316	1225899	23496, serial 35271; Resp 23496, 128
◆	15:16:31.4434	LCP/GRE	Reverse Link Control: Configure Request--Configure Ack	1.1969	35271	Map-00000000h; PPP-CHAPv2 (algorithm 81h); ID-FBC25276h; Ack Map-00000000h; PPP-CHAPv
•	15:16:32.0207	LCP/GRE	Link Control: Configure Request--Configure Reject	0.0206	35271	MRU-1400; ID-7D8E426Ch; P-field compression (7); Adrs&control-field compression (8); Call-back; R
•	15:16:32.0904	LCP/GRE	Link Control: Configure Request--Configure Ack	0.0199	35271	MRU-1400; ID-7D8E426Ch; Ack MRU-1400; ID-7D8E426Ch
+	15:16:35.7688	CHAP/GRE	Reverse Handshake Authentication: Challenge--Respon...	0.3452	35271	Access granted
•	15:16:36.1147	CCP/GRE	Reverse Compression Control: Configure Request--Config...	0.0957	35271	40-bit keys, 128-bit keys, stateless (new key every packet); Nak(change) 128-bit
•	15:16:36.2103	CCP/GRE	Compression Control: Configure Request--Configure Nak(...	0.0209	35271	compress, 40-bit keys, 128-bit keys, 56-bit keys, stateless (new key every packet); Nak(change) 128
•	15:16:36.2104	IPCP/GRE	IP Control: Configure Request--Terminate Ack (renegotiate)	0.0216	35271	0.0.0.0; NBNS-0.0.0.0
•	15:16:36.233	CCP/GRE	Reverse Compression Control: Configure Request--Config...	0.0476	35271	128-bit keys, stateless (new key every packet); Ack 128-bit
•	15:16:36.2805	CCP/GRE	Compression Control: Configure Request--Configure Ack	0.0207	35271	128-bit keys, stateless (new key every packet); Ack 128-bit
•	15:16:38.1601	IPCP/GRE	IP Control: Configure Request--Configure Reject	0.1481	35271	0.0.0.0; NBNS-0.0.0.0; Reject NBNS-0.0.0.0
•	15:16:38.307	IPCP/GRE	Reverse IP Control: Configure Request--Configure Reject	0.0730	35271	Van-Jacobson comprsd header, max slot 15, slot ID may be comprsd; 192.168.0.6; Reject Van-Jaco
•	15:16:38.3909	IPCP/GRE	IP Control: Configure Request--Configure Nak(change)	0.0205	35271	0.0.0.0; Nak(change) 192.168.0.226
•	15:16:38.4003	IPCP/GRE	Reverse IP Control: Configure Request--Configure Ack	0.0797	35271	192.168.0.6; Ack 192.168.0.6
•	15:16:38.4909	IPCP/GRE	IP Control: Configure Request--Configure Ack	0.0272	35271	192.168.0.226; Ack 192.168.0.226
•	15:20:45.7574	LCP/GRE	Link Control: Terminate Request--Terminate Ack	0.0294	35271	
•	15:20:46.0078	PPTP/TCP	Control: Call-Clear Request--> no resp; term by server	0.0225	1225899	49152
•	15:20:55.0675	PPTP/TCP	Control: Start-Control-Connection Request--successful ch...	0.0202	5730972	
□	15:20:55.1875	PPTP/TCP	Control: Outgoing-Call Request--call established without e...	0.0278	5730972	0, serial 35272; Resp 0, 256
×	15:20:55.2164	LCP/GRE	Reverse Link Control: Configure Request--Configure Ack	0.0814	35272	Map-00000000h; PPP-CHAPv2 (algorithm 81h); ID-9A6F93E0h; Ack Map-00000000h; PPP-CHAPv
•	15:20:55.2976	LCP/GRE	Link Control: Configure Request--Configure Reject	0.0172	35272	MRU-1400; ID-21993274h; P-field compression (7); Adrs&control-field compression (8); Call-back; Re
•	15:20:55.3677	LCP/GRE	Link Control: Configure Request--Configure Ack	0.0172	35272	MRU-1400; ID-21993274h; Ack MRU-1400; ID-21993274h
+	15:20:55.3859	CHAP/GRE	Reverse Handshake Authentication: Challenge--Respon...	0.1112	35272	Access granted
✲	15:20:55.4978	CCP/GRE	Reverse Compression Control: Configure Request--Config...	0.0619	35272	40-bit keys, 128-bit keys, stateless (new key every packet); Nak(change) 128-bit
•	15:20:55.5482	CCP/GRE	Compression Control: Configure Request--Configure Nak(...	0.0164	35272	compress, 40-bit keys, 128-bit keys, 56-bit keys, stateless (new key every packet); Nak(change) 128
•	15:20:55.5483	IPCP/GRE	IP Control: Configure Request--Terminate Ack (renegotiate)	0.0170	35272	0.0.0.0; DNS-0.0.0.0; NBNS-0.0.0.0
◇	15:20:55.5764	CCP/GRE	Reverse Compression Control: Configure Request--Config...	0.0612	35272	128-bit keys, stateless (new key every packet); Ack 128-bit
•	15:20:55.6087	CCP/GRE	Compression Control: Configure Request--Configure Ack	0.0165	35272	128-bit keys, stateless (new key every packet); Ack 128-bit
■	15:20:57.3883	IPCP/GRE	IP Control: Configure Request--Configure Reject	0.2715	35272	0.0.0.0; DNS-0.0.0.0; NBNS-0.0.0.0; Reject NBNS-0.0.0.0

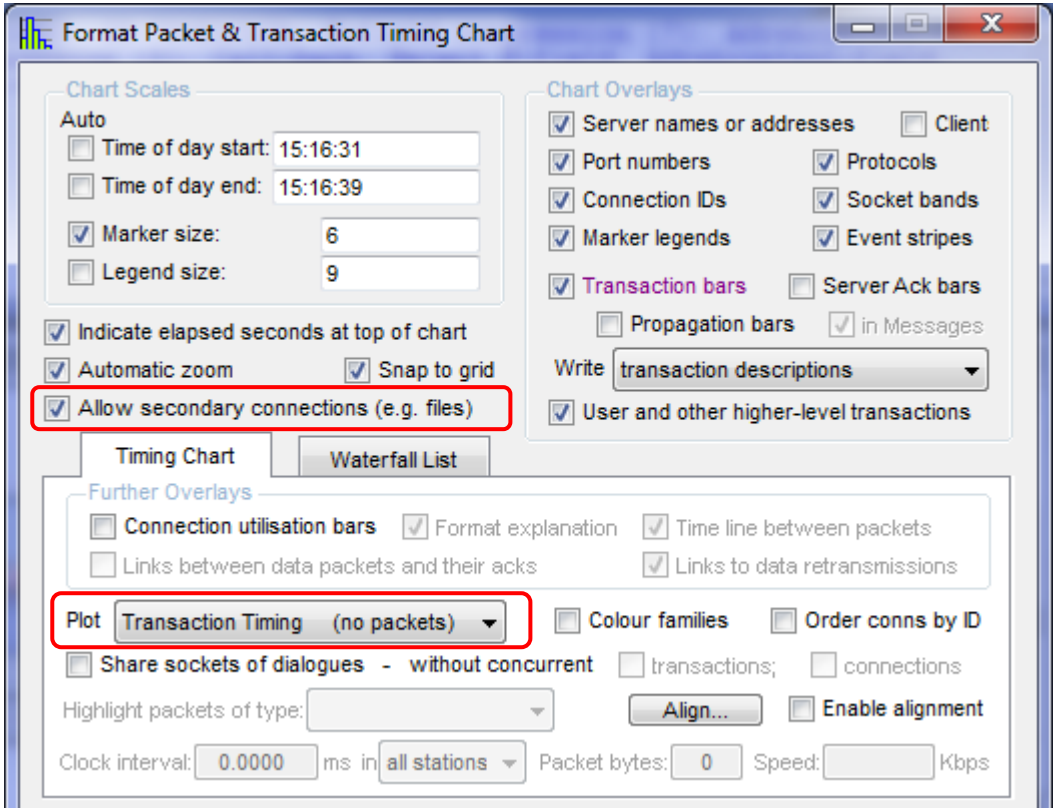
This table displays the PPTP and PPP transactions that established two sequential channels in the tunnel. The Data column lists the key data (option values) of request and response messages, separating the two lists with a word such as Reject, Nak or Ack that indicates the type of response. The table provides a clearer picture of how the negotiations progress. Note that the serial number specified in a PPTP transaction becomes NetData's secondary connection ID (i.e. the channel ID) that appears in the ConnID column for PPP transactions.

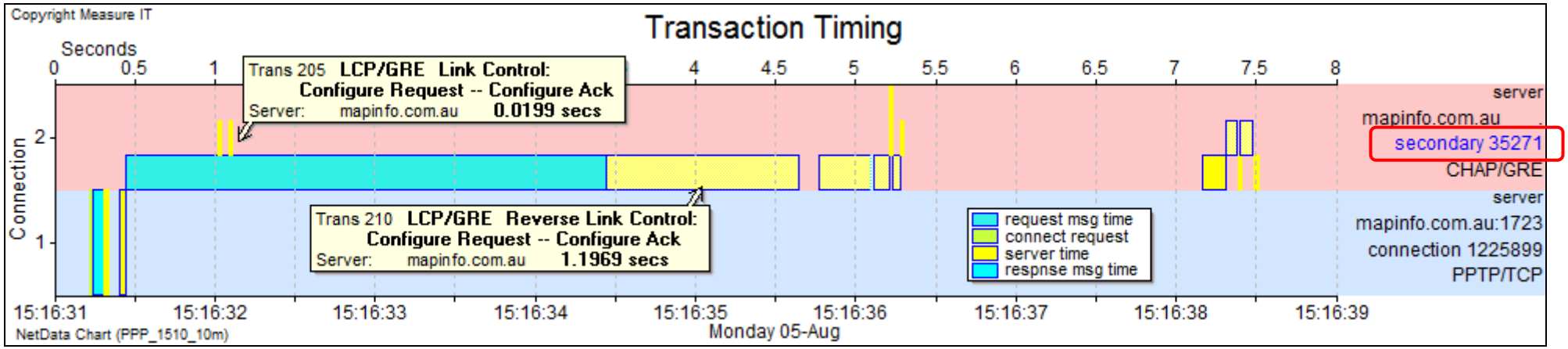


NetData's packet-timing chart shows the relative times of all the transactions on both the PPTP and PPP/GRE tunnel connections, and reveals the degree of concurrency by stacking transaction bars in the socket bands. The popup cream boxes describe some of the transactions to illustrate the progression through Terminate Ack, Reject, Nak and Ack responses by IPCP. The bright and pale yellow transaction bars distinguish round-trips initiated by the client and the server (described as *Reverse* transactions). Because NetData regards PPP control packets as containing only application headers and no application data, they are marked by blue-green diamonds rather than black squares. This chart makes clear the consequences of the 3-second PPP retransmission timeout – the critical session-establishment thread is delayed by 3 seconds.

As with any multiplexed traffic the timing chart will plot the packets and transactions of the different channels on separate bands when the secondary-connections box is checked. The following chart provides a different view of the same traffic that is displayed in the previous chart, by hiding packet markers and allowing secondary connections.

Because this tunnel had only one bi-directional channel the only effect of allowing secondary connections was to display the secondary connection ID (in blue) rather than the connection ID for the tunnel as a whole.



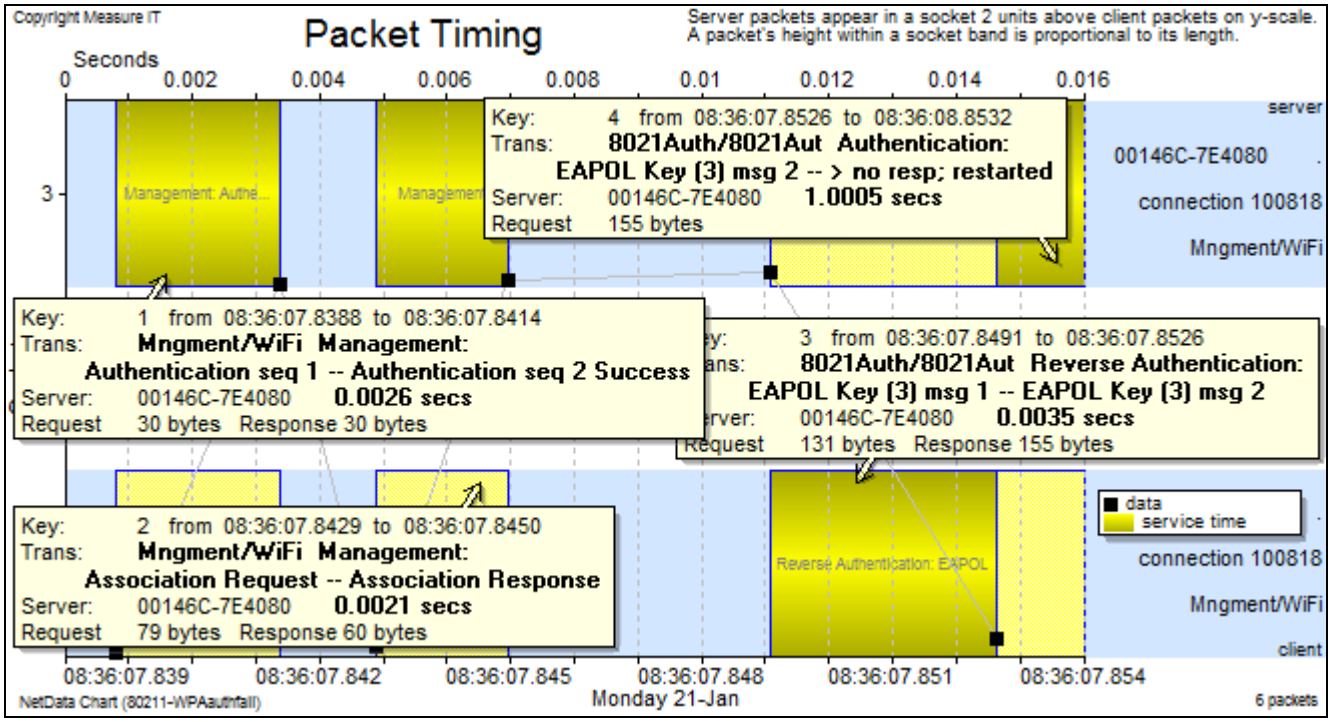


18 WiFi

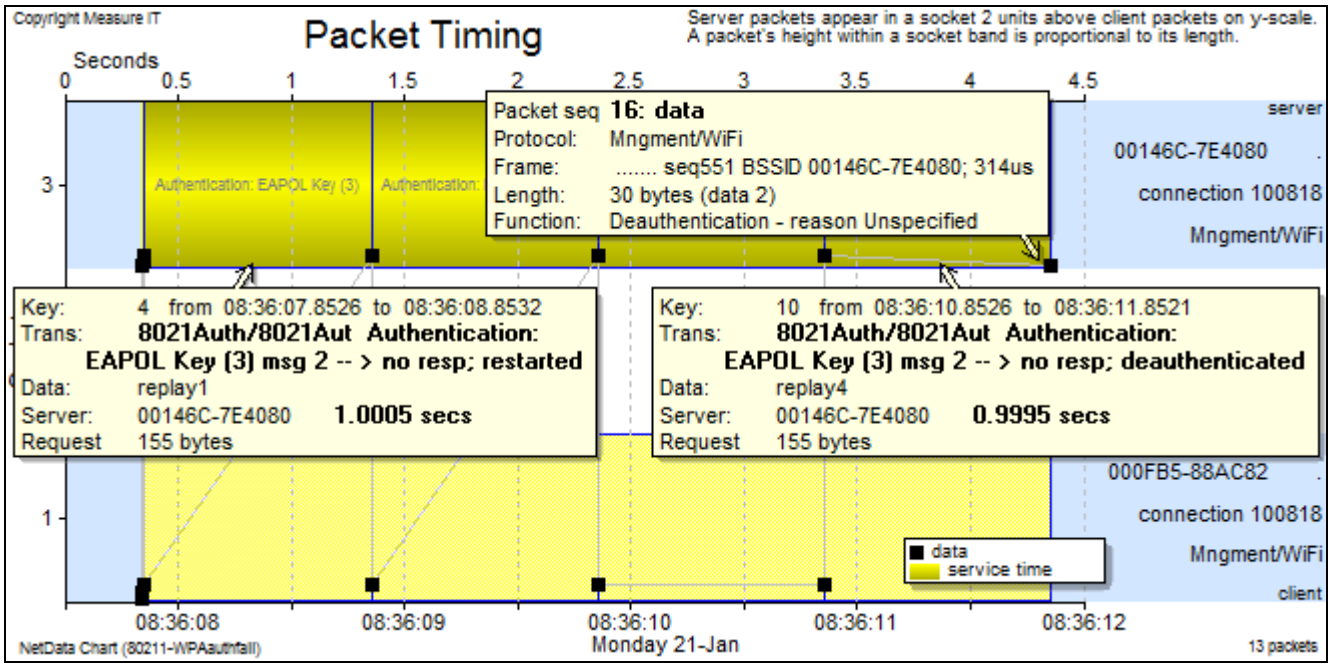
18.1 WiFi Management Traffic

NetData decodes WiFi (IEEE 802.11) management and control frames in detail, with descriptions of their common information elements, status codes and reason codes. Authentication frames and their acknowledging control frames are assigned common connection identifiers to ensure that they are plotted together on timing charts. Protected (i.e. encrypted) frames are labeled as '(protected)' and their data components are ignored. Management request-response pairs, and authentication round-trips, are characterised and plotted as transactions.

NetData reads Wireshark pcap files formatted with a link type of 105 (IEEE 802.11 Wireless LAN).



The above timing chart illustrates a WPA authentication sequence that eventually failed after four one-second timeouts:

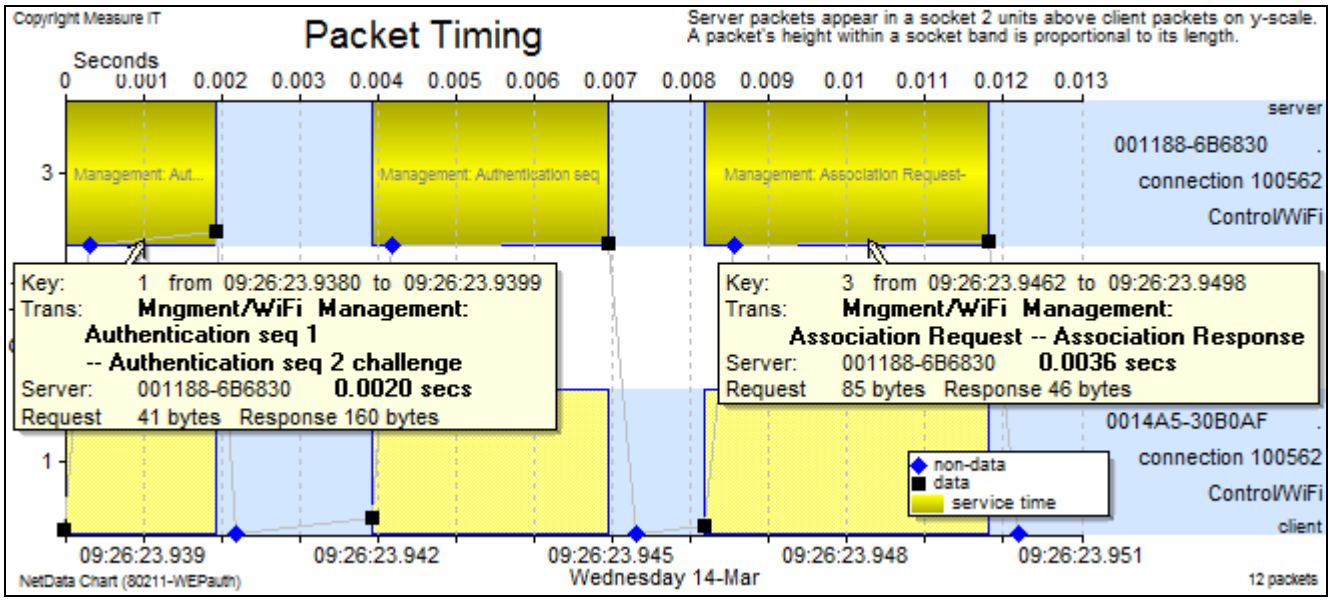


If a frame includes a Service Set ID (SSID) it is displayed in the Data column of the packet table. Other information conveyed in the WiFi frame headers is displayed in the 'Frame Description' column. Entries in this column start with a set of eight flags and list the sender's sequence number, the Basic Service Set Identifier (BSSID, usually the AP's MAC address), and a duration in microseconds.

	Time Of Day	Seq	Source	Destination	Len	Hdr	Frame Description	AppType	Data	Function
	09:26:23.800723	1	001188-6B6830	Local Broadcast	90	28	..... seq1387 BSSID 001188-6B6830	Mngment	62	Beacon
■	09:26:23.937975	2	0014A5-30B0AF	001188-6B6830	45	28	..... seq42 BSSID 001188-6B6830; 314us	Mngment	17	Authentic
◆	09:26:23.938313	3		0014A5-30B0AF	14	14	.....	Control		Ack
■	09:26:23.939938	4	001188-6B6830	0014A5-30B0AF	164	28	..... seq1388 BSSID 001188-6B6830; 314us	Mngment	136	Authentic
◆	09:26:23.940187	5		001188-6B6830	14	14	.....	Control		Ack
■	09:26:23.941938	6	0014A5-30B0AF	001188-6B6830	183	28	.p..... seq43 BSSID 001188-6B6830; 314us	Mngment	155	Authentic
◆	09:26:23.942187	7		0014A5-30B0AF	14	14	.....	Control		Ack
■	09:26:23.944938	8	001188-6B6830	0014A5-30B0AF	34	28	..... seq1389 BSSID 001188-6B6830; 314us	Mngment	6	Authentic
◆	09:26:23.945312	9		001188-6B6830	14	14	.....	Control		Ack
■	09:26:23.946188	10	0014A5-30B0AF	001188-6B6830	89	28	..... seq44 BSSID 001188-6B6830; 314us	Mngment	61	Associati
◆	09:26:23.946562	11		0014A5-30B0AF	14	14	.....	Control		Ack
	09:26:23.949189	12	001188-6B6728	Local Broadcast	88	28	.p....Fr seq1390 BSSID 001188-6B6830	Data	60	(protecte
■	09:26:23.949813	13	001188-6B6830	0014A5-30B0AF	50	28	..... seq1391 BSSID 001188-6B6830; 314us	Mngment	22	Associati
◆	09:26:23.950187	14		001188-6B6830	14	14	.....	Control		Ack
	09:26:24.634	15	0014A5-30B0AF	001188-6B6830	28	28	.....To seq45 BSSID 001188-6B6830; 44us	Data		null
	09:26:24.634098	16		0014A5-30B0AF	14	14	.....	Control		Ack
	09:26:24.634151	17	0014A5-30B0AF	Local Broadcast	386	28	.p....To seq46 BSSID 001188-6B6830; 44us	Data	358	(protecte

When raised, the eight frame-control flags are displayed with the following characters:

`O.......`	Strictly **Ordered**
`.p......`	Data is **Protected** (i.e. encrypted)
`..D.....`	**More Data** is buffered
`...u....`	**Under Power Management** (i.e. not staying up)
`....R...`	A **Retry** (charted as a retransmission)
`.....m..`	**More Fragments** remain
`......Fr.`	**From DS** (Distribution System) to STA (Station) via AP
`......To`	**To DS** from STA via AP



18.2 Radiotap WiFi Decoding

NetData decodes frames in Wireshark files with a link-type of 127. They are WiFi frames with a standard WiFi header prefixed with a Radiotap header that can convey many fields of frame metadata including frequency, signal strength and noise level. A brief summary of this header's contents begins with a length indicator such as 'Rtap[18]' and is appended to the WiFi header summary in the Frame Description column of the packet table, as in

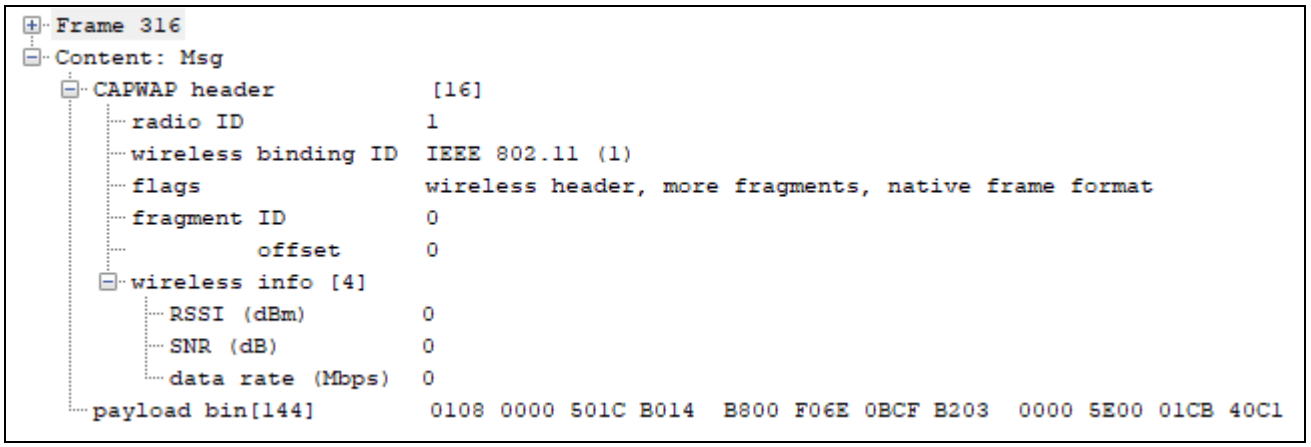
```
seq650 ....Retry.Fr BSSID DC9FDB-715CE3; 84us TID1
Rtap[18] WEP, FCS  6.5Mbps 2462MHz 2GHz, DynCCK-OFDM
         Signal-62dBm Noise-100dBm Qlty65
```

Some of the eight frame-control flags in a WiFi header are now displayed with enlarged acronyms rather than single characters:

`O.......`	Strictly **Ordered**
`.P......`	Data is **Protected** (i.e. encrypted)
`..Dbuf.....`	**More Data** is buffered
`...Sleep....`	**Under Power Management** (i.e. not staying up)
`....Retry...`	A **Retry** (charted as a retransmission)
`.....MoreFrag..`	**More Fragments** remain
`......Fr.`	**From DS** (Distribution System) to STA (Station) via AP
`......To`	**To DS** from STA via AP

18.3 Control And Provisioning of Wireless Access Points (CAPWAP)

The CAPWAP data protocol (Control And Provisioning of Wireless Access Points) encapsulates the complete contents of WiFi packets in UDP packets and is often seen in Fortinet firewalls that provide a bridge between WiFi access points. NetData decodes the first part of the UDP payload which provides metadata for the radio packet, as in the following description:



The metadata is usually followed by a complete radio packet, starting with a frame-control field, four wireless addresses, an LLC header and an IP header. NetData doesn't yet decode the radio packet in this context.

The CAPWAP data protocol uses port 5247, and a CAPWAP control protocol uses port 5246. The messages of both protocols begin with a one-byte preamble and three bytes reserved for flags. The remainder of control messages usually form a DTLS (Datagram Transport Layer Security) record.

19 Routing Protocols

19.1 BGP Decoding

NetData's BGP (Border Gateway Protocol) decoder describes BGP version-4 packets with either 2- or 4-byte Autonomous System (AS) numbers:

```
Open [58]
    BGP version              4
    My Autonomous System     23456 (AS_TRANS)
    Hold time                180 secs
    BGP speaker ID           IP=10.0.0.1
  Optional parameters [29]
    Capability [6]
      Multiprotocol Extension (1)[4]
    Capability [2]
      Cisco Route Refresh (128)[0]
    Capability [2]
      Route Refresh (2)[0]
    Capability [3]
      Cisco Multisession BGP (131)[1]: 00h
    Capability [6]
      Support for 4-octet AS numbers (65)[4]; my ASN: 10.1=655361
```

This Update message describes a path twice, with 2- and 4-byte AS numbers:

```
Route Update [58]
  Path attributes [33]
    attibute              flags                                  value
    ---------------------------------------------------------------------
    Origin              (complete, transitive, well-known)  from interior (IGP)
    AS4 Path            (complete, transitive, optional)
      AS Sequence [2]:  10.1=655361  40.1=2621441
    AS Path             (complete, transitive, well-known)
      AS Sequence [2]:  23456 (AS_TRANS)  23456 (AS_TRANS)
    Next Hop            (complete, transitive, well-known)  IP=172.16.3.1
  Network Layer Reachability Info (NLRI) [2]    IP=40.0.0.0/8
```

This Route Refresh Message has two Outbound Route Filter **(O**RF) entries with an ORF type of 128 (Cisco Prefix List):

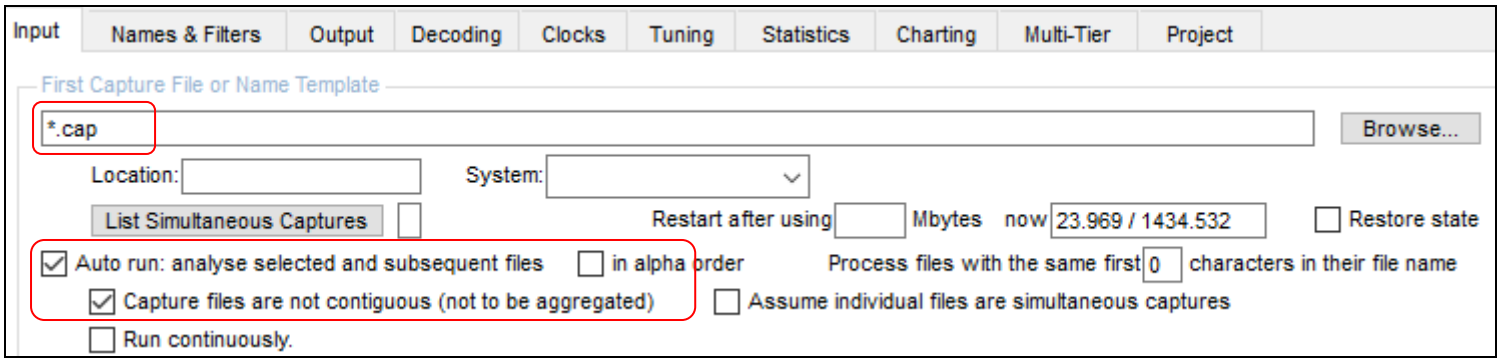
```
Keepalive [19]
Route Refresh [46]
    Adrs Family ID           IPv4 (1)
    Subsequent AFI           unicast forwarding (1)
    refresh                  Immediately
  Outbnd Route Filter:       Cisco Prefix List (128)
    Cisco Prefix List        Add, Deny matching updates
      sequence position 5
      prefix mask len.  24 - 24
      prefix            IP=1.1.0.0/21
    Cisco Prefix List        Add, Permit matching updates
      sequence position 10
      prefix mask len.  0 - 32
      prefix            IP=0.0.0.0/0
```

As in most message descriptions, numbers in parentheses indicate type codes, and numbers in square brackets indicate field lengths.

This release is accompanied by a collection of 18 capture files with only the packets of BGP connections. They can also be downloaded from a Wireshark site and from PacketLife.net:

http://packetlife.net/captures/protocol/bgp/

The easiest way to browse samples of these BGP messages is to analyse all the capture files in the one project whose input controls are configured as below:



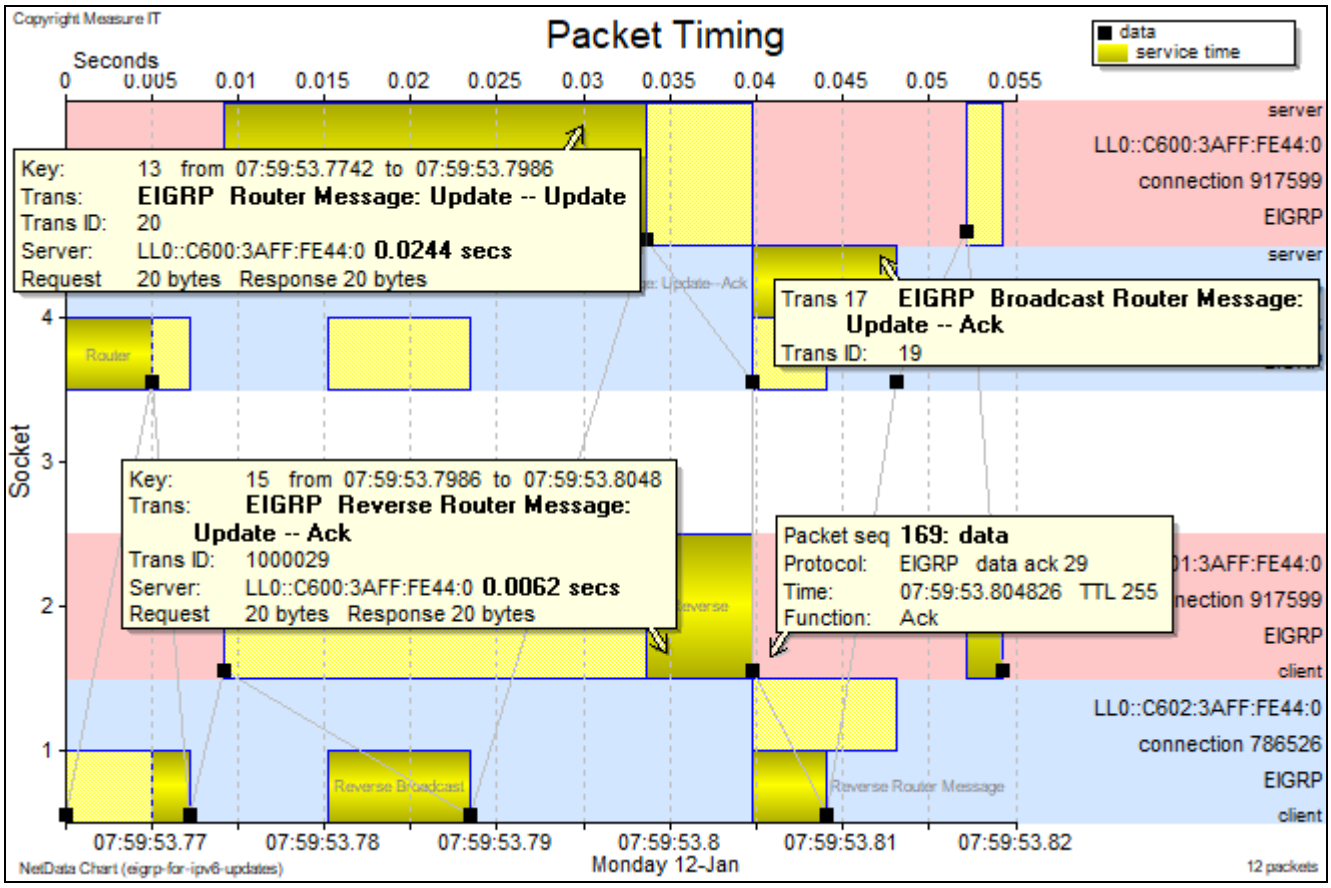
Load all the packets, view the packet table, and double click in the Content column of interesting packets.

19.2 EIGRP Decoding

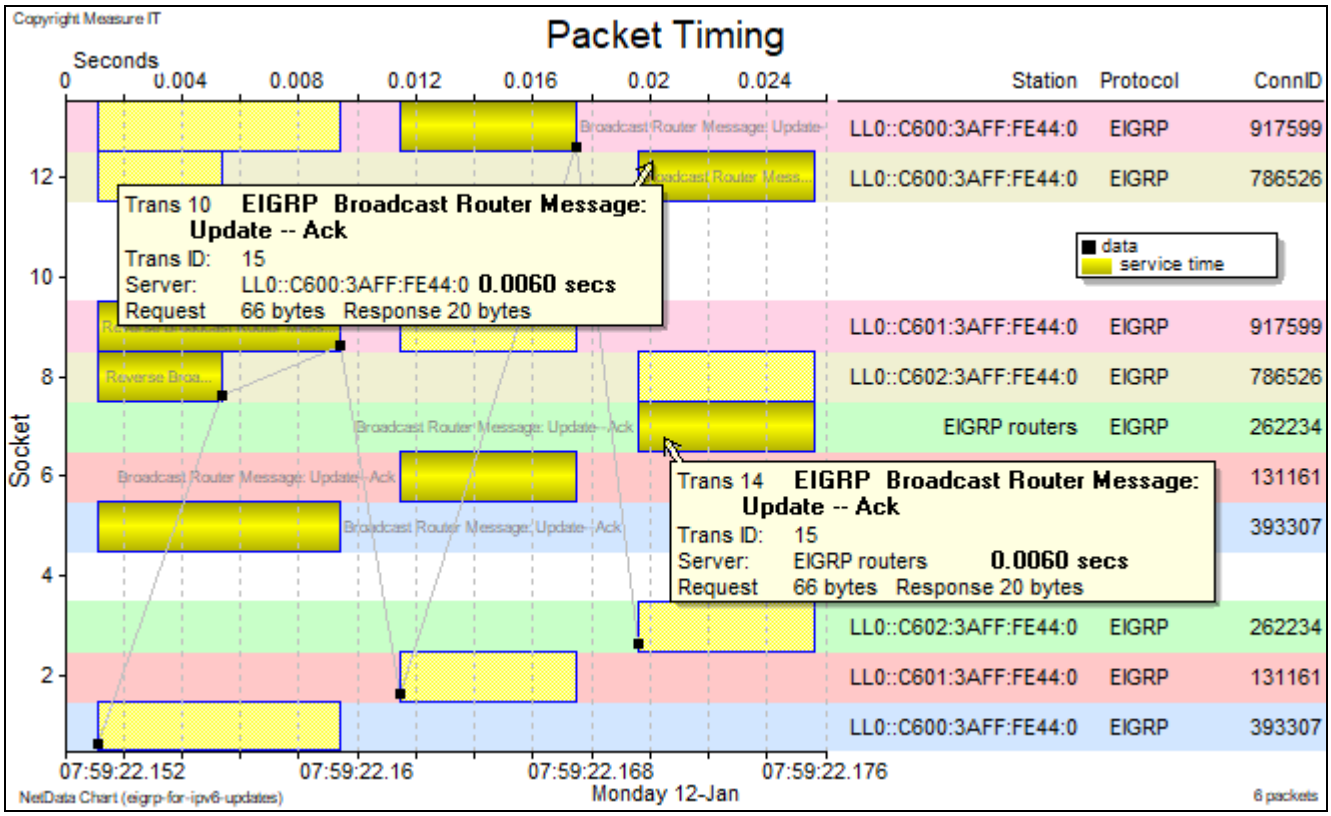
NetData decodes routing packets of EIGRP (Enhanced Interior Gateway Routing Protocol), a protocol designed by Cisco and described by RFC 7868.

EIGRPv2	Update
flags	EOT (End of Table) - all Updates completed
sequence	31
acknowledgment	20
Virtual Router ID	0 (unicast address family)
Autonomous System	666
IPv6 Internal [42]	
next hop	::0
Vector metric:	
delay	5.00 ms
bandwidth	10,000,000 Kbps
minimum path MTU	1514
hop count	0
error rate	fully reliable (FFh)
utilisation	1%
internal tag	0
flags	inactive
destination	IP=2001::1:1:0:0:0:0/64

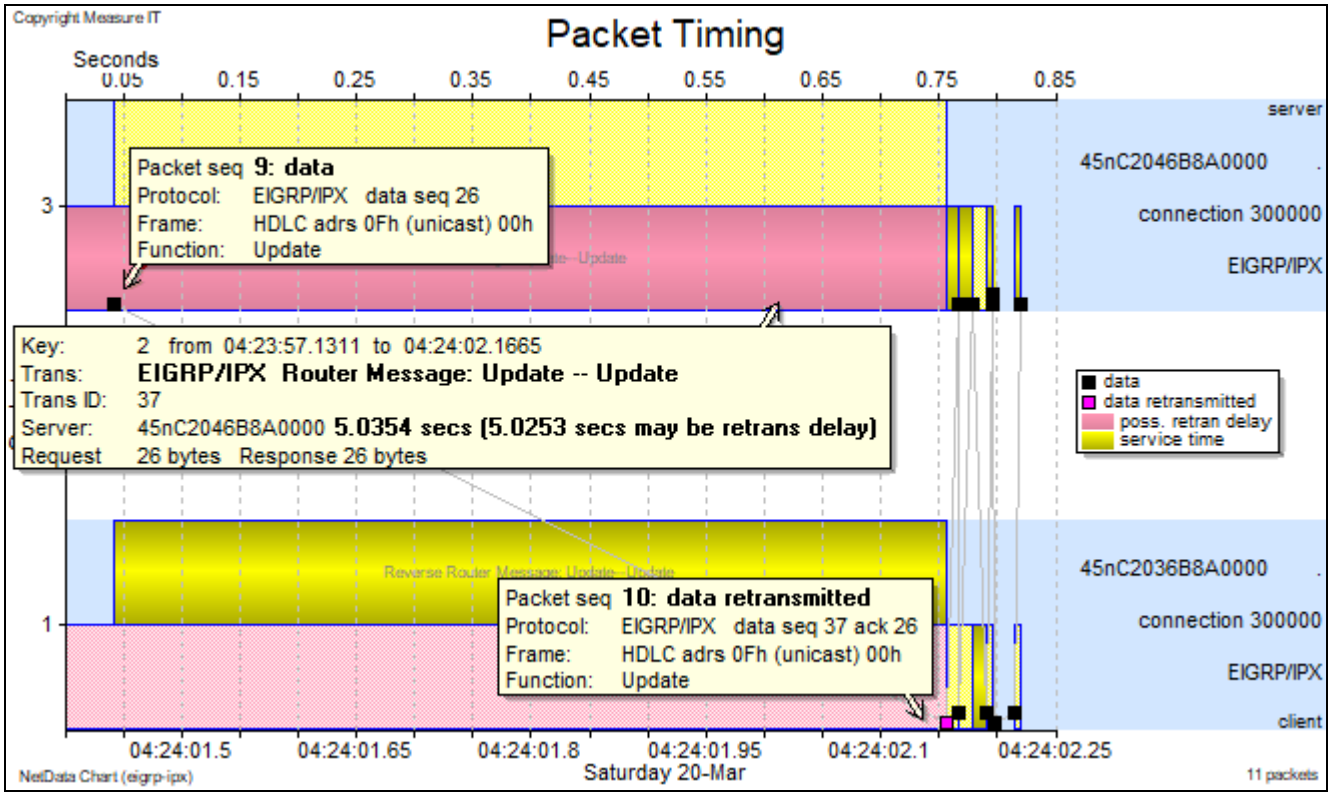
After matching message sequence and ack numbers, NetData measures round-trip times between router neighbours and plots them as transaction response times. Large times may indicate a bottleneck in the convergence of routing tables. Round-trips can originate from both the client and server.



NetData also measures and displays response times when several routers acknowledge an individual broadcast message. In the chart below, the first packet is an Update message broadcast from LL0::C600, and it is acknowledged with unicast messages from both LL0::C601 and LL0::C602. Transaction bars appear on both the unicast and multicast socket bands.



NetData highlights a retransmitted message in the chart below which identifies a router that stopped communicating for several seconds. EIGRP messages were carried in IPX packets transmitted over a Cisco HDLC link between routers.



19.3 IS-IS Decoding

NetData decodes packets of the IS-IS routing protocol (Intermediate System to Intermediate System). This OSI (Open Systems Interconnection) intra-domain routing protocol may be used as an interior gateway protocol (IGP) to support networks of TCP/IP as well as OSI; it is documented in RFC 1195 and the International Standard ISO/IEC 10589.

IS-IS packets use an LLC header with a Network Service Access Pont (NSAP) address of FEh which identifies the OSI network layer. The LLC header is followed by an intra-domain routing-protocol discriminator of 83h (an Initial Protocol Identifier defined in ISO/IEC TR9577) to specify IS-IS.

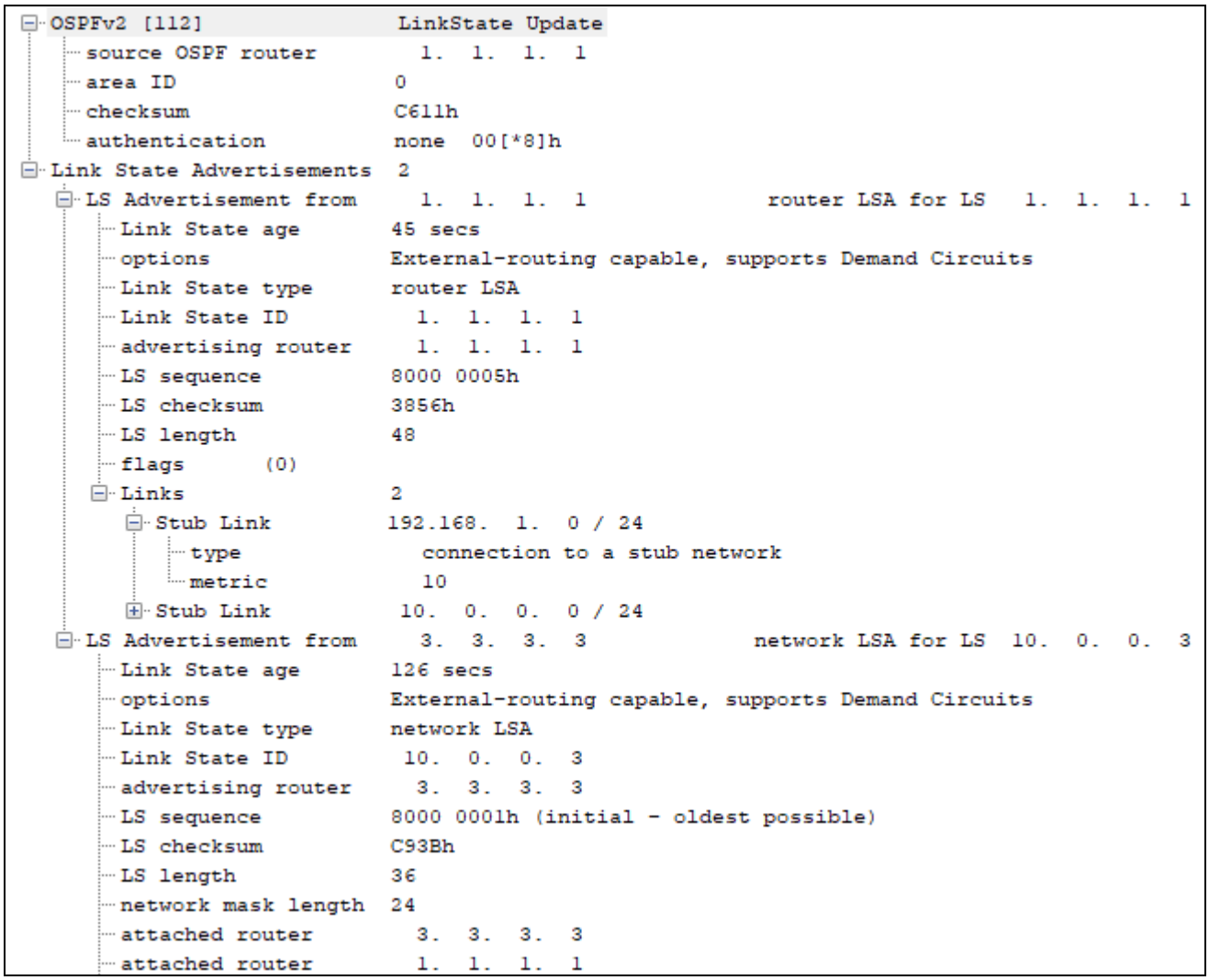
```
IS-IS PDU header [27]       L1 Hello (15)
    vsn, protocol ID extn   1
    version                 1
    max area addresses      0
    circuit type            Level 1
    source ID [0]           3333.3333.3333
    holding time            10 secs
    PDU length              1497
    priority                64
    designated LAN ID       3333.3333.3333.02
supported protocols (129)   CCh (IP)
area addresses (1) [4]      49.000A
IP interface address (132) 10.0.10.1
restart signalling (211)    (0)
IS neighbour (6) [6]        C20129-980000
```

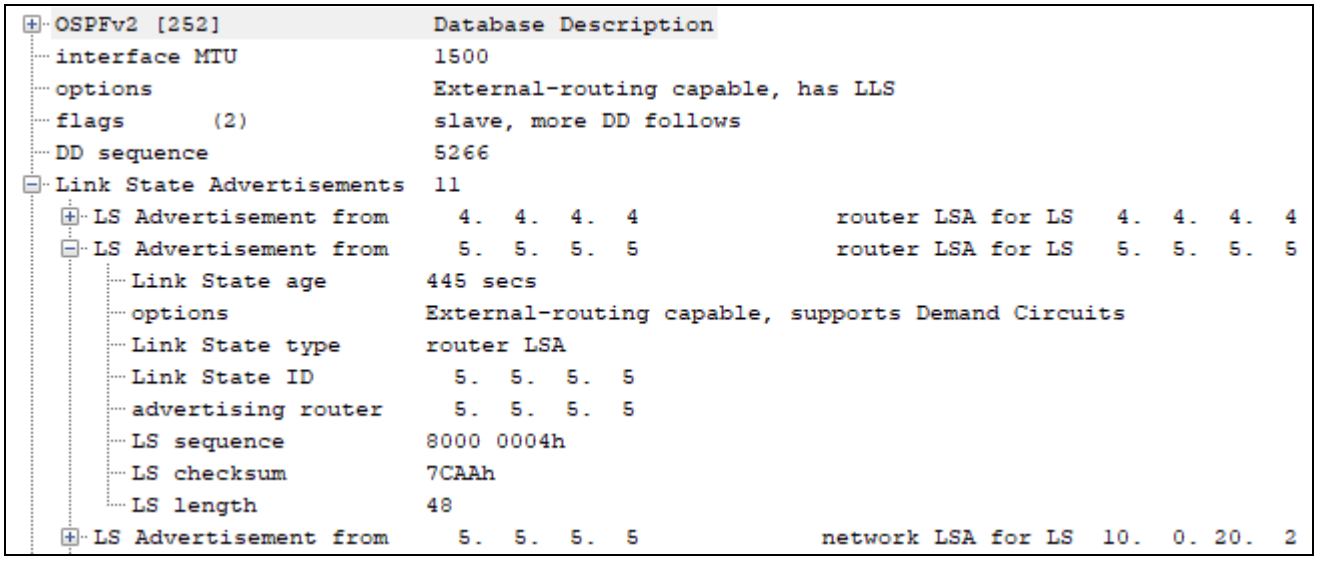
```
IS-IS PDU header [33]       L1 Complete Sequence Numbers (24)
    vsn, protocol ID extn   1
    version                 1
    max area addresses      0
    PDU length              83
    source ID               3333.3333.3333.00
    start LSP ID            0000.0000.0000.00-00
      end LSP ID            FFFF.FFFF.FFFF.FF-FF
LSP entries (9) [48]
  LSP entry:                2222.2222.2222.00-00
      remaining lifetime    1174 secs
      LSP ID                2222.2222.2222.00-00
      sequence              14
      checksum              22800
```

```
IS-IS PDU header [27]       L1 Link State (18)
area addresses (1) [4]      49.000A
supported protocols (129)   CCh (IP)
host name (137)[2]          R2
IP interface address (132) 192.168.10.1
IP internal reachability (128) [24]
  IPv4 prefix               10.0.10.0 / 30
      default metric        10 internal, distribution:down
        delay metric        0 not supported
      expense metric        0 not supported
        error metric        0 not supported
  IPv4 prefix               192.168.10.0 / 24
IS neighbours (2) [12]
  neighbour                 3333.3333.3333.02
      default metric        10 internal
```

19.4 OSPF Decoding

NetData's decoder for the routing protocol OSPF (Open Shortest Path First, RFC 2328) fully describes all LSAs (Link State Advertisements) in all types of packets, and covers OSPFv3 (RFC 5240) which handles IPv6.





```
OSPFv2 [400]                    LinkState Update
    source OSPF router            4.  4.  4.  4
    area ID                     20
    checksum                    D794h
    authentication              none  00[*8]h
Link State Advertisements       11
  LS Advertisement from           5.  5.  5.  5              router LSA for LS   5.  5.  5.  5
  LS Advertisement from           4.  4.  4.  4              router LSA for LS   4.  4.  4.  4
  LS Advertisement from           5.  5.  5.  5             network LSA for LS  10.  0. 20.  2
  LS Advertisement from           4.  4.  4.  4        summary (IP) LSA for LS 192.168. 10.  0
  LS Advertisement from           4.  4.  4.  4        summary (IP) LSA for LS  10.  0. 10.  0
  LS Advertisement from           4.  4.  4.  4        summary (IP) LSA for LS  10.  0.  0.  0
      Link State age            11 secs
      options                   External-routing capable, supports Demand Circuits
      Link State type           summary (IP) LSA
      Link State ID              10.  0.  0.  0
      advertising router          4.  4.  4.  4
      LS sequence               8000 0001h (initial - oldest possible)
      LS checksum               E03Bh
      LS length                 28
      network mask length       30
      metric                    10
  LS Advertisement from           4.  4.  4.  4      summary (ASBR) LSA for LS   2.  2.  2.  2
      Link State age            11 secs
      options                   External-routing capable, supports Demand Circuits
      Link State type           summary (ASBR) LSA
      Link State ID               2.  2.  2.  2
      advertising router          4.  4.  4.  4
      LS sequence               8000 0001h (initial - oldest possible)
      LS checksum               6FA0h
      LS length                 28
      network mask length       0
      metric                    20
  LS Advertisement from           2.  2.  2.  2         AS external LSA for LS 172. 16.  3.  0
      Link State age            197 secs
      options                   supports Demand Circuits
      Link State type           AS external LSA
      Link State ID             172. 16.  3.  0
      advertising router          2.  2.  2.  2
      LS sequence               8000 0001h (initial - oldest possible)
      LS checksum               2860h
      LS length                 36
      network mask length       24
      Type 1 metric             100
      forwarding address        none
      extern route tag          0
  LS Advertisement from           2.  2.  2.  2         AS external LSA for LS 172. 16.  2.  0
  LS Advertisement from           2.  2.  2.  2         AS external LSA for LS 172. 16.  1.  0
  LS Advertisement from           2.  2.  2.  2         AS external LSA for LS 172. 16.  0.  0
```

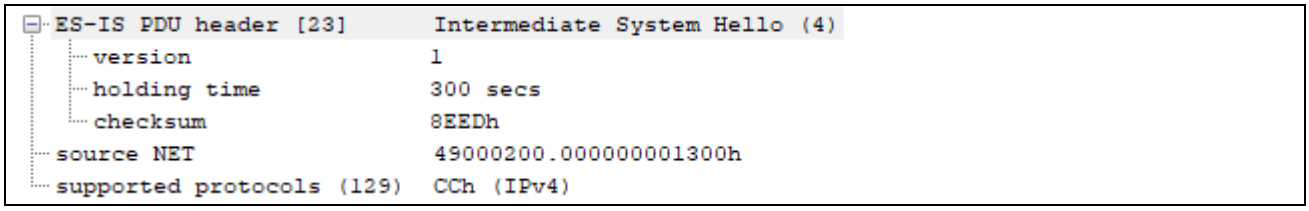
```
OSPFv3 [288]                    LinkState Update
Link State Advertisements       7
  LS Advertisement from           1.  1.  1.  1              router LSA for LS 0
  LS Advertisement from           1.  1.  1.  1   inter-area-prefix LSA for LS 3
  LS Advertisement from           1.  1.  1.  1   inter-area-prefix LSA for LS 2
  LS Advertisement from           1.  1.  1.  1   inter-area-prefix LSA for LS 1
  LS Advertisement from           1.  1.  1.  1   inter-area-prefix LSA for LS 0
  LS Advertisement from           1.  1.  1.  1                link LSA for LS 5
  LS Advertisement from           1.  1.  1.  1   intra-area-prefix LSA for LS 0
```

19.5 ES-IS Decoding

NetData decodes packets of the ES-IS routing protocol (End System to Intermediate System). This protocol is documented in the International Standard ISO 9542.

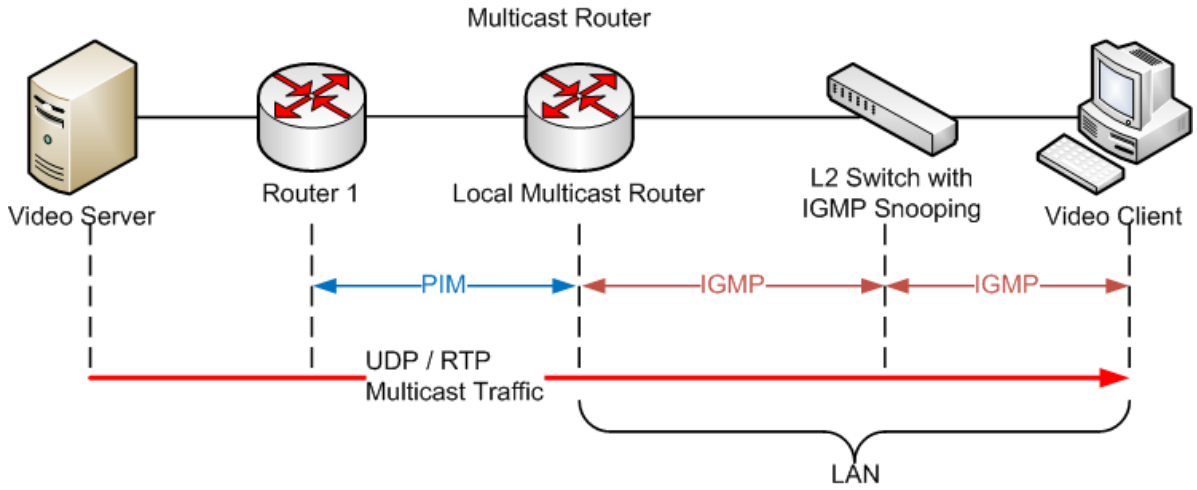
Both ISO routing protocols, ES-IS and IS-IS, allow devices to have a variable-length address known as a Network Entity Title (NET). These addresses range from 8 octets to 20 octets in length and are displayed in hexadecimal form. Generally, the format consists of an Authority and Format Identifier (AFI), a domain ID, an area ID, a system identifier, and a selector. The simplest format omits the domain ID and is 10 octets long.

The system identifier must be unique within the network and in an IP network may be assigned the device's IP address.



19.6 IGMP and Protocol Independent Multicast (PIM)

NetData's decoding of IGMP (Internet Group Management Protocol) has been updated. IGMP is an integral part of IP multicast, used by hosts and adjacent routers to establish multicast group memberships. Wikipedia provides the following diagram to illustrate the architectural relationship between IGMP and PIM (Protocol Independent Multicast):



The router data blocks in multicast traceroute ('mtrace') packets are presented in tabular form:

ArrivalTime	Incoming I/F	Outgoing I/F	Previous Router	InputPkts	OutputPkts	TotalPkts	Protocol	TTL	Mask	Fwding Code
1194083740	10. 0. 0. 14	10. 0. 0. 14	10. 0. 0. 13	242	0	0	PIM	0	24	NO_ERROR
1194049400	10. 0. 0. 6	10. 0. 0. 13	10. 0. 0. 5	240	0	0	PIM	0	24	NO_ERROR

PIM is a multicast routing protocol that can use the underlying unicast routing-information base or a separate multicast-capable routing-information base. It operates in either of two distinct modes – Sparse Mode (SM) or Dense Mode (DM) – that is chosen to best suit the density of receivers in the network. PIM is specified in RFCs 3973, 4061 and 5015.

NetData's decoding of PIM messages has been updated and extended to decapsulate the original packets in PIM-SM Register packets as in the following example. The first of the two packets is a demonstration ICMP Ping addressed to the private IP multicast address 239.1.2.3. A PIM procedure in the router at 192.168.0.6 – the sender's Designated Router (DR) – has encapsulated that packet, sending it as a unicast packet to a router at 192.168.1.254 that acts as a Rendezvous Point (RP) and can copy the packet for the intended receivers. The process of encapsulating data packets to the RP is called *registering*, and the encapsulation packets are known as PIM Register packets.

As it does when decapsulating any packet, NetData confines information from the outer header to the Frame Description column and passes only the original packet to the analysis and charting module.

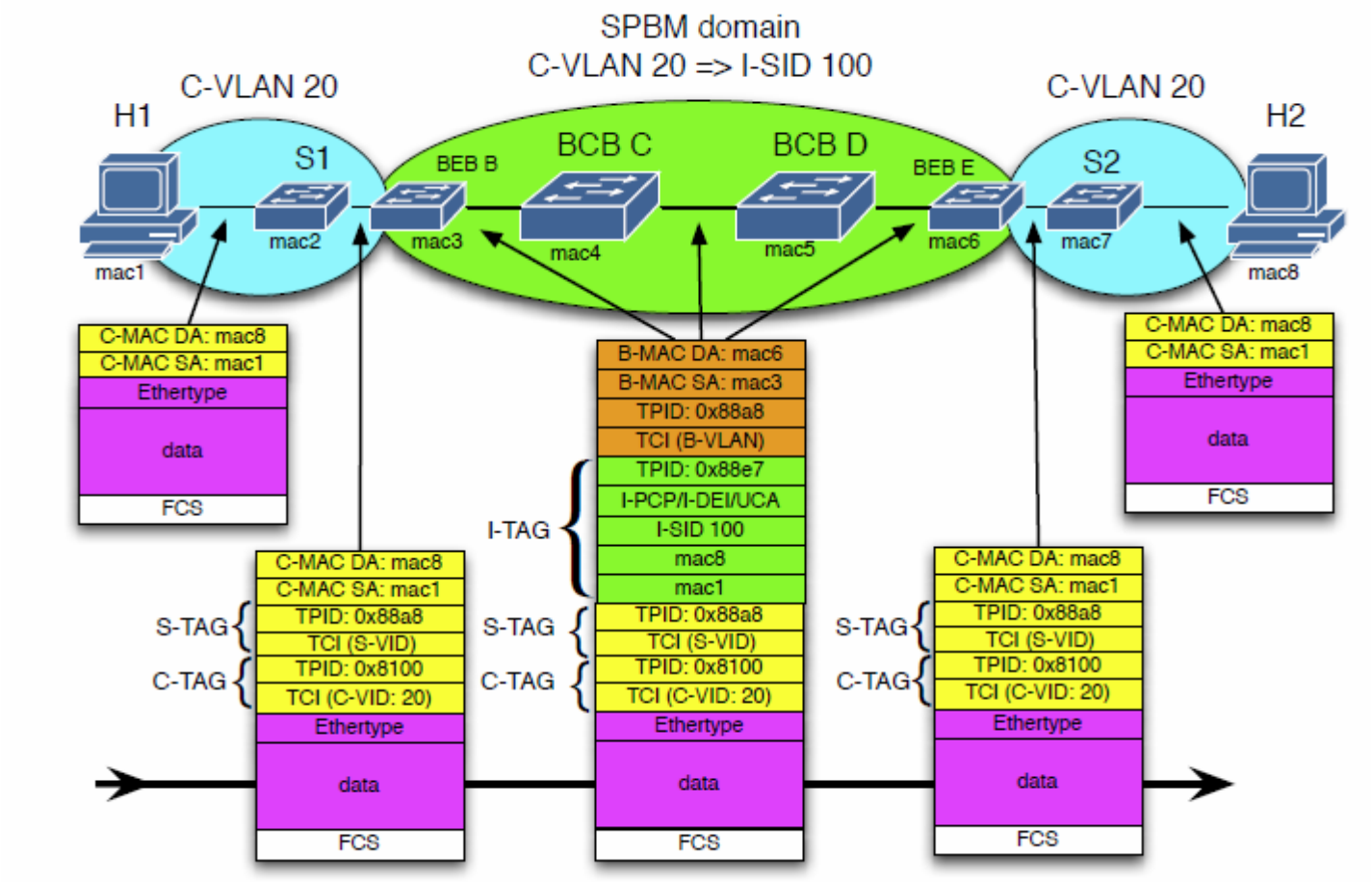
Source	Destination	Len	Frame Description	Net	Tspt	App...	Function
192.168.20.10	239.1.2.3	146	PIMv2 Register, outer IP 192.168.0.6(DR)->(RP)192.168.1.254	PIMv2/IP4	ICMP4	Ping	Echo Request
192.168.1.254	192.168.0.6	64		IP4	PIMv2	PIMv2	Register-Stop

If the RP decides that encapsulation is inappropriate – perhaps because the RP is further from the sender than the receivers are from the sender – it initiates a source-specific Join towards the sender. Eventually the Join will cause the source to send packets in two forms – natively and encapsulated – and when the RP receives a packet in both forms it sends a Register-Stop packet to the DR, as exemplified by the second of the two packets above.

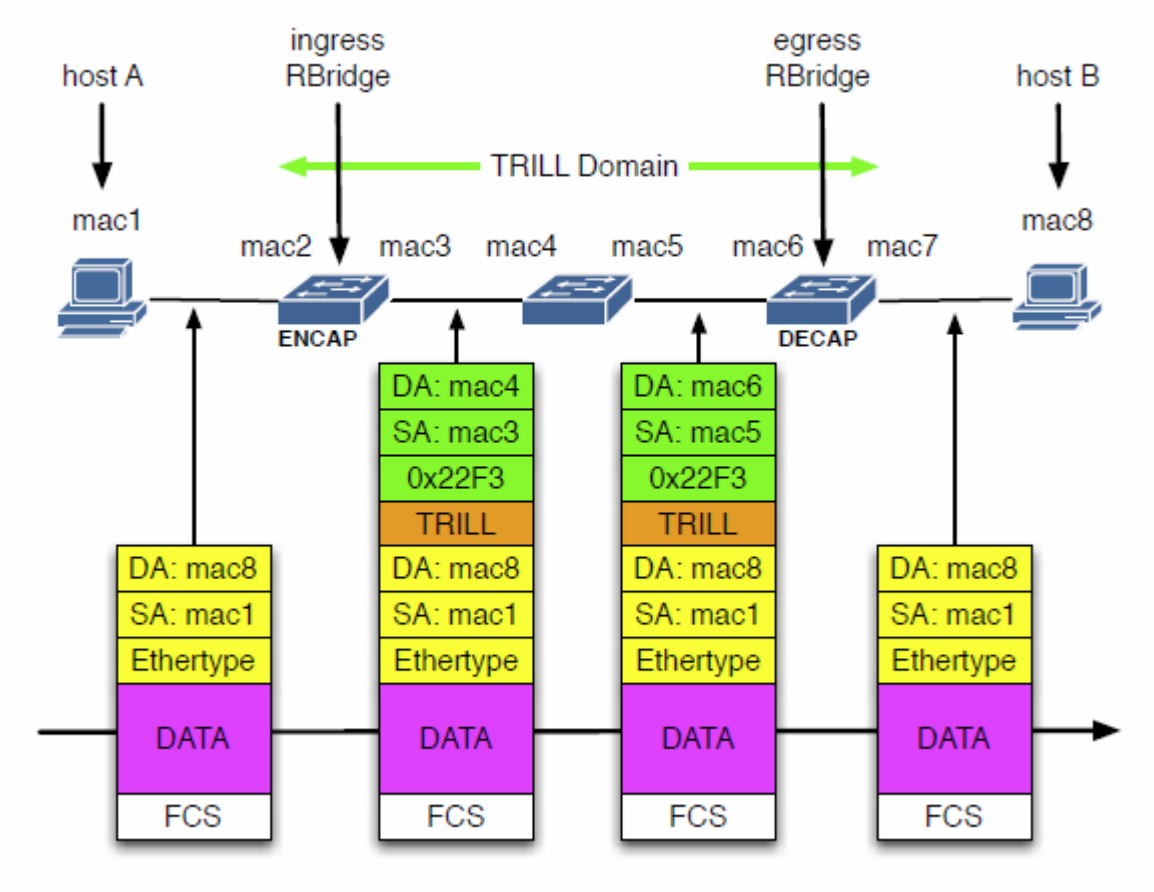
19.7 Shortest Path Bridging (SPB) and TRILL

IEEE and IETF (Internet Engineering Task Force) standardised two different schemes to overcome the limitations of the Spanning Tree Protocol (STP) and allow multipath routing, without loops, over extended layer-2 networks. The IEEE solution is Shortest Path Bridging (SPB) documented in IEEE 802.1aq and RFC 6329. The IETF solution is TRansparent Interconnection of Lots of Links (TRILL) which is covered by several RFCs including 6325, 6326 and 6165.

The two schemes have much in common. They can establish multiple (equal-cost) paths for a *unicast* stream, or multiple trees for a *multicast* stream, across a layer-2 network. Identifiers for the chosen routes are carried by data packets in tags that are like conventional (IEEE 802.1Q) VLAN tags inserted between a packet's MAC addresses and its payload. The new tags, together with new MAC addresses for routing across the layer-2 network, constitute an outer header that is prepended to the original packet. Changes in the assembly of headers as packets traverse the network are illustrated in the following two diagrams from the article *TRILL and IEEE 802.1aq Overview*, by Ronald van der Pol, Draft Version 1.2, April 2012, at https://kirk.rvdp.org/publications/TRILL-SPB.pdf



In this example C-VLAN 20 (Customer VLAN) is mapped to I-SID 100 (Instance Service ID). The C-VLAN ID (VID) is carried in a C-TAG with Ethertype 8100h, and the I-SID is carried in an I-TAG (Instance tag) with Ethertype 88E7h. Service and Backbone VLAN IDs are carried in S-TAGs and B-TAGs with Ethertype 88A8h.



With Ethertype 22F3h, TRILL inserts a header that identifies both an ingress and an egress RBridge (Routing Bridge), the two devices responsible for encapsulating and decapsulating the original packet at the edge of the layer-2 network.

In both SPB and TRILL, routing information is exchanged between switches in messages of the IS-IS routing protocol. Both schemes extend IS-IS command messages with different TLVs (Type, Length, Value fields) and sub-TLVs. IS-IS is readily extensible by adding new types of TLVs, and because an individual TLV can associate many different attributes. In an individual switch, the resources of different routing schemes and topologies can be handled by different instances of IS-IS. A TRILL instance is identified by Ethertype 22F4h, whereas all SPB instances are identified by the ISO NLPID 193 (C1h) and an Instance ID conveyed in a TLV. Basic IS-IS (Instance zero) has NLPID 131 (83h). The functions of all these Ethertypes and NLPIDs are summarised in NetData's table of recognisable protocols.

NetData recognises all IS-IS TLVs and fully decodes most of them. For data packets it decodes all the encapsulation headers and tags with Ethertypes 22F3h, 8100h, 88A8h and 88E7h. In keeping with its analysis objective to demultiplex packet streams as far as possible, to characterise individual transactions, NetData strips off the outer headers in order to forward only the original packet contents to its analysis and charting engine. Outer MAC addresses and the contents of outer headers and tags are described in the Frame Description column of the packet table.

19.8 MPLS (MultiProtocol Label Switching)

MultiProtocol Label Switching (MPLS) allows efficient access to routing tables as packets traverse a network – it inserts short numeric labels in packets to identify their routes. Each packet carries a stack of one or more four-byte headers inserted just below its IP header, and each MPLS header includes a 20-bit label and a one-byte TTL (hop) count.

NetData decodes MPLS headers and records their contents in the frame-description column of the packet table.

19.9 Pseudo-Wire (PW) Emulation in MPLS Networks

Pseudo Wire Emulation Edge-to-Edge (PWE3) is a mechanism that emulates the essential attributes of a service such as ATM, Frame Relay or Ethernet over a packet-switched network (see RFC 3916 and RFC 3985). Encapsulation of pseudo-wire (PW) packets in an MPLS network requires a 4-byte control word (CW) to be appended to the MPLS tag (RFC 4385), to indicate that the tag is *not* followed by an IP header as is normally the case. The control word's first nibble must avoid a value of 4 or 6, the version number in an IP header. A CW nibble value of 0 indicates a PW *data* packet – such as an encapsulated Ethernet packet that begins with another pair of MAC addresses – and a value of 1 indicates a PW *associated channel* (ACh) that conveys PW management information.

NetData now handles both types of PW control words. For a PW data packet, the outer MAC addresses and the control word's sequence number, if non-zero, are displayed in the Frame Description column of the packet table. By analysing bit patterns in the encapsulated packet's link header following the control word, NetData can distinguish Frame Relay from Ethernet packets.

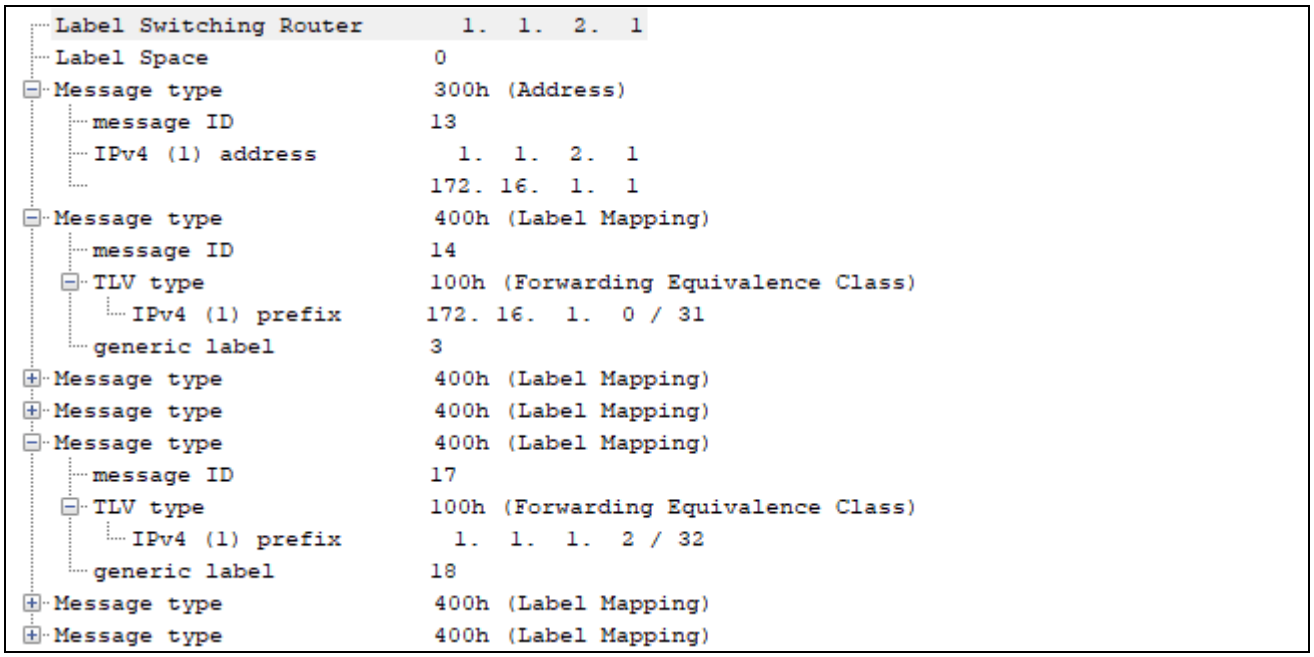
If the control word indicates an associated channel of type 1 or 2, for a Management or Signalling Communication Channel (MCC or SCC), NetData decodes the payload according to the protocol specified in the subsequent two bytes. Otherwise, NetData simply displays the channel type and the raw data of the packet payload.

19.10 Label Distribution Protocol (LDP) for MPLS Networks

NetData now decodes messages of the Label Distribution Protocol (LDP). Procedures of this protocol are used by Label Switching Routers (LSRs) to agree on the meaning of labels they bind to packets in an MPLS (MultiProtocol Label Switching) network. LDP is specified in RFC 5036.

LDP associates a Forwarding Equivalence Class (FEC) with each Label Switched Path (LSP) that is created by a pair of LSRs, and the FEC specifies which packets are 'mapped' to the path. Another LDP function is to manage the mapping of labels to FECs.

An LDP packet usually conveys a single PDU (Protocol Data Unit) which consists of one or more LDP messages. Each message has a set of mandatory parameters transmitted in a fixed order, and a set of optional parameters that are formatted as TLVs (Type, Length, Value fields) and transmitted in any order. Hello messages, transported by UDP, are used by LSRs to discover peers. All other types of messages are transported over TCP connections, to initialise and conduct sessions between LSR peers.


Label Switching Router 1. 1. 2. 1
Label Space 0
Message type 300h (Address)
message ID 13
IPv4 (1) address 1. 1. 2. 1
172. 16. 1. 1
Message type 400h (Label Mapping)
message ID 14
TLV type 100h (Forwarding Equivalence Class)
IPv4 (1) prefix 172. 16. 1. 0 / 31
generic label 3
Message type 400h (Label Mapping)
Message type 400h (Label Mapping)
Message type 400h (Label Mapping)
message ID 17
TLV type 100h (Forwarding Equivalence Class)
IPv4 (1) prefix 1. 1. 1. 2 / 32
generic label 18
Message type 400h (Label Mapping)
Message type 400h (Label Mapping)

20 OSI Protocols

20.1 LLC OSI Network Layer

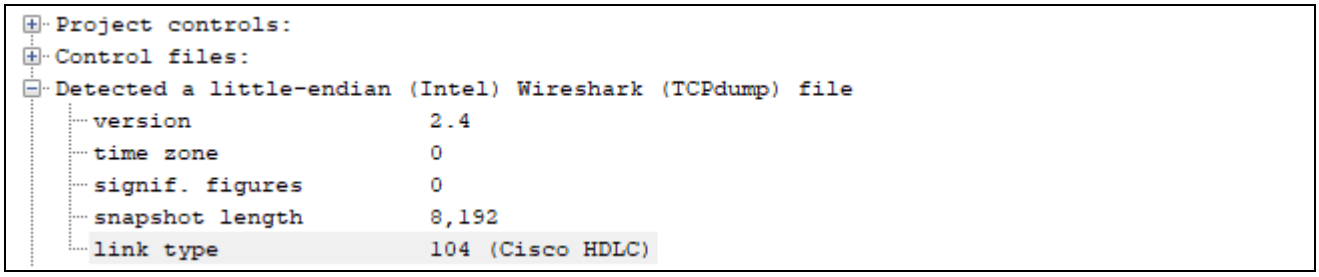
The PDUs of ISO protocols designed to control connectionless-mode networks are usually preceded by an LLC header with a DSAP (Destination Service Access Point) of 254 (FEh). The DSAP is followed by a one-byte protocol discriminator called the Network Layer Protocol ID (NLPID), and NetData recognises all the NLPIDs in the following table:

Name	Description	Decoder	Transport	IP	DSAP	NLPID
Q.933	ITU-T Q.933 messages for control and status monitoring over frame-relay permanent virt...	d	LLC, ISO		FEh	8h
Q2933LMI	Cisco Local Management Interface (LMI) for signalling between routers and frame-relay s...	d	LLC, ISO		FEh	9h
IEEE_SNAP	SubNetwork Access Protocol on ISO network layer	d	LLC, ISO		FEh	80h
CLNP	Connectionless Network Protocol (ISO 8473)		LLC, ISO		FEh	81h
ES-IS	OSI routing protocol, End System to Intermediate System (ISO 9542)	d	LLC, ISO		FEh	82h
IS-IS	OSI intra-domain routing protocol, Intermediate System to Intermediate System (RFC 119...	d	LLC, ISO		FEh	83h
IPv6	Internet Protocol v6 on ISO network layer	d	LLC, ISO		FEh	8Eh
FRF.9	Frame Relay Forum Agreement 9		LLC, ISO		FEh	B0h
FRF.12	Cisco Voice over Frame Relay (VoFR) protocol that allows large data frames to be fragm...		LLC, ISO		FEh	B1h
TRILL	TRansparent Interconnection of Lots of Links (TRILL) to create large Layer-2 clouds wit...		LLC, ISO		FEh	C0h
SPB	Shortest Path Bridging (SPB), a configuration protocol to avoid network loops while enab...	d	LLC, ISO		FEh	C1h
IPv4	Internet Protocol v4 on ISO network layer	d	LLC, ISO		FEh	CCh
PPP	Point-to-Point Protocol on ISO network layer	d	LLC, ISO		FEh	CFh

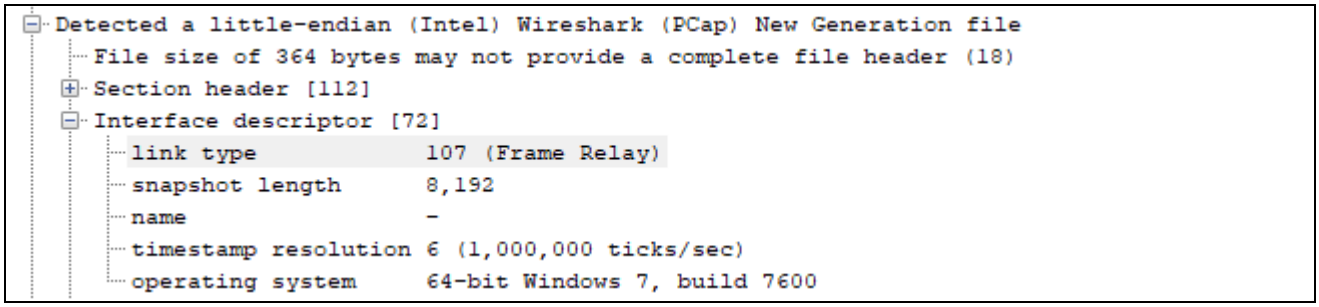
NetData fully decodes those protocols with a ‘d’ in the above Decoder column.

20.2 Cisco HDLC and Frame Relay Frame Formats

Some Cisco routers exchange routing information over serial links which support network and higher-layer protocols with a simplified version of the HDLC (High-level Data Link Control) protocol. In effect these packets replace an Ethernet header with just two bytes: an address byte which is either 0Fh (unicast) or 8Fh (multicast); and a zero control byte. If captured by Wireshark, the capture file indicates a link type of 104. NetData now analyses packets of this link type:



NetData now also analyses frame-relay packets recorded with a link type of 107. Their link headers specify a DLCI (Data Link Connection ID) that usually needs only two bytes but can extend up to four bytes for large DLCIs. Three flags in the header serve congestion-control purposes.



The link headers of both frame formats are summarised in the Frame Description column of the packet table. The frame format itself is indicated by `CLC` or `FrR` in the Frame Type column.

20.3 Serial Line Address Resolution Protocol (SLARP) and Inverse ARP

When communication is first established over a serial link a host may need to determine the IP address of the host or subnet at the remote end. On a frame-relay link the process involves Inverse-ARP request and reply packets of the Address Resolution Protocol (ARP), a protocol that is specified by an Ethertype of 0806h. In those packets the hardware address refers to the DLCI (Data Link Connection ID) of a PVC (Permanent Virtual Connection) supported by the link.

This frame-relay inverse-ARP request from 10.0.0.1 is for the IP address associated with DLCI 103:

```
Inverse Request
    Address          Hardware    Protocol
    ---------------------------------------
    type     Frame Relay DLCI   800h (IPv4)
    sender          0 (0000h)   10.  0.  0.  1
    target        103 (1871h)    0.  0.  0.  0
data type 1 [2]               0040
```

The query was resolved by this inverse-ARP reply from 10.0.0.3:

```
Inverse Reply
    Address          Hardware    Protocol
    ---------------------------------------
    type     Frame Relay DLCI   800h (IPv4)
    sender          0 (0000h)   10.  0.  0.  3
    target        103 (1871h)   10.  0.  0.  1
data type 1 [2]               0040
```

A Cisco HDLC link uses a serial-line version of the Reverse Address Resolution Protocol (RARP) which has an Ethertype of 8035h. The Cisco protocol, with the same Ethertype, is known as SLARP. A SLARP Reply packet contains both an IPv4 address and a 32-bit mask to define a subnet. NetData displays the mask information as a number of bits in the subnet address.

```
Frame 2
Content: Request
    Request (0)                0.  0.  0.  0 / 0
```

```
Frame 3
Content: Reply
    Reply (1)                 15.  0.  0.  1 / 30
```

In addition to Request and Reply packets, SLARP specifies a Keep-alive packet for sending at regular intervals a pair of sequence numbers: a *transmission* sequence number; and the sequence number that was in the last received keep-alive packet.

```
Frame 3
Content: Keepalive
    Keepalive (2)           TX seq 486        RX seq 485
    reliability check       FFFFh (good)
    extra data [6]          054D E484 0002
```

21 Industrial Protocols

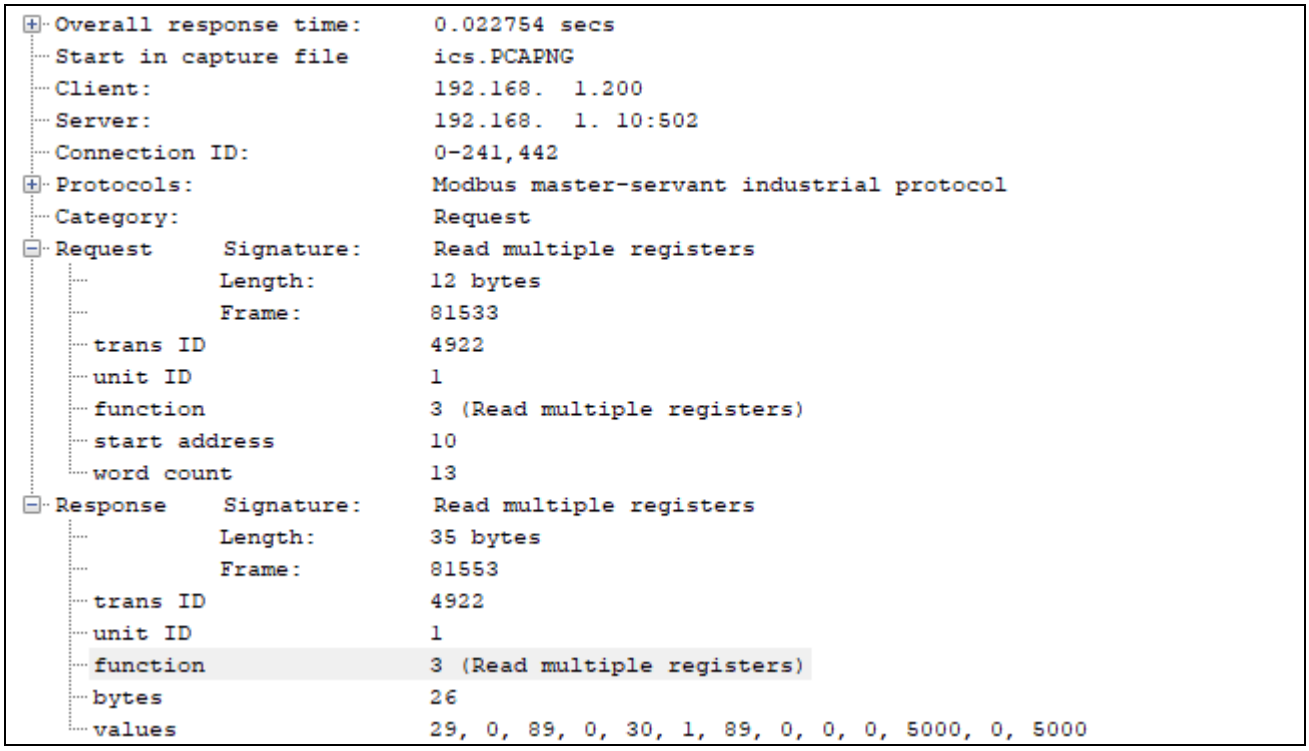
21.1 Pilz SafetyNET p Real-Time Frame Network (RTFN), IEC 61158-series Type 22 - Fieldbus

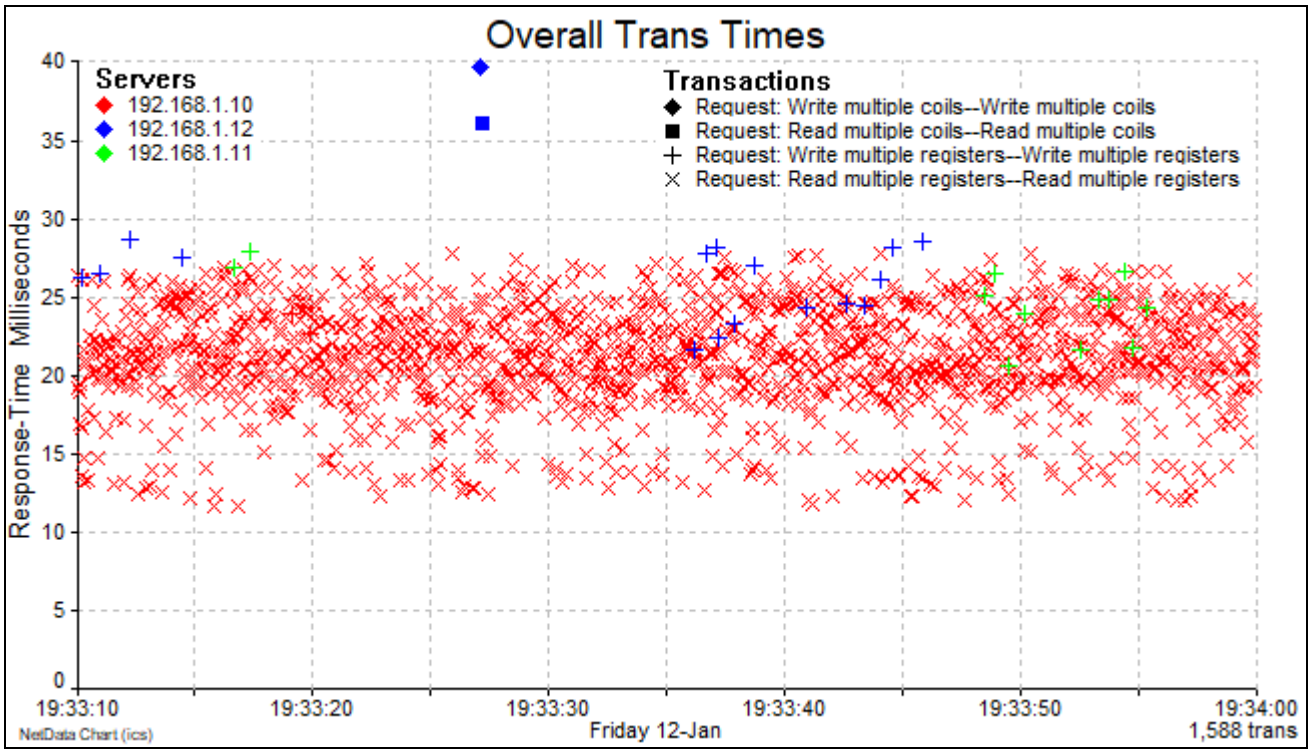
This Pilz decoder may be of interest to those with a network of automation controllers compatible with the Fieldbus standard, and supplements NetData's other automation decoder, for ODVA EtherNet/IP (Industrial Protocol). RTFN traffic uses only UDP. NetData's decoder parses headers, recognises the various frame types, and assigns unique secondary connection IDs to the data streams from individual control objects, in order that the packet timing chart can separate the packets from different objects into different bands, even though all packets use the same port number (40000).

21.2 Modbus-TCP

The Modbus protocol was developed in 1979 by Modicon for its programmable controllers and has become a widely-accepted standard for industrial control and monitoring devices. It uses a master-slave terminology which equates closely with the traditional client-server model. Only a master can initiate transactions that command actions in, or read data from, a variety of peripheral slave devices. Individual data bits are described as ‘coils’ because they once referred to the on/off state of coils in relay-logic control systems, and 16-bit words of data are referred to as ‘registers’. Modbus differentiates two types of registers: *input* registers intended to capture analog signals, and *holding* registers to hold temporary values.

The Modbus RTU (Remote Terminal Unit) protocol originally ran over serial lines (RS-232/RS-485) but is now also seen in Ethernet networks running over TCP. NetData fully parses all the common types of messages and matches responses with requests to measure response times even when multiple transactions execute concurrently.



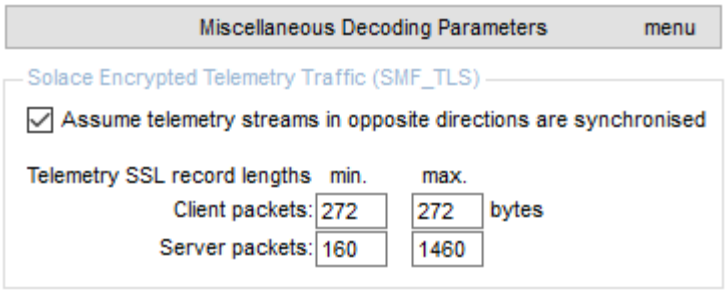

Overall Trans Times
Servers
192.168.1.10
192.168.1.12
192.168.1.11
Transactions
Request: Write multiple coils--Write multiple coils
Request: Read multiple coils--Read multiple coils
Request: Write multiple registers--Write multiple registers
Request: Read multiple registers--Read multiple registers
Response-Time Milliseconds
40
35
30
25
20
15
10
5
0
19:33:10
19:33:20
19:33:30
19:33:40
19:33:50
19:34:00
Friday 12-Jan
1,588 trans
NetData Chart (ics)

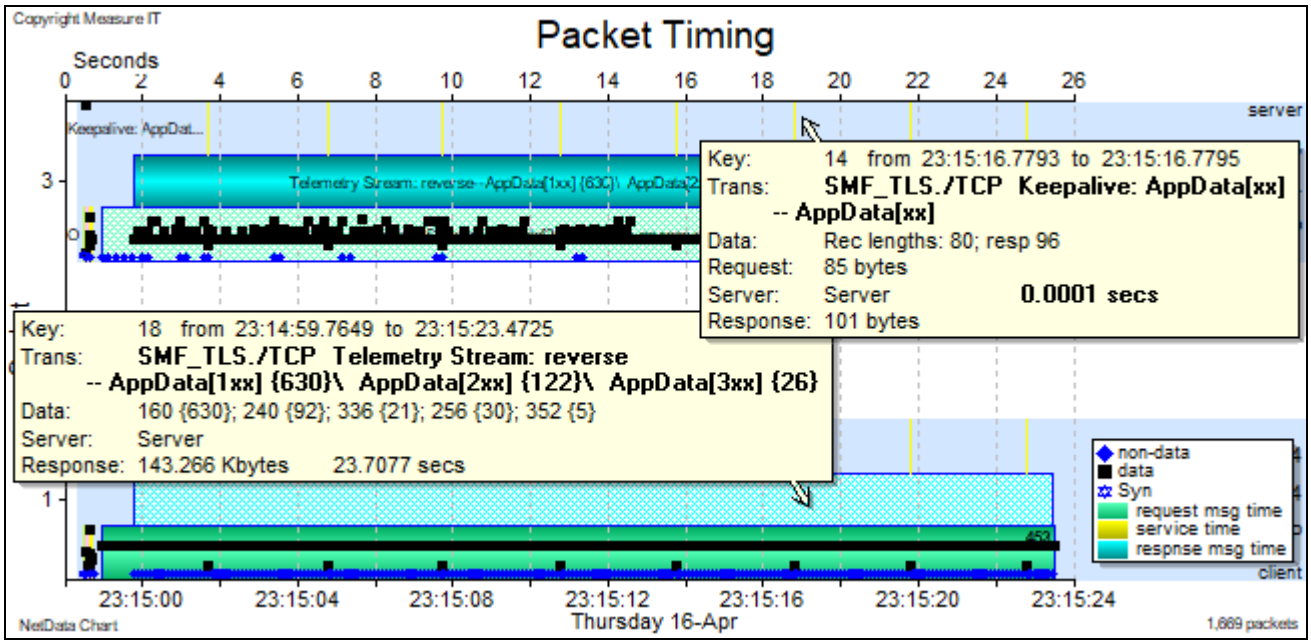
21.3 Encrypted Solace Telemetry Traffic

Solace software is ideally suited to telemetry systems but when the traffic is encrypted with TLS/SSL NetData's standard SSL decoder is unable to separate the telemetry data from concurrent transactions such as keepalive round-trips. Another decoder, tagged as 'SMF_TLS', can separate the different types of traffic according to the lengths of their SSL application-data records. Control parameters specify the minimum and maximum lengths of telemetry data, and NetData is able to characterise both the normal round-trips and the data streams.

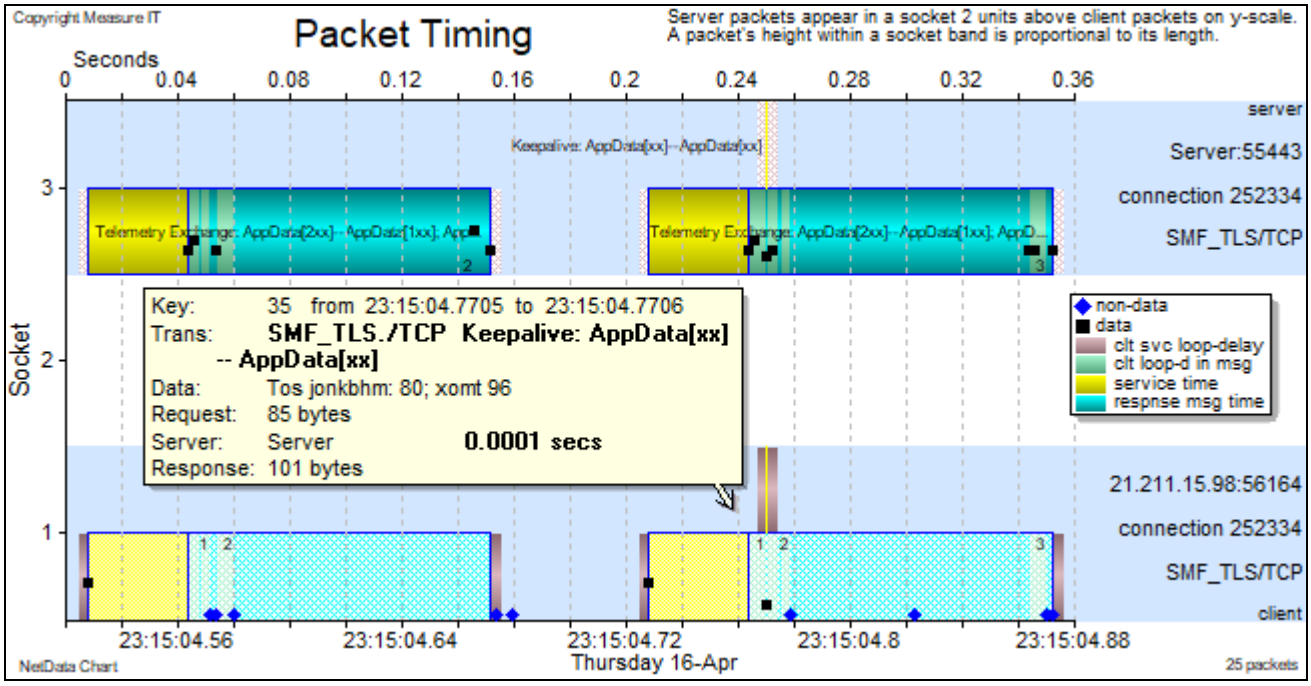
If a related control specifies that the client and server streams are synchronised, NetData will form pseudo transactions that characterise telemetry round-trips. There may be no significance in the request-response nature of these transactions, but, when plotted, the intervals measured as server-processing and client-preparation times reveal delays due to any form of network congestion or stress in the client or server.

The relevant controls are labelled *Solace Encrypted Telemetry Traffic (SMF-TLS)* and are found in the menu of Miscellaneous Decoding Parameters on the Decoding page of controls:



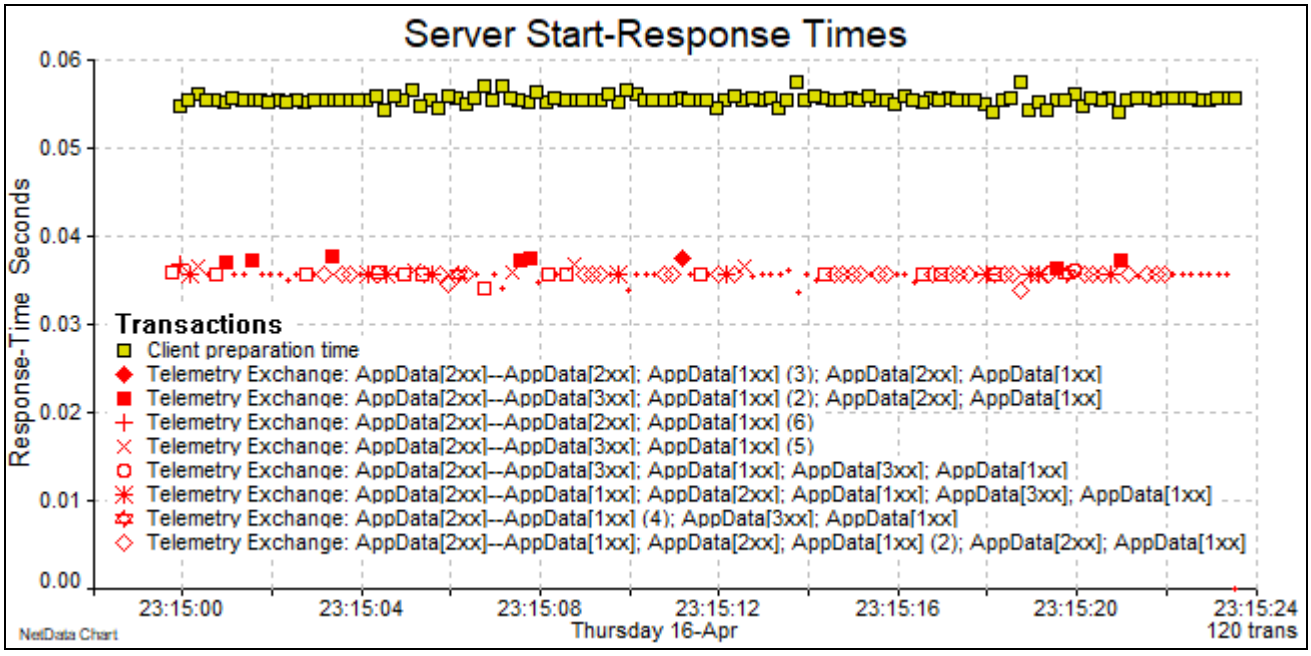


For this chart NetData didn't assume that the telemetry streams were synchronised. The two streams are characterised by pseudo transactions with long request and response messages respectively. The very short transaction bars of eight keepalive transactions are stacked above the two telemetry bars.



When NetData assumes that the client and server streams are synchronised as in this capture, it characterises a long sequence of individual round-trips, with keepalive transactions stacked above them.

A chart of client-preparation and server-response times reveals any small variations in the time intervals that might be caused by congestion or stress somewhere in the syatem.



21.4 Message Queuing Telemetry Transport (MQTT)

NetData decodes all types of messages and characterises transactions in Message Queuing Telemetry Transport (MQTT). This protocol is widely used in the networks of IoT (Internet of Things).

21.5 Internet Relay Chat

NetData decodes IRC messages which use TCP port 6667.

22 Packet Capture

22.1 Supported Wireshark Frame Formats

NetData handles the following Wireshark PCap and PCapNG frame formats:

Link Type (DLT)	NetData Frame Type	Description
0		Null – Raw IP or Ethernet, mixed
1	Eth	Ethernet
1	Prf	Ethernet prefixed with 28 bytes of ProfiShark metadata
6	TkR	Token Ring
12	---	Raw IP (starts with IP header)
99	SG2/SG3	may be Symantec Gateway Security appliance
101	---	Raw IP
104	CLC	Cisco HDLC (serial High-level Data Link Control)
105	WiF	IEEE 802.11 Wireless LAN (WiFi)
107	FrR	Frame Relay (Q.922)
113	LxC	Linux 'cooked-mode' capture of 'any' interface
127	WRt	IEEE 802.11 Wireless LAN with Radiotap header
147		DLT_USER0 - assumed Ethernet unless specified in PPI when it is assumed to be BTLE captured by an Ubertooth One
149		DLT_USER2 or PKTAP, raw IP probably from Apple iOS
157	BLE	DLT_USER10 - assumed BlueTooth Low Energy (BTLE)
162	Eth	DLT_USER15 - assumed Ethernet
177	LAP	Linux LAPD (Link Access Protocol for ISDN Data channel)
178	Jpr	Juniper Ethernet
192		Per-Packet Information (PPI)
274	Exp	IEEE 802.3br Ethernet Interspersing Express Traffic (IET) (Ethernet frames with their preamble of 8 bytes)

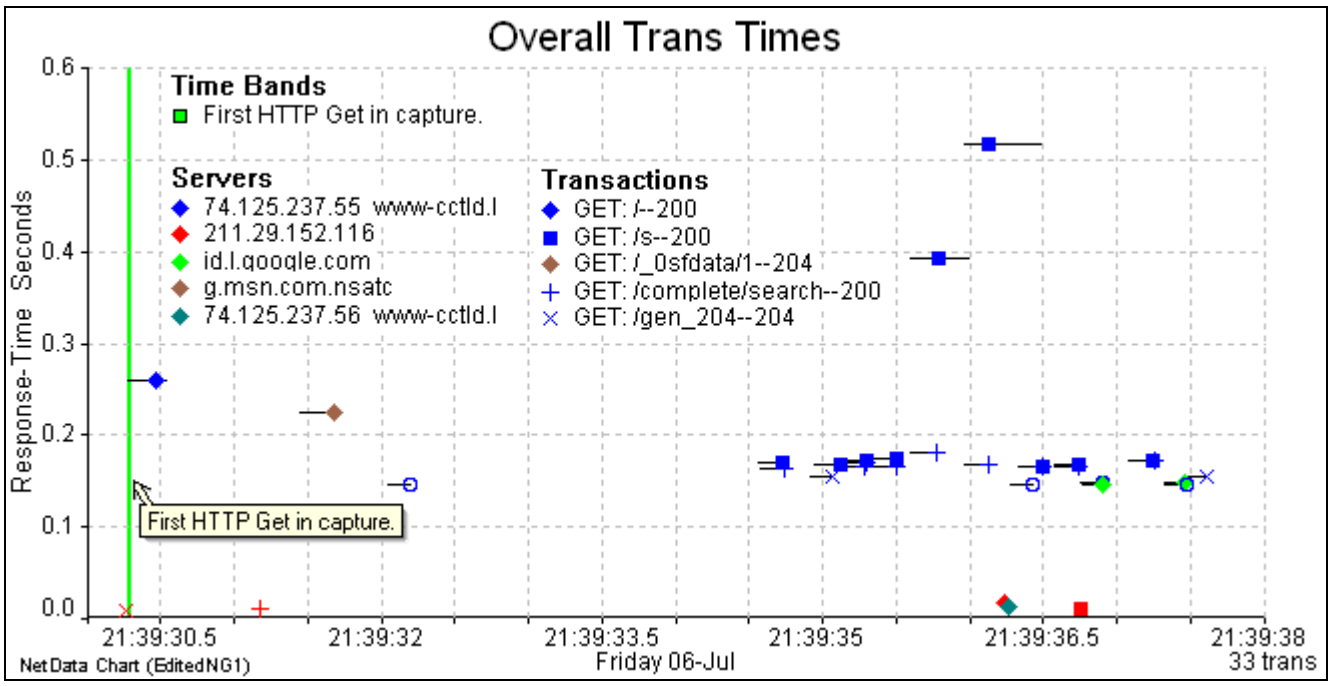
22.2 Wireshark PCap New Generation Capture Files

Version 1.8 of Wireshark was the first to be able to record traffic from multiple network interfaces, to mix packets with different frame formats ("encapsulations"), and to record the interface associated with each packet. It also introduces a new, extensible capture-file format with the name-extension pcapng (for PCap New Generation). The new format's versatility was demonstrated by adding user comments to individual packets and to the capture as a whole. It was also able to record the domain names of IP addresses, and record summaries of interface statistics at various points in the capture.

NetData reads New Generation capture files and records such non-packet blocks as section headers, interface descriptors, interface statistics and name-resolution blocks in its log file during analysis. It reads all three types of packet block covered by the NG specification, although only the Enhanced Packet block is likely to be seen. Comments attached to packets are written in the log file and on the trace window during analysis, and appear in the Comments column of the packet table. They are also recorded in the table of network events and like any event when loaded into the charting module can be indicated on the performance or timing chart with a coloured vertical line.

Start	End	Category	Description	Plot	Type	Frame
21:39:30.2844	21:39:30.2844	Comment	First HTTP Get in capture.	Yes		7
21:41:43.0177	21:41:43.0182	Pkt Order	server filled all gaps after 0.0005 secs	No	HTTP	593
21:41:43.0189	21:41:43.0189	TCP SAck	client acknowledged data in all gaps after 0.0000 secs	No	HTTP	595
21:43:14.9814	21:43:14.9815	Pkt Order	server filled all gaps after 0.0001 secs	No	HTTP	657
21:43:14.9823	21:43:14.9823	TCP SAck	client acknowledged data in all gaps after 0.0000 secs	No	HTTP	659
21:43:46.4274	21:43:51.4981	Fin-Wait	Reset of client ends server Fin-Wait state after 5.0707 secs	No	HTTP	969
21:44:16.28	21:44:16.28	Pkt Order	server filled all gaps after 0.0000 secs	No	HTTP	989
21:44:16.2808	21:44:16.2808	TCP SAck	client acknowledged data in all gaps after 0.0000 secs	No	HTTP	991
21:45:12.7495	21:45:12.7495	ICMP	Dest. Unreachable: Port unreachable 192.168.0.1:67 -> 192....	No		1450

In this example comments in the Events table have been selected for plotting with a green line:



The following panels show parts of a NetData log file that document a capture file's section header, interface descriptor, name-resolution block and first packet at the start of the file, and a statistics block at the end of the file.

```
Project controls:
Control files:
Detected a little-endian (Intel) PCap New Generation file
  Section header [164]
      byte-order magic     4D3C2B1Ah
      NG version           1.0
      section length       -1 (unknown)
      comment              Short Measure IT test capture
      operating system     Windows XP Service Pack 2, build 2600
      user application     Dumpcap 1.8.0 (SVN Rev 43431 from /trunk-1.8)
  Interface descriptor [132]
      link type            Ethernet
      snapshot length      65,535
      name                 \Device\NPF_{2006B410-A84D-45C2-90F7-E14FA6C62B62}
      timestamp resolution 6 (1,000,000 ticks/sec)
      operating system     Windows XP Service Pack 2, build 2600
  Resolved names [3248]   107
      Vrsn  Address                          Domain Name
      ------------------------------------------------------------------------------
      IPv4  74.125.237. 55                   www-cctld.l.google.com
      IPv4  74.125.237. 56                   www-cctld.l.google.com
      IPv4  74.125.237. 63                   www-cctld.l.google.com
      IPv4 216.239. 32. 10                   ns1.google.com
      IPv4 216.239. 34. 10                   ns2.google.com
```

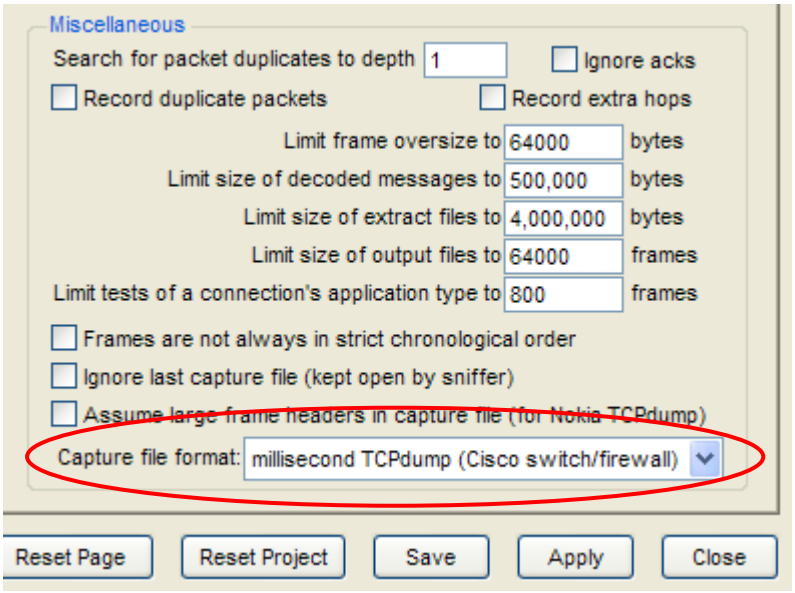
```
      IPv4 203. 62.195. 76                   ns5.mydyndns.org
      IPv6 2600:2005::76                     ns5.mydyndns.org
  Enhanced packet [92]
      interface ID         0
      timestamp            1341574759.967623 secs = 21:39:19.9676, 06/07/12  local time
      captured length      60
      packet length        60
      packet header        001E E5h jKutD 01E3 419C 0800 4500  0028 DC5C 4000 F606  DBE7 CF
Events:
```

```
      1795 21:48:49.4775 ans2.adsoftware 192.168.0.10       1172226 Final frame leaves ga
  Non-packet block at end of file:
  Interface stats [108]
      interface ID         0
      timestamp            1341575457.597750 secs = 21:50:57.5977, 06/07/12
      comment              Counters provided by dumpcap
      start time           1341574758.769625 secs = 21:39:18.7696, 06/07/12
      end time             1341575457.597750 secs = 21:50:57.5977, 06/07/12
      received packets     0
      I/F dropped packets  0
  Reached end of source file
Analysis time:             00:00:01.37 (+1.37 secs)
Broadcast IP addresses     1
Uncommon Max.Segment Sizes 2
Capture buffer runs from 3,544 to 1,245,268 bytes
Capture period overall:    00:11:35.46 (+695.4562)
```

22.3 Millisecond TCPdump Capture-File Format

A capture-file format has been encountered (2011) which carries the signature of little-endian TCPdump (as for Windows) but uses a quite different timestamp format that measures time in milliseconds. Even worse for performance analysis, the time-stamping clock has a resolution of only 10 milliseconds. These capture files were produced by a firewall card on a Cisco 6503 switch.

Wireshark reports the timestamps as invalid and its interpretations of time values tend to fall into closely packed clusters. The clusters occur at one-second intervals but the timestamps within a cluster are nearly identical. NetData normally interprets the timestamps in a similar fashion and the one-second intervals become readily apparent in the performance and timing charts. To read the timestamps correctly NetData has a new drop-down list in the bottom right-hand corner of the Tuning page of controls to specify a capture-file format explicitly. The new file format is described in the list as 'millisecond TCPdump'.



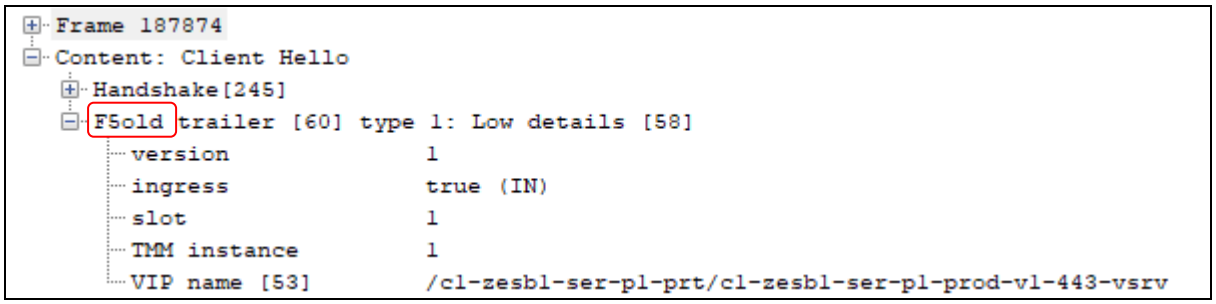
22.4 F5 Ethernet Trailers

F5's implementation of TCPdump on its Big-IP traffic-manager systems can be configured to record additional internal Traffic Management Microkernel (TMM) information in packet trailers. (See *K13637: Capturing internal TMM information with tcpdump*.).

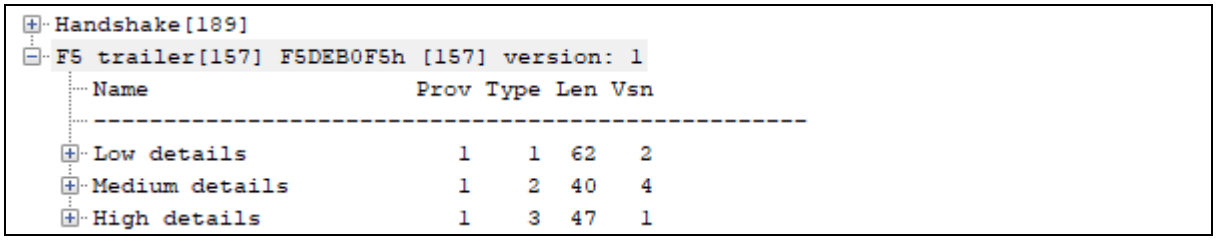
The first packet in the capture file is a pseudo packet that contains file information including the TCPdump command line:

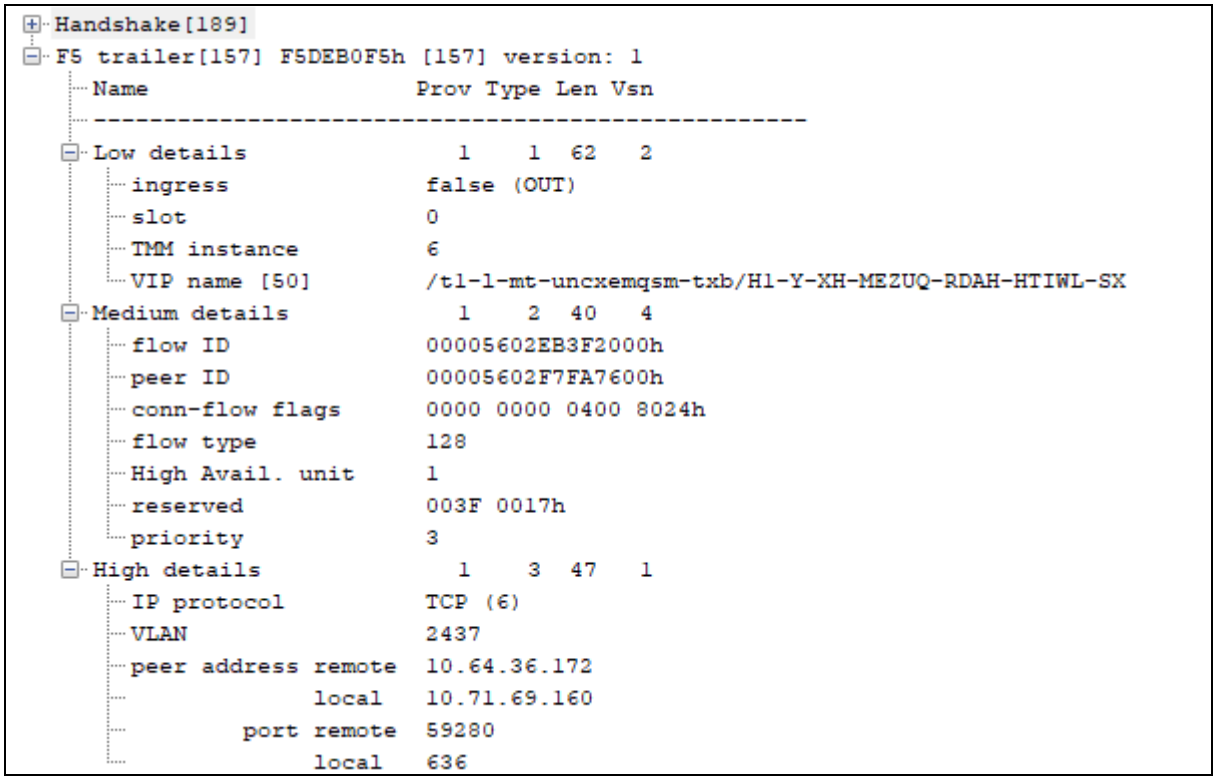
```
Frame 1
Content:
   F5-Pseudo-pkt
   CMD:                    tcpdump -s0 -nni 0.0:nnnp -w /var/tmp/CTASK0473851.pcap -vvv
                           host 10.46.135.28 and host 10.36.93.54
   VER:                    14.1.2.6 0.0.2
   HOST:                   gh2106-ltm02-CMD.mgmt.trav.business.com.au
   PLAT:                   Z100
   PROD:                   BIG-IP
   SESS:                   0
```

NetData displays a trailer at the end of a packet's *Content* field in a packet description, or in the window displayed by double clicking a field in the packet table's Content column. It distinguishes between a new format that first appeared in Big-IP 14.0, and the old format as in this example.



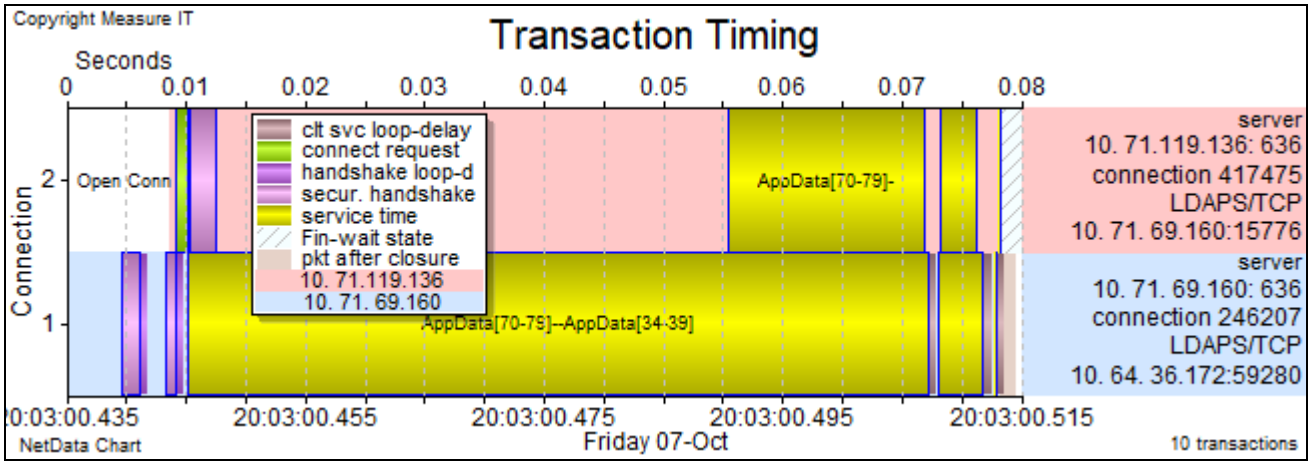
In the new format, each block of data is preceded by a provider ID, a type index, its length, and a version number. Provider 1 refers to the three optional *noise* levels of details, illustrated here by a full set of all three levels:





If the trailers contain high-level details in the new format, NetData records the connection ID of each connection's peer in the CoConn column of the connection table. This coupling of connection records

ensures that on a timing chart the coupled connections will be displayed on adjacent bands.to facilitate comparison of transaction and packet timing in the pairs of front- and back-end connections.



This chart shows that the F5 began opening a back-end connection immediately the front-end handshake had finished. There was a significant pause of 50 ms after completing the back-end handshake before relaying the user's request.

Normally the front-end connection is displayed under the back-end connection but this order can be reversed with the drop-down *Path* menu at the bottom of the chart's format-control window.

A third option in the Path menu causes the chart to ignore the coupling of connections.

22.5 CheckPoint Firewall-Monitor

The firewall monitor (*FW Monitor*) built into Check Point firewall and VPN software is designed to investigate firewall performance and determine where in the packet-processing chain an address is translated or a packet is dropped. Because it sees packets only above the link layer (layer 2) it has no knowledge of MAC addresses but it does record details of a packet's network interface and can capture a packet's contents at many different points in the firewall's inbound and outbound packet-processing chains. With default settings each packet is captured at two points in each direction (inbound and outbound), before and after processing by the firewall's virtual machine (the software module that does most of the packet processing).

FW Monitor writes capture files in the same format as Sun's Snoop. The only difference with Snoop is that the 12 bytes of source and destination MAC addresses in each packet are occupied by a description of the captured packet's direction, its capture point in the processing chain, and its network interface. NetData recognises FW-monitor capture files automatically and displays the descriptive information in the packet table's column normally occupied by the source MAC address.

The following table displays 24 captures of the one packet as it traversed the inbound and outbound processing chains, from eth-s1p1c0 to eth-s1p3c0. The IP identifier in the last column confirms that all the rows refer to the same packet. The hop count (TTL) was decremented between the inbound and outbound chains, and the client's private address was translated to a public address when the outbound indicator changed from lowercase to uppercase.

	Time Of Day	Seq	Source	MAC Address	Destination	Len	Transit	TOS/TTL	IP ID
■	07:27:20.6073	22655	clientPrivate: 4133	i 0 eth-s1p1c0	OnDemandSvr: 1445	200	0.000061	TTL 121	31421
■	07:27:20.6073	22656	clientPrivate: 4133	i 1 eth-s1p1c0	OnDemandSvr: 1445	200	0.000002	TTL 121	31421
■	07:27:20.6073	22657	clientPrivate: 4133	i 2 eth-s1p1c0	OnDemandSvr: 1445	200	0.000005	TTL 121	31421
■	07:27:20.6073	22658	clientPrivate: 4133	i 3 eth-s1p1c0	OnDemandSvr: 1445	200	0.000007	TTL 121	31421
■	07:27:20.6073	22659	clientPrivate: 4133	i 4 eth-s1p1c0	OnDemandSvr: 1445	200	0.000009	TTL 121	31421
■	07:27:20.6073	22660	clientPrivate: 4133	i 5 eth-s1p1c0	OnDemandSvr: 1445	200	0.000012	TTL 121	31421
■	07:27:20.6073	22661	clientPrivate: 4133	i 6 eth-s1p1c0	OnDemandSvr: 1445	200	0.000014	TTL 121	31421
■	07:27:20.6073	22662	clientPrivate: 4133	I 7 eth-s1p1c0	OnDemandSvr: 1445	200	0.000018	TTL 121	31421
■	07:27:20.6073	22663	clientPrivate: 4133	I 8 eth-s1p1c0	OnDemandSvr: 1445	200	0.000020	TTL 121	31421
■	07:27:20.6073	22664	clientPrivate: 4133	I 9 eth-s1p1c0	OnDemandSvr: 1445	200	0.000023	TTL 121	31421
■	07:27:20.6073	22665	clientPrivate: 4133	I a eth-s1p1c0	OnDemandSvr: 1445	200	0.000025	TTL 121	31421
■	07:27:20.6073	22666	clientPrivate: 4133	I b eth-s1p1c0	OnDemandSvr: 1445	200	0.000027	TTL 121	31421
■	07:27:20.6073	22667	clientPrivate: 4133	I c eth-s1p1c0	OnDemandSvr: 1445	200	0.000028	TTL 121	31421
■	07:27:20.6073	22668	clientPrivate: 4133	I d eth-s1p1c0	OnDemandSvr: 1445	200	0.000030	TTL 121	31421
■	07:27:20.6073	22669	clientPrivate: 4133	I e eth-s1p1c0	OnDemandSvr: 1445	200	0.000032	TTL 121	31421
□	07:27:20.6073	22670	clientPrivate: 4133	o 0 eth-s1p3c0	OnDemandSvr: 1445	200	0.000037	TTL 120	31421
□	07:27:20.6073	22671	clientPrivate: 4133	o 1 eth-s1p3c0	OnDemandSvr: 1445	200	0.000039	TTL 120	31421
□	07:27:20.6073	22672	clientPrivate: 4133	o 2 eth-s1p3c0	OnDemandSvr: 1445	200	0.000043	TTL 120	31421
□	07:27:20.6073	22673	clientPrivate: 4133	o 3 eth-s1p3c0	OnDemandSvr: 1445	200	0.000045	TTL 120	31421
□	07:27:20.6073	22674	clientPrivate: 4133	o 4 eth-s1p3c0	OnDemandSvr: 1445	200	0.000047	TTL 120	31421
□	07:27:20.6073	22675	clientPrivate: 4133	o 5 eth-s1p3c0	OnDemandSvr: 1445	200	0.000049	TTL 120	31421
■	07:27:20.6073	22676	clientPublic:25209	O 6 eth-s1p3c0	OnDemandSvr: 1445	200	0.000055	TTL 120	31421
■	07:27:20.6073	22677	clientPublic:25209	O 7 eth-s1p3c0	OnDemandSvr: 1445	200	0.000057	TTL 120	31421
■	07:27:20.6073	22678	clientPublic:25209	O 8 eth-s1p3c0	OnDemandSvr: 1445	200	0.000059	TTL 120	31421
■	07:27:20.6073	22679	clientPublic:25209	O 9 eth-s1p3c0	OnDemandSvr: 1445	200	0.000061	TTL 120	31421

The format of the capture description (in the MAC Address column) is

```
d h ttt-ffffff
```

where `d` indicates the direction:

- `i` pre-inbound processing
- `I` post-inbound processing
- `o` pre-outbound processing
- `O` post-outbound processing

`h` indicates the capture point in the inbound processing chain:

- `0` IP Options Strip (ipopt_strip)
- `1` **FWmonitor (i/f side)**
- `2` VPN decrypt (vpn)
- `3` Stateless verifications (asm)
- `4` VPN tagging inbound (tagging)
- `5` VPN decrypt verify (vpn_ver)
- `6` SecureXL conn sync (secxl_sync)
- `7` FW VM inbound (fw)
- `8` Wire VM inbound (wire_vm)
- `9` VPN policy inbound (vpn_pol)
- `a` SecureXL inbound (secxl)
- `b` **FWmonitor (IP side)**
- `c` FW SCV inbound (scv)
- `d` TCP streaming (in) (cpas)
- `e` IP Options Restore (ipopt_res)

or in the outbound processing chain:

- `0` IP Options Strip (ipopt_strip)
- `1` **FWmonitor (IP side)**
- `2` VPN nat outbound (vpn_nat)
- `3` TCP streaming (out) (cpas)
- `4` VPN tagging outbound (tagging)
- `5` Stateless verifications (asm)
- `6` FW VM outbound (fw)
- `7` Wire VM outbound (wire_vm)
- `8` VPN policy outbound (vpn_pol)
- `9` SecureXL outbound (secxl)
- `a` VPN encrypt (vpn)
- `b` **FWmonitor (i/f side)**
- `c` TCP streaming post VM (cpas)
- `d` IP Options Restore (ipopt_res)

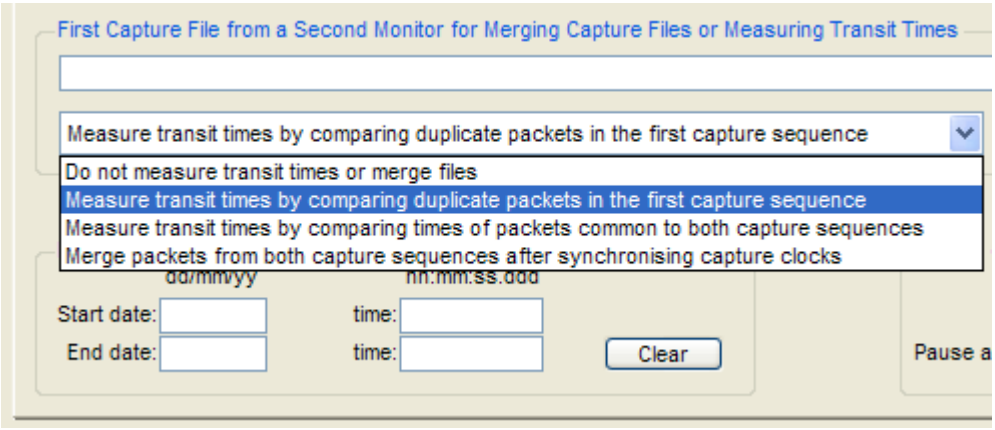
and

`ttt-ffffff` describes the network interface (e.g. '`eth-s1p3c0`')

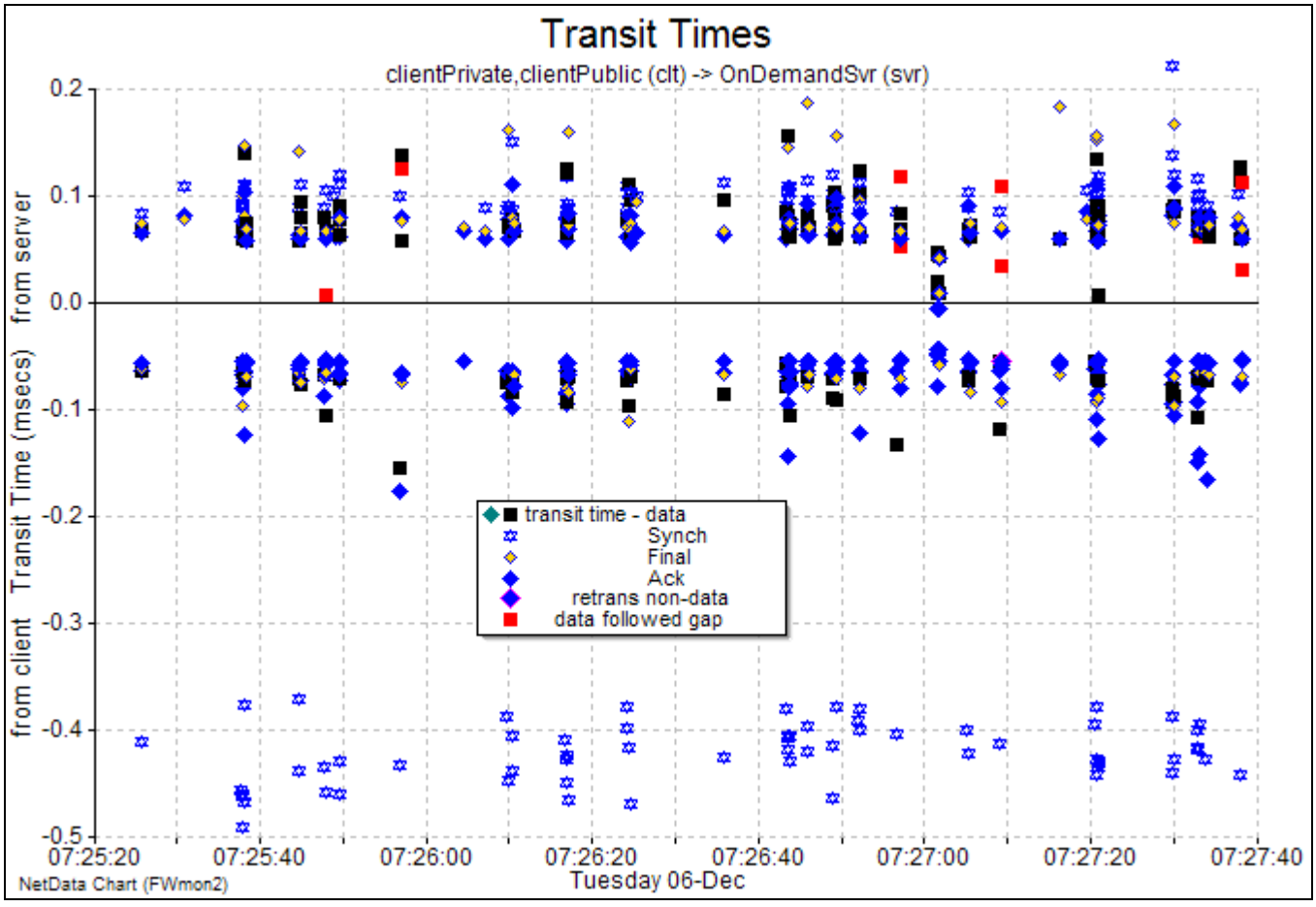
Capture points are defined by both a single character and an alias that is shown above in parenthesis. The alias can be used in the monitor's command line to specify required capture points.

The Transit column in the above packet table displays a transit time for each captured packet measured from the time of the first capture of the packet. The transit time associated with the first capture is the overall transit time from the first to the last capture points.

Transit times are measured only when 'Measure transit times by comparing duplicate packets in the first capture sequence' is selected from the drop-down list on the Input page of controls.



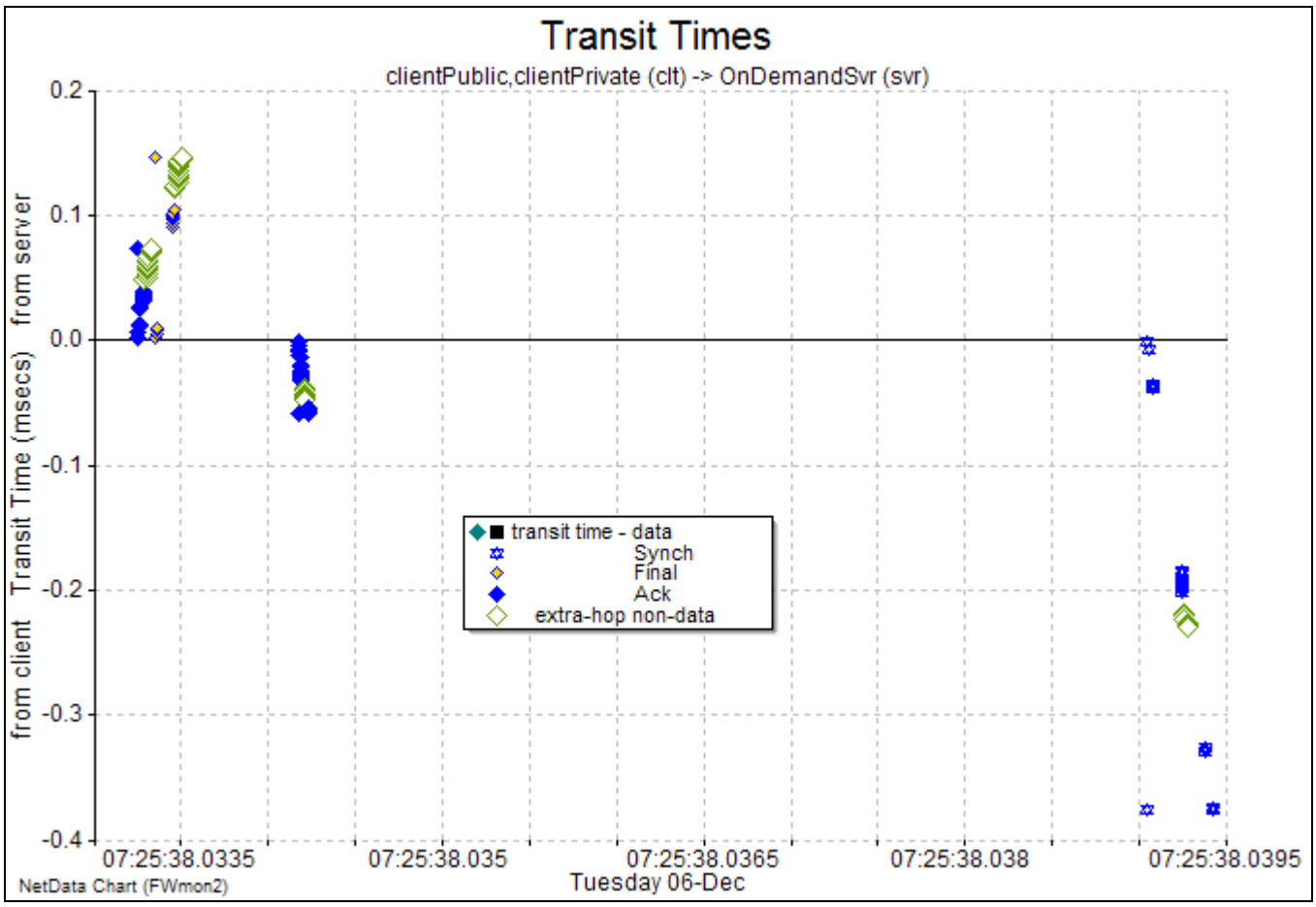
NetData regards all but the first capture of each packet as either a duplicate or an extra-hop copy and the extra copies are not normally recorded in the database. In the normal case every displayed transit time is an overall time for firewall processing. The chart below displays overall transit times for the client and server packets of a particular server as they traverse the firewall.



The negative transit times, plotted below the zero axis, relate to client packets and the markers above the axis relate to server packets. Each vertical tower of markers describes the packets of a cluster of transactions and includes packets needed to open and close their TCP connections. This chart indicates that most transit times are within 200 microseconds but the Synch packets from the client incur an extra delay of a third of a millisecond, probably to establish a record of the new connection in firewall memory. The red markers indicate TCP sequence gaps and the absence of green markers –

markers that would indicate gaps being filled – is evidence that the firewall monitor missed some packets entirely.

To see the transit times at every step along the processing chains within the firewall, as in the table above, it is necessary to check 'Record duplicate packets' on the Tuning page of controls. The chart below displays the progressive transit times of four packets, three packets at the end of one connection and a Synch packet to open a new connection.



The first appearance of an olive-coloured marker indicates a change in the hop counter (TTL) and this usually occurs between the inbound and outbound processing chains. A subsequent return to the packet marker's normal colour (blue for Synch and Ack packets; black for data packets) indicates that an address has been translated, and NetData regards the packet as belonging to a different connection. Markers for both the first and last copies of a packet are plotted at equal heights because those copies are recorded with the packet's overall transit time.

There are two large steps in the Synch packet's transit time that identify the modules contributing most to the overall transit time of 376 microseconds. The second step involves a marker change and therefore indicates the time needed for address translation (nearly 100 microseconds in this case).

22.6 Citrix NetScaler Nstrace Capture Files

NetData handles Citrix NetScaler capture files recorded by nstrace version 3.5 as well as old v2 files. The major change concerns packet metadata – the frame header – that contains a timestamp and specifies the packet's length, network card (NIC), VLAN and processing core. The header has increased in size from 24 to a minimum of 35 bytes, adding an error code, application indicator and four bytes of flags. Extra metadata is added to some packets to help in debugging TCP flow control by reporting the size of receive and congestion-avoidance (CWND) windows, sizes of receive and send buffers, and estimates of bandwidth and RTT. NetData displays a packet's metadata in the frame-description column of the packet table (as below).

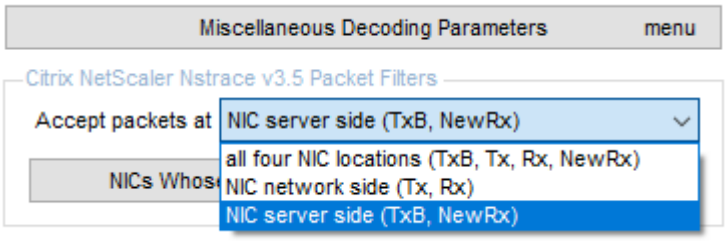
The first byte of a frame header indicates the location where the packet was captured, relative to the specified NIC. Version 3.5 adds a new location making a set of four acronyms for the different locations:

TxB	Buffered for transmission
Tx	Transmitted
Rx	Received before NIC pipelining
NewRx	Received after NIC pipelining

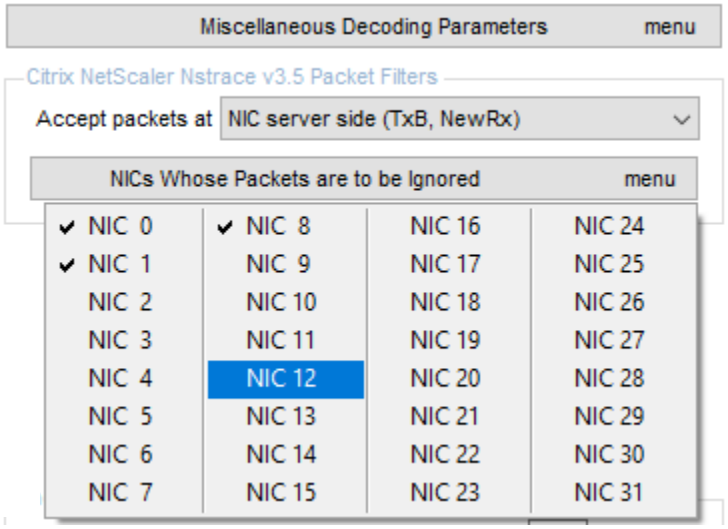
Time Of Day	Seq	Source	Desti...	Len	Frame Description	IP ID
15:26:11.290116	1569050	210.1...	10.39....	58	Rx NIC 3; PCB 0-0 len54; VLAN191; core0; hdr35; err0; appL2; fla...	5027
15:26:11.290116	1569051	210.1...	10.39....	58	Rx NIC 3; PCB 0-0 len54; VLAN191; core0; hdr35; err0; appL2; fla...	5030
15:26:11.290124	1569053	210.1...	10.39....	78	newRx NIC 8; PCB 10AE5AD7h-10AE5ADFh len74; VLAN191; co...	5025
15:26:11.290125	1569054	210.1...	10.39....	58	newRx NIC 8; PCB 10AE5AD7h-10AE5ADFh len54; VLAN191; co...	5028
15:26:11.290128	1569055	210.1...	10.39....	70	newRx NIC 8; PCB 10AE5AD7h-10AE5ADFh len66; VLAN191; co...	5026
15:26:11.290129	1569056	210.1...	10.39....	58	newRx NIC 8; PCB 10AE5AD7h-10AE5ADFh len54; VLAN191; co...	5027
15:26:11.290129	1569057	210.1...	10.39....	58	newRx NIC 8; PCB 10AE5AD7h-10AE5ADFh len54; VLAN191; co...	5030
15:26:11.290136	1569058	10.39...	210.1...	1388	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len1384; VLAN191; cor...	9262
15:26:11.290138	1569059	10.39...	210.1...	141	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len137; VLAN191; core...	9263
15:26:11.290138	1569060	10.39...	210.1...	1388	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len1384; VLAN191; cor...	9264
15:26:11.290139	1569061	10.39...	210.1...	141	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len137; VLAN191; core...	9265
15:26:11.290142	1569062	10.39...	210.1...	1388	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len1384; VLAN191; cor...	9266
15:26:11.290142	1569063	10.39...	210.1...	141	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len137; VLAN191; core...	9267
15:26:11.290143	1569064	10.39...	210.1...	1388	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len1384; VLAN191; cor...	9268
15:26:11.290143	1569065	10.39...	210.1...	141	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len137; VLAN191; core...	9269
15:26:11.290144	1569066	10.39...	210.1...	1167	TxB NIC 3; PCB 10AE5AD7h-10AE5ADFh len1163; VLAN191; cor...	9270
15:26:11.290183	1569073	10.39...	210.1...	1459	Tx NIC 3; PCB 0-0 len1384; VLAN191; core0; hdr106; err0; appTE...	9262
15:26:11.290183	1569074	10.39...	210.1...	212	Tx NIC 3; PCB 0-0 len137; VLAN191; core0; hdr106; err0; appTE...	9263

If nstrace is allowed to record as many packets as possible, it records every packet twice, as in the above portion of a packet table. A flow chart for an individual connection will reveal the time spent by packets in the transmission buffer or in the receiver's pipeline. However, NetScaler can handle a quite large throughput that may overload nstrace, causing it to miss more than half the packets. In such circumstances it is preferable to configure nstrace to record only one copy of each packet, say Tx and Rx to measure network performance, or TxB and NewRx to measure server performance.

If packets have been captured from all four locations, NetData can be configured to skip packets from some locations when analysing capture files. The necessary filter is found in the menu of Miscellaneous Decoding Parameters provided by a wide button on the Decoding page of controls:



Even after accepting packets from only one Tx and one Rx location some packets may be duplicated because they traverse two interfaces. These duplicates can be removed by filtering out the traffic of specified interfaces with the second drop-down menu:



22.7 Bluetooth Smart (Low Energy)

NetData decodes Bluetooth Smart (once known as Bluetooth Low Energy or BLE) traffic captured by a sniffer such as the Adafruit Bluefruit LE sniffer that uses a Nordic Semiconductor Bluetooth chip, or Ubertooth One that uses the CC2400 chip from Texas Instruments. For a sniffer, the RF chip is usually mounted with a processor on a small circuit board that can be plugged into a PC's USB port, and for its chip Nordic provides a Windows program that reads the captured packets and either displays them directly in Wireshark or writes them to a capture file. Sniffers like this can be bought online for less than US$30.

Captured Bluetooth packets are recorded with various types of metadata such as RSSI (Received Signal Strength Indicator), radio channel number or frequency, CRC validity, source indicator (master or slave) and timing information. Wireshark sees the metadata as a special header prefixed to the transmitted packet, which is why it misrepresents packet length by assuming that all header bytes were "on wire". NetData always includes a packet's CRC length in its count of the overall length, and in the BLE case counts the 3-byte CRC but subtracts the length of the metadata header.

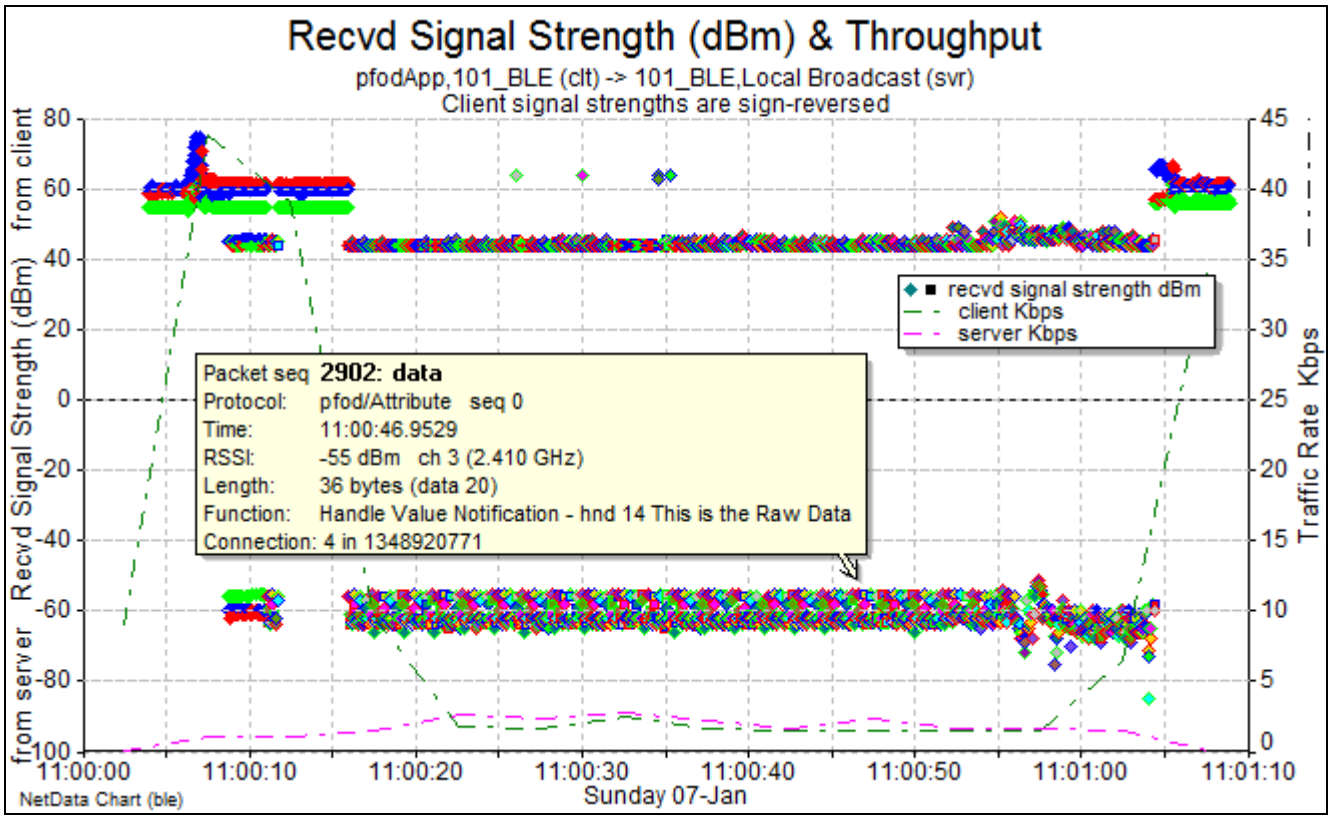
Different sniffers format the metadata in various ways. **Nordic sniffers** specify a user-defined data-link type such as User10 (DLT=157) and provide a metadata header of 17 bytes, but not all versions of Wireshark handle this format equally well. Various documents recommend version 1.10.14, version 1.12.1 or any subsequent version before version 2. Nevertheless, version 2.0.3, for example, does decode BLE packets provided Wireshark's 'preference' for protocol DLT_USER is edited, to specify a payload protocol of *btle* and a header size of 17 for the relevant data-link type. Wireshark version 2, however, does not accept the Nordic plug-in and can't dissect the metadata in the 17-byte header.

The capture files of **Ubertooth One sniffers** take a standardised approach by specifying a PPI (Per-Packet Information) data-link type (DLT=192), but this versatile formatting scheme is now deprecated. A PPI header contains a version number, an alignment flag, a length indicator, a data-link type for the packet contents, and a list of metadata fields in TLV (Type, Length, Value) format. Ubertooth One specifies a packet type of User0 (DLT=147) and provides a single, composite PPI field with a user-specified type of 30,006 and length of 7. Standard Wireshark parses the PPI header and dissects the BLE packet contents but can't dissect the metadata unless it has a Ubertooth plug-in. NetData dissects the composite field, extracting a channel frequency, a zero byte that may be reserved for flags or RSSI, and a 4-byte timestamp in tenths of a microsecond. Because the Ubertooth metadata doesn't yield a packet source (master or slave) NetData deduces the source from channel-hopping behaviour during the connection state: the first packet of a new connection event, which must change the channel number, is assumed to be sent by the master; acknowledgements or responses from the slave always use the same channel as the master.

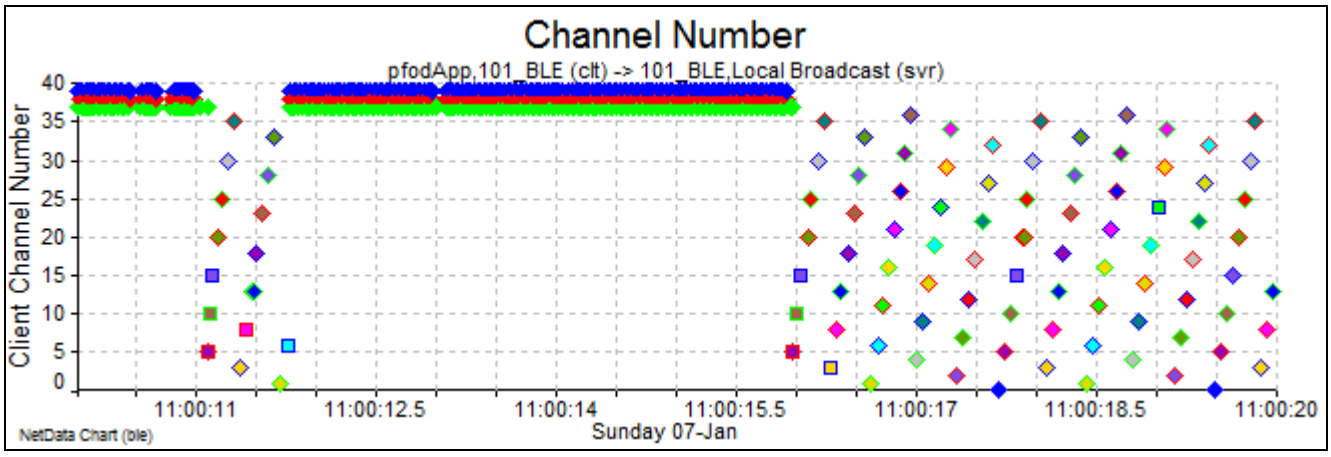
Ubertooth One is able to monitor only one of the three advertising channels.

NetData presents packet metadata in the Frame Description column of the packet table. Descriptions of other Bluetooth headers are found in the columns for Network-Layer and Transport-Layer Options, leaving free the Content column for descriptions of application messages provided by NetData's higher-layer application decoders.

As it can for WiFi traffic, NetData's flow chart can plot the signal strength of individual packets, as in the chart below. The server (slave) packet pop-up indicates signal strength (-55 dBm), radio channel number (3), frequency (2.410 GHz), connection channel ID (4), channel protocol (Attribute), application protocol (pfod), packet function (Handle Value Notification), attribute handle (14), and packet payload ("This is the Raw Data…").



The above chart of signal strengths uses the 'Colour by ID' option that assigns colours to markers according to their radio channel number. Advertising packets cycle through channels 37, 38 and 39 (at the middle and ends of the radio spectrum), and other packets cycle through channels 0 to 36 in a pseudo random fashion that mitigates the effect of signal fading. The colours reveal the cyclic pattern of channels and any consistent weaknesses in particular channels.



Checking the box to request 'Channels' plots Bluetooth packet markers against a scale of channel numbers (as above). In this case it reveals that the first packet of a connection used channel 5 and successive pairs of master and slave packets incremented the channel number by 5, modulo 37. The resulting pattern is not particularly random but steps quickly across the available frequency spectrum and eventually uses every channel. The connection master specifies the set of channels to be used, and the channel increment – a random number between 5 and 16 – in its connection-request packet.

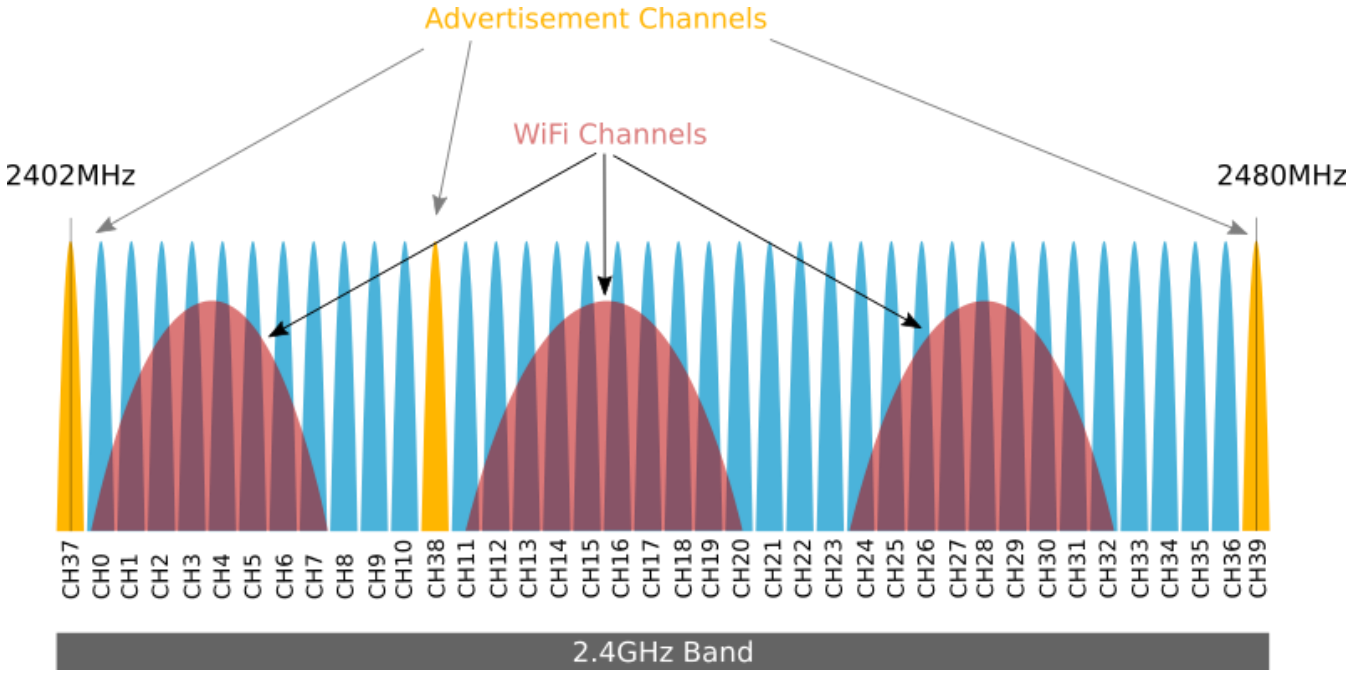
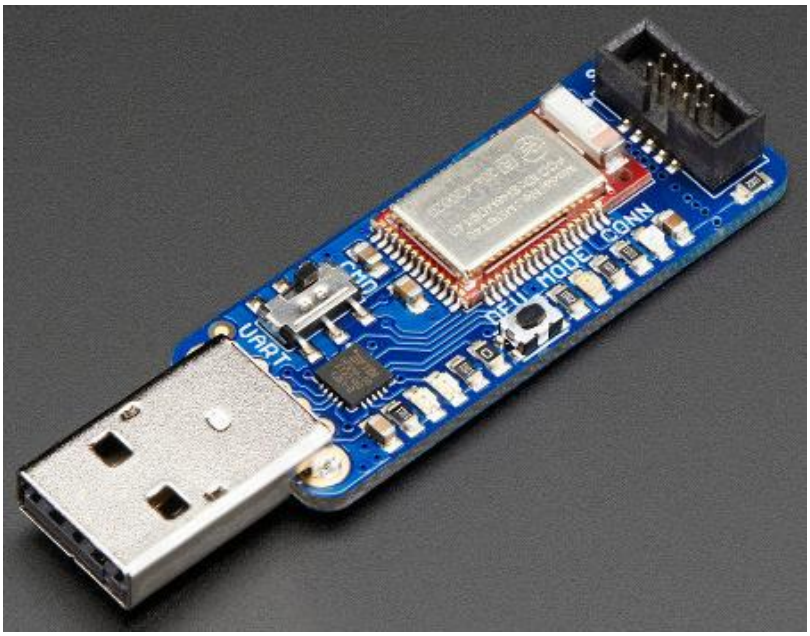
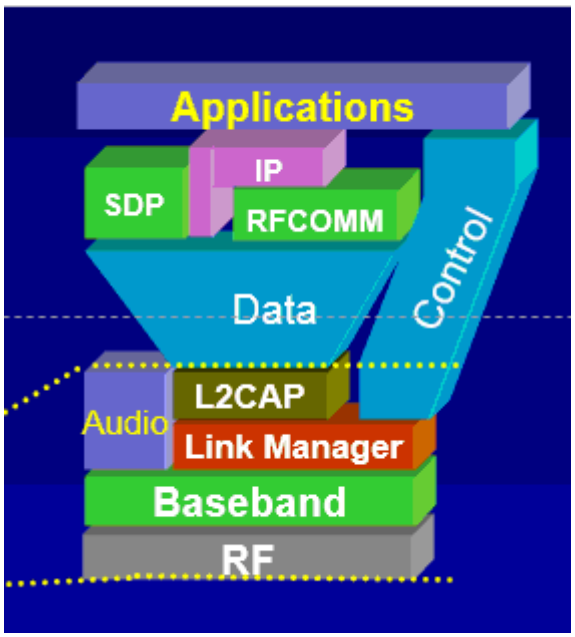


Diagram by Argenox

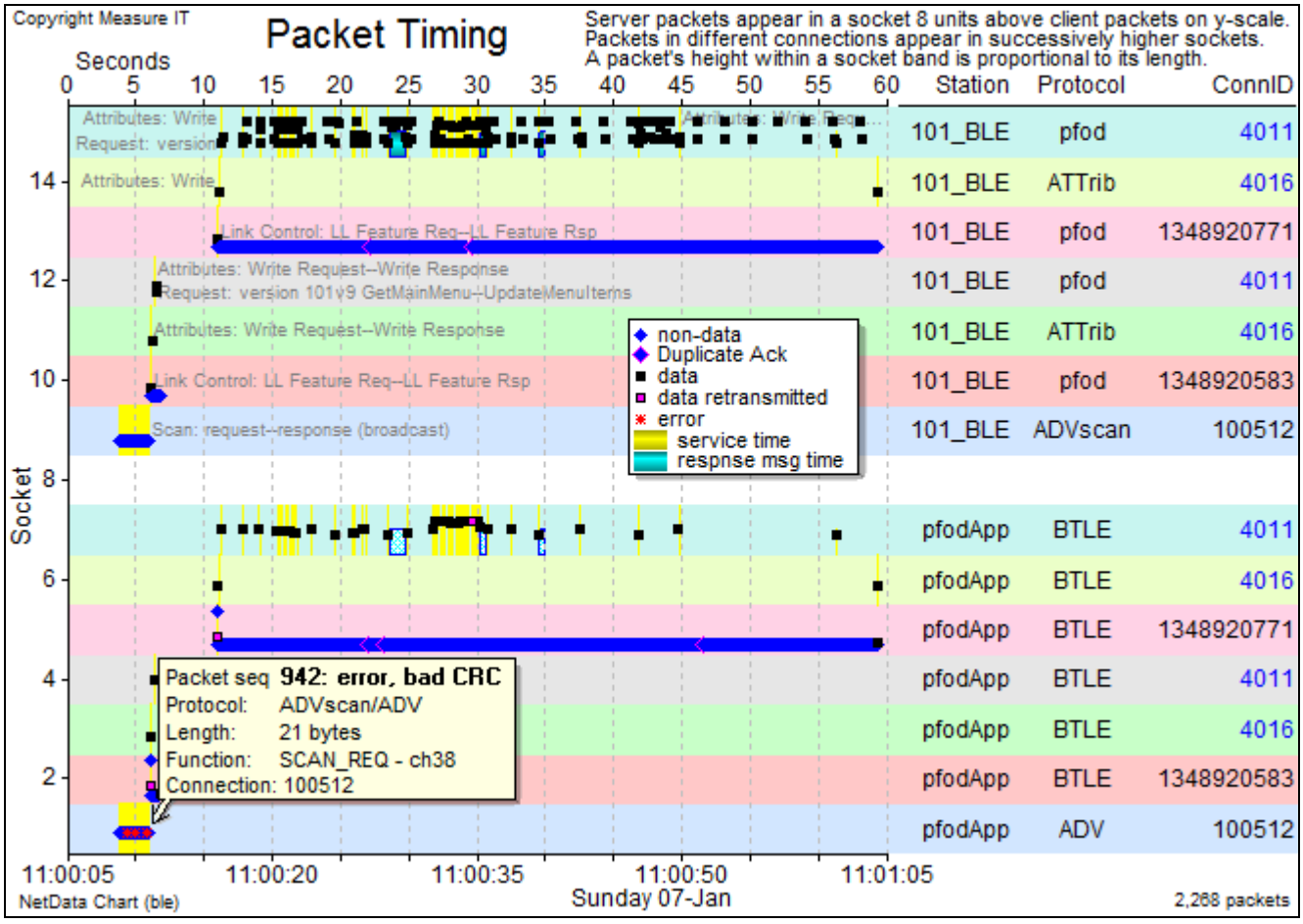




In operation the sniffer records all the advertising and other packets of a nominated device, including all the packets involved in a connection with another device. NetData decodes the broadcast advertising packets, assigns unique connection IDs to scanning exchanges with other devices, and when a connection is established decodes the link-layer, link-control, data-PDU and L2CAP (Logical Link Control and Adaptation Protocol) headers. A single connection between a pair of devices can be divided into any number (up to 255) of bi-directional channels whose packets are distinguished by a channel identifier (CID) in the header. NetData regards the CID as a secondary connection ID, to separate different channel flows on the timing chart.

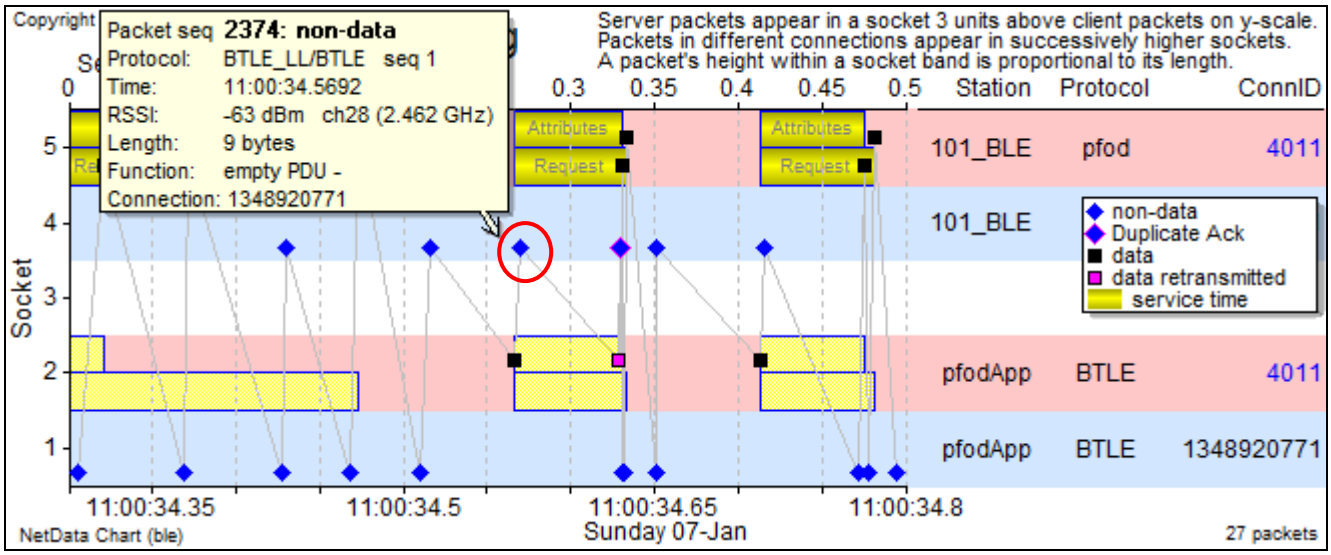
The first 63 CIDs are reserved for specific L2CAP functions, and NetData decodes messages of the Attribute protocol that uses a CID of 4 to search, read and write device attributes. Different attributes are identified by their handles and NetData assigns attribute Read and Write transactions to different secondary connections according to their handles. Some applications convey all their data as values of attributes.

Although NetData handles the common protocols it doesn't yet decode all the protocols defined in the 24-Mbyte PDF *Bluetooth Core Specification Version 4.2*, December 2014, by the Bluetooth SIG.

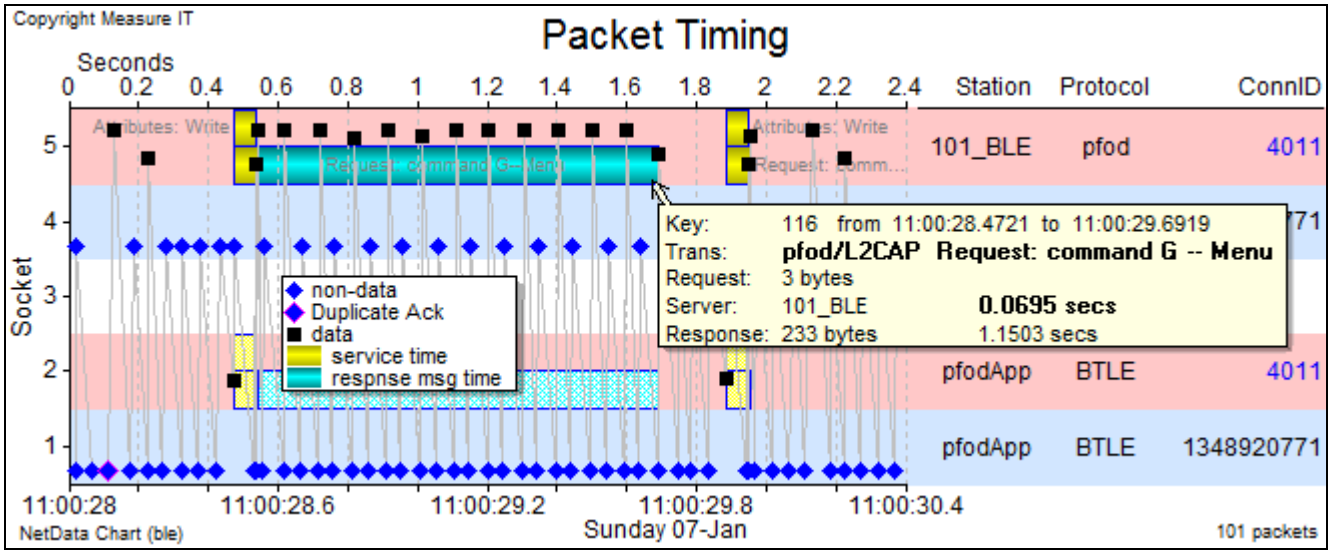


In this sample timing chart ADV_IND advertising packets have been ommitted to save space. The bottom connection carries scanning exchanges – unicast SCAN_REQ packets from the client and responding broadcast SCAN_RSP packets from the server – between an Android phone (the *Initiator*) running a pfod device-conntrol application and a device named '101_BLE' (the A*dvertiser*). A short connection was followed by a long connection, and both connections have been split into three pairs of connection bands, the bottom pair carrying the empty LL packets (polls and acknowledgements), the top pair conveying pfod transactions in the Attribute channel (CID=4) involving attributes with handle 11, and the other handling attributes with handle 16. The connection IDs of the two secondary connections are simple functions of the CID and the handles of their respective attributes (making 4011 and 4016)

The sniffer reported that three SCAN_REQ packets had a bad CRC and they are highlighted with red stars (in the bottom-left corner of the chart). The headers of link-layer packets have single-bit data-sequence and ack numbers and like the receiving device NetData checks these to recognise sequence gaps or retransmissions. On the packet-timing chart it highlights them in the usual way with red and purple markers. The sequence numbers (0 or 1) appear in the same columns of the packet table as TCP sequence numbers.



In this chart the BLE slave device acknowledged a data packet from the master by issuing an empty PDU (circled in red) but the ack was probably not received by the master because it retransitted its data packet, and that retransmission prompted a retransmission of the empty PDU (described by NetData as a duplicate ack). The apparently lost packet was received by the sniffer on channel 28 with a signal strength of -63 dBm, an uncommonly low strength according to the chart of all the packet signal strengths. However, the packet loss in this case had no effect on performance.



This chart shows the response time for a request from a smart phone app to download a menu of commands from a Bluetooth device. Although the radio signalling rate can handle 1 Mbps, this transaction took 1.1 seconds to dowload a message of only 233 bytes because each data packet was restricted to a payload of 20 bytes, the device took about 50 ms to react to each received packet, and it alternated data packets with acknowledgements in separate, empty packets.

In the secondary connection at the top of these charts, transactions bars are stacked in pairs. One bar measures the response time from Write Request to Write Response, a round-trip that conveys only an application request. The application response is conveyed in an unsolicited Notification message from server to client, and the second bar measures the application response time from Write Request to the Notification.

22.8 Juniper Ethernet Encapsulated Packets

NetData decodes capture files with packets encapsulated as *Juniper Ethernet* which is identified by Wireshark link-type 178.

The Juniper encapsulation prepends a header with packet metadata including interface information which NetData records in the frame-description column of the packet table:

```
MGC In, no L2, bit3, extd hdr; (3)DevMedia=Ethernet;
   (6)LgcEncaps=Ethernet; (1)DevI/F=129; (4)LgcI/F=325; (5)LgcUnit=201
```

The header always begins with a magic code ('`MGC`') and a direction indication (In/Out). A flag in the fourth byte indicates whether the encapsulated packet has been stripped of its Level-2 header, in which case the packet begins with a specified Level-3 protocol – commonly IP4.

The distinguishing characteristics of a Juniper trace can be found in the packet table by revealing columns that contain MAC addresses, the frame type ('Jpr') and the frame description. The absence of Level-2 information ('no L2') is confirmed by the absence of MAC addresses.

	Time Of Day	Seq	Source	MAC/Tx Addr...	MAC/Rx Addre...	Destina...	Frm	Frame Description	AppT...	Data	Blks	Function
■	00:42:27.439891	62547	174.68...			174.68....	Jpr	MGC In, no L2, bit3,...	BGP4-4	102	1	Route Upd
◆	00:42:27.540527	62548	174.68...	Juniper-89C7C1	DA18D3-BF1D5F	174.68....	Jpr	MGC Out, extd hdr; ...	BGP4-4			
■	00:42:33.504251	62549	174.68...			174.68....	Jpr	MGC In, no L2, bit3,...	BGP4-4	216	2	Route Upd
◆	00:42:33.606569	62550	174.68...	Juniper-89C7C1	DA18D3-BF1D5F	174.68....	Jpr	MGC Out, extd hdr; ...	BGP4-4			
■	00:42:33.875161	62551	174.68...	Juniper-89C7C1	DA18D3-BF1D5F	174.68....	Jpr	MGC Out, extd hdr; ...	BGP4-4	480	6	Route Upd
■	00:42:33.875194	62552	174.68...	Juniper-89C7C1	DA18D3-BF1D5F	174.68....	Jpr	MGC Out, extd hdr; ...	BGP4-4	480	6	Route Upd
◆	00:42:33.877081	62553	174.68...			174.68....	Jpr	MGC In, no L2, bit3,...	BGP4-4			
◆	00:42:33.877096	62554	174.68...			174.68....	Jpr	MGC In, no L2, bit3,...	BGP4-4			
■	00:42:33.878594	62555	174.68...	Juniper-89C7C1	DA18D3-BF1D5F	174.68....	Jpr	MGC Out, extd hdr; ...	BGP4-4	480	6	Route Upd
■	00:42:33.878623	62556	174.68...	Juniper-89C7C1	DA18D3-BF1D5F	174.68....	Jpr	MGC Out, extd hdr; ...	BGP4-4	480	5	Route Upd
■	00:42:33.878644	62557	174.68...	Juniper-89C7C1	DA18D3-BF1D5F	174.68....	Jpr	MGC Out, extd hdr; ...	BGP4-4	373	4	Route Upd
◆	00:42:33.881533	62558	174.68...			174.68....	Jpr	MGC In, no L2, bit3,...	BGP4-4			
◆	00:42:33.980356	62559	174.68...			174.68....	Jpr	MGC In, no L2, bit3,...	BGP4-4			
■	00:42:34.102628	62560	174.68...			174.68....	Jpr	MGC In, no L2, bit3,...	BGP4-4	121	1	Route Upd

22.1 National Instruments Observer

NetData reads capture files written in the native format of National Instruments Observer (NIO), version 9. This format has large frame headers of 56 bytes which comprise four 2-byte length indicators, an 8-byte timestamp that records the number of nanoseconds since the beginning of the 21st century, some quite static parameters, some variable parameters, and the sniffer's own frame sequence number. NetData records the sequence number and variable parameters in the frame-description column of the packet table, and notes any changes of the static parameters in the analysis log file. NetData also logs any gaps in the NIO sequence numbers.

At one-second intervals Observer writes in the capture file a frame that doesn't refer to any particular packet. NetData records the contents of these frames in the log file.

22.1 Ethernet Interspersing Express Traffic (IET)

NetData handles frames recorded with the format of Wireshark's link-type 274 which prepends the Ethernet 8-byte preamble to the normal Ethernet frame. The preamble begins with alternating one and zero bits (55h) to synchronise the receiving clock, and normally ends with an eighth byte of D5h called the Start Frame Delimiter (SFD). To accommodate express frames that can be interspersed with preemptable normal frames, the last two bytes of the preamble can convey information about the type of frame, as specified by IEEE 823.3br. *Interspersing Express Traffic (IET).*

Capturing frames with their preambles is an option of ProfiShark taps.

In the IET protocol, frames are known as mPackets, and the starting delimiter as the SMD (Start mPacket Delimiter). Express frames use the traditional delimiter SFD labelled as SMD-E. A preemptable frame starts with SMD-S, and, if it is pre-empted, its continuation frames are delimited with an SMD-C in the seventh byte and a fragment count in the eighth byte. To protect against bit errors and frame loss – because the preamble is not included in the frame's CRC – the byte codes of both an SMD-S and an SMD-C cycle through four different values with a Hamming distance of 4. The codes for fragment counts use the same values as an SMD-S.

Two unique SMDs serve to negotiate the use of IET by the device at each end of a link sending a *Verify* frame (07h) and expecting a *Respond* frame (19h). These frames carry only null bytes to pad them out to the minimum frame size.

NetData assigns IET frames a frame type of 'Exp' and summarises each preamble in the Frame Description column of the packet table. Special frames (SMD-S, Verify and Respond) are also recorded as network events.

22.2 ProfiShark Metadata with High-Resolution Timestamps

ProfiTap's ProfiShark appliance is a network tap with two Ethernet (RJ45) ports (labelled A and B) and a USB 3.0 port to output captured packets for recording traffic reliably from both ports at full one-Gbps speeds and with timestamping at a resolution of 8 nanoseconds. It is supplied with a USB utility for controlling the tap and recording output packets directly to disk in the PCapNG format.

Another form of output through the USB port allows traffic to be monitored in near real time with a sniffer such as Wireshark. In this case the PC installs a USB driver that simulates an Ethernet port, and the sniffer applies its own timestamps to packets as they are read from the port. Unfortunately, the ProfiShark doesn't push packets through the port in real time, but, for efficiency, records packets in a half-megabyte buffer that is emptied only after a few seconds or when the buffer becomes full. With a traffic rate of 20 Mbps, for example, some packets can be timestamped late by the sniffer by up to 250 ms, and groups of up to 300 packets receive almost identical timestamps.

To overcome this late timestamping by the sniffer, ProfiShark can be configured to prefix every packet with 28 bytes of metadata that includes ProfiShark's own timestamp, but the sniffer must be modified to read these timestamps. ProfiTap provides a DLL for Wireshark to read the metadata.

NetData automatically detects the presence of the metadata and adopts the ProfiShark timestamps in place of the sniffer timestamps. The Frame Description column of the packet table identifies the packet's ProfiShark port (A or B), records the difference between each packet's two timestamps, and displays some packet-length information. For example, the column entry

```
PrfShk 6, dL6, ptB, dt0.263479s
```

indicates that the packet was captured from Port B and the sniffer timestamp was delayed by 263 ms. NetData assigns these ProfiShark frames a frame type of 'Prf'.

22.3 NetMon 3.3 Capture Files

NetMon 3 uses the same format as NetMon 2 for its capture files, but it often inserts two pseudo packets at the beginning to record the path names of NetMon itself and the capture file it created. NetData recognises such special frames and logs their contents, but doesn't analyse them any further.

NetMon 3 also tends to corrupt the timestamps of its first set of recorded frames, producing stamps whose high-order bytes contain hex 99. These timestamps increment like a normal clock, and eventually the timestamp jumps to a normal series beginning with a clock value of zero. Wireshark usually interprets the corrupted timestamps as negative or refuses to display them in any form. NetData corrects the corrupted timestamps to start a series with a value of zero. When the series jumps to normal values, NetData adjusts its correction mechanism to continue the timestamp series without a jump. The true time delta from the end of the corrupted series to the start of the next series is unknown, and NetData assumes a delta of zero. The resulting series of timestamps is monotonic and starts at the date and time-of-day indicated in the capture-file header. It appears that the resulting times are higher than the true time by the timespan of the corrupted set of frames, usually just a few seconds.

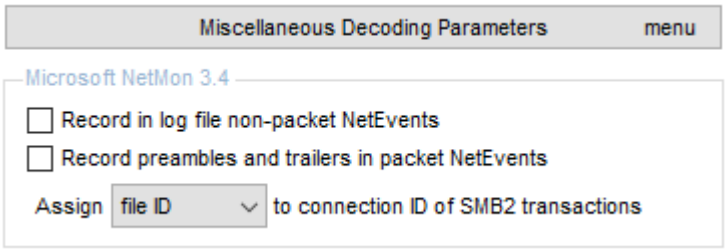
22.4 NetMon 3.4

The latest version of Microsoft Network Monitor, NetMon 3.4, is able to read Event Trace Log (ETL) files produced by the ETW (Event Tracing for Windows) system when under the control of the program *NetSh*. NetSh provides a powerful means of collecting and combining ETW traces from many Windows components (called event *providers*) in the protocol stack, such as WinSock, TCP/IP, Wireless LAN Services and NDIS. If the NetSh command line includes the parameter ‘capture=yes’ the resulting ETL file will contain raw network packets from NDIS, each with NDIS and NetEvent headers that contain extra information such as an interface index and the ID numbers of the issuing process and thread.

If NetMon reads an ETL file and saves it in its capture-file format 2.3, the resulting .cap file can be read by NetData. In addition to the 12-byte NDIS header and an 89-byte NetEvent header, each frame has the standard 16-byte NetMon frame header – with a timestamp, full packet length and the captured length – and a trailer of 15 bytes that conveys the packet’s media type, an index to a Process Information table, a high-resolution timestamp and a time-zone index.

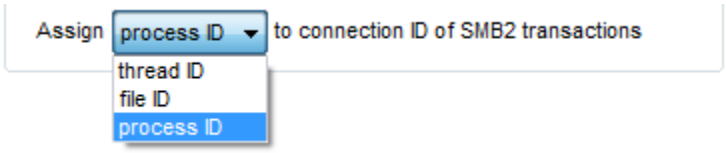
When analysing .cap files derived from ETL files NetData leaves non-packet events out of its database but does parse them partially and records them in its analysis log file. At the end of the log it creates a tree of Windows processes that displays their parent-child relationships.

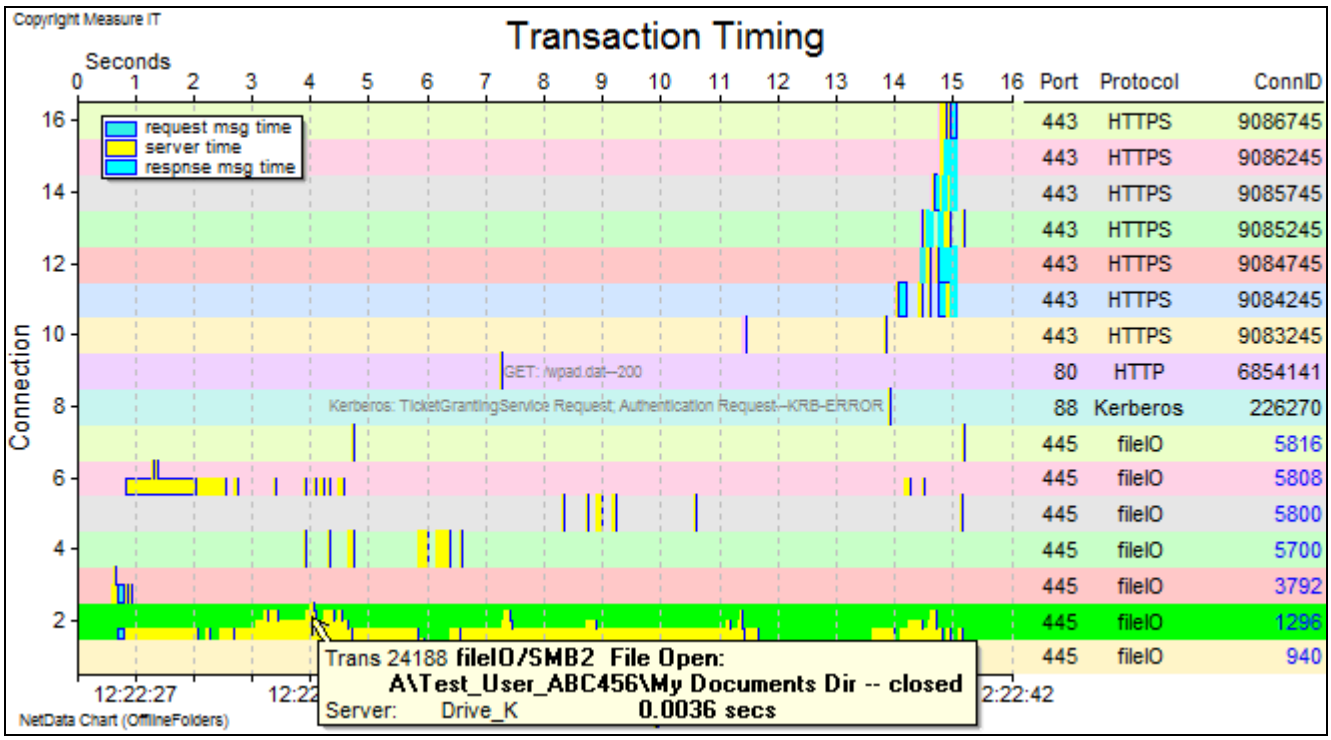
A group of NetMon 3.4 decoding controls is found in the menu of Miscellaneous Decoding Parameters on the Decoding page of controls. A checkbox configures NetData to record packet prefixes and trailers in the frame-description column of the packet table.



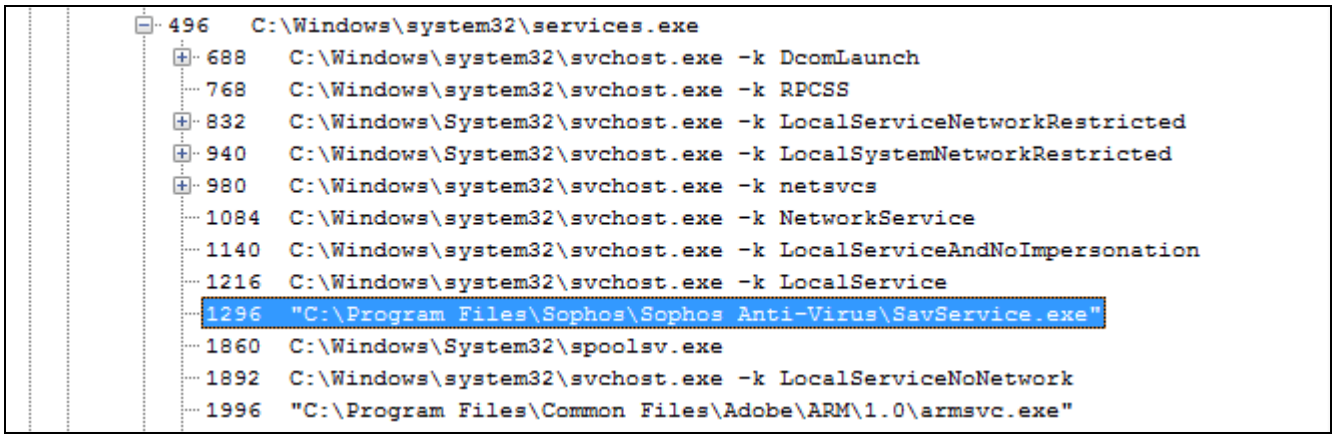
Three options in a drop-down menu determine the values assigned to the secondary connection IDs of SMB2 transactions. Normally, the secondary ID is a function of the transaction’s file ID and when secondary connections are allowed on timing charts they separate the activity of different files. The new options, only relevant to capture files derived by NetMon3.4 from ETL files, assign the ID of

either the process or the thread that initiated the file access, allowing timing charts to separate the activity of different processes or threads.

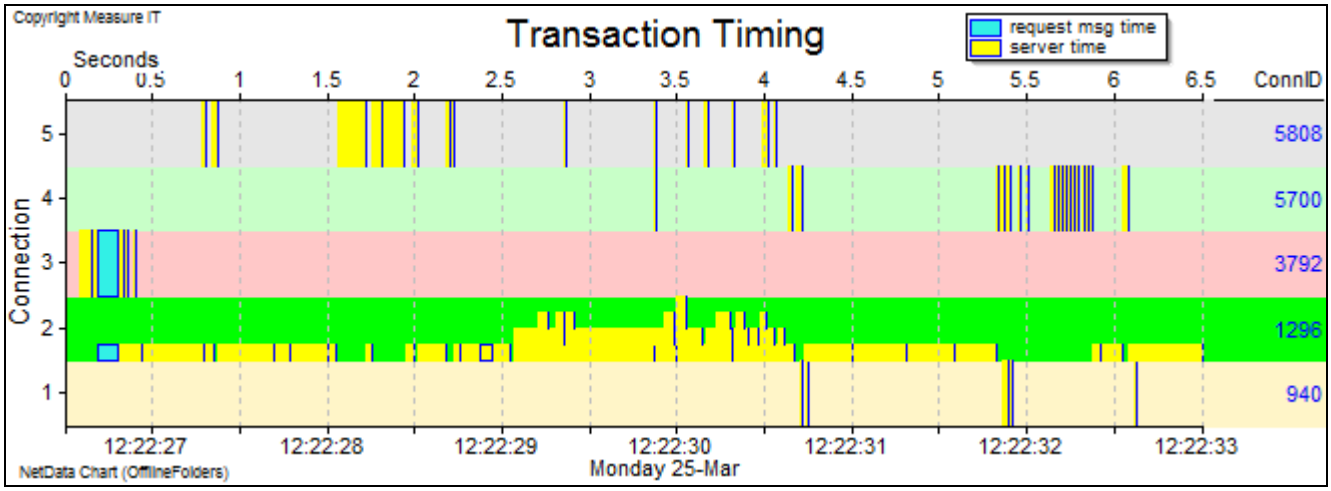




This transaction-timing chart summarises network activity while Internet Explorer was started and downloaded its first page. The seven bands at the top of the chart plot bars of the HTTPS transactions that downloaded the various files needed to render the page. The bands below the Kerberos authentication request display the file-IO activity referring to files and directories of Drive K on a remote file server. The connection IDs printed in blue, being NetData secondary connection IDs, are Windows process IDs. This chart indicates that most of the file-server activity was generated by Process 1296, which is described in the log file as being launched by an anti-virus program:



Other charts and tables revealed that the anti-virus file-IO activity repeatedly interrogated the timestamps of five co-located directories starting with My Documents, using 16 threads and sometimes accessing the same directory four times concurrently.



The file-server activity of four Internet Explorer processes dovetailed closely with gaps in the anti-virus activity (on the green band), suggesting that the overall time depended on some critical resource such as a processor core or a file-server communications service. The latter seems more likely because the anti-virus process always completed a cycle through its five subject directories before another process used the SMB connection with the file server.

22.5 Linux 'Cooked-mode' Captures

TCPdump – or, more specifically, libpcap – running under Linux can monitor specified network interfaces, in which case the capture files are in the normal 'raw' format, or can monitor 'any' device, in which case captured packets are recorded in the Linux 'cooked' format. Instead of a 14-byte Ethernet header this format provides a 16-byte pseudo header comprising a packet type, a device (address) type, an address length, up to 8 bytes of a link-layer source address, and an ether-type code. NetData handles Linux 'cooked' headers and constructs a fake destination MAC address of 6 bytes that concatenates two zero bytes, the 2-byte device type, one byte for the packet type, and one byte for the address length. NetData labels packets in this frame format as 'LxC' in the frame-description column of the packet table.

22.6 HP-UX Tracing Tool Nettl

NetData decodes packet-trace files produced by Hewlett-Packard's HP-UX (v10, 11) tracing and logging tool *nettl*, whether IP packets have been recorded with or without Ethernet headers. A capture file may have a mixture of frame formats, and the header for each frame provides such information as header length, device ID, process ID, user ID, frame format and frame kind – as well as the usual indications of packet length and a timestamp. NetData displays the additional information in the frame-description column of the packet table, for example:

```
d-1 p1569394 u43647 IP PDU Out (38)
```

Following the three identifiers, 'IP' signals the IP-only frame format, 'PDU Out' is the frame kind (Protocol Data Unit output), and the final number is an unknown, varying parameter that follows the device ID in the frame header.

22.7 TCPdump Null File Format for IP-Only Frames

TCPdump's capture-file header includes a 4-byte number that specifies a link type and determines the format of packet frames recorded in the file. A value of one, for example, specifies Ethernet, and a value of 101 specifies 'raw IP' – packets without an Ethernet header. A value of zero, however, is regarded as the 'Null' format and the file relies on a 4-byte number prefixed to each frame to indicate the frame's format. Frame-format codes are Protocol Family (PF) codes as defined in the header file *socket.h*. TCPdump running under AIX might use the Null file format with a frame-format code of PF_INET (= 2) to record IP-only loop-back packets or packets traversing the backplane between LPARs.

NetData handles the TCPdump Null file format in files with a mixture of Ethernet and IP-only frames.

22.8 Citrix NetScaler Nstrace Files

Citrix NetScaler is an application delivery system that provides load-balancing, content-switching and related functions. It includes a tracing utility called *nstrace* capable of capturing filtered Ethernet packets traversing the NetScaler packet engine, and writing capture files in either the common TCPdump format or a proprietary format. A native NetScaler trace file records additional information such as each packet's network interface number; its VLAN tag; the IDs of its Protocol Control Block (PCB) and a linked PCB; its core number; and whether it was received or transmitted.

Citrix originally developed a plug-in decoder for its own version of Ethereal to analyse NetScaler files, but the Citrix decoder is now included in standard releases of Wireshark.

NetData can analyse NetScaler trace files, but the present release handles only versions of nstrace that record frame headers in the V23 format.

Nstrace itself has been released in two major versions, V10 and V20. The major version is identified in the trace file's first record, the file's signature that NetData records in its log file:

```
Detected a NetScaler V20 trace file
   File signature (1)      NetScaler V20 Performance Data
                           NetScaler NS9.3: Build 62.4.nc, Date: Apr 17 2013, 08:32:52  n
      capture started (7)  07:28:12 04/09/13 (-1)
      file header size     104 bytes
      first frame at       07:28:12.0255 04/09/13
```

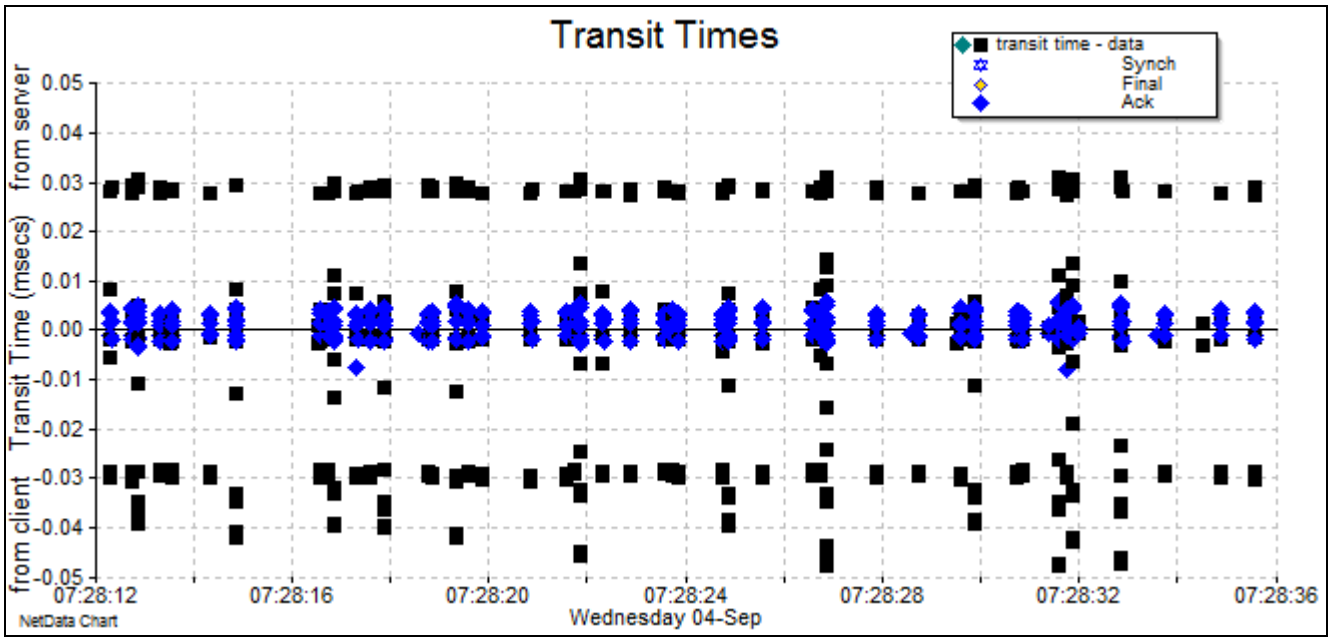
This example indicates that nstrace ran in NetScaler Release 9.3.

NetData records the additional NetScaler frame information in the frame-description column of the packet table. The acronym TXB refers to a packet buffered for transmission; TX to a transmitted packet; RX to a packet received before NIC pipelining; and NEW_RX to a packet received after NIC pipelining.

Packets traversing the NetScaler engine may be recorded twice, when they are received and when transmitted. If requested on the Input page of controls, NetData will find the matching copies and measure the engine's transit time for those packets:

	Time Of Day	Seq	Source	Destination	Len	Transit...	Description	IP ID
■	07:28:22.8656	2292	10.2.213.132: 9674	10.97.31.40: 443	383		TXB NIC 3; PCB 687E1400 link E9571600h; VLAN 19;...	248
■	07:28:22.8685	2293	10.97.31.40: 443	10.2.213.132: 9674	623		RX NIC 3; PCB 687E1400 link E9571600h; VLAN 19; ...	1405
◆	07:28:22.8685	2294	10.2.213.132: 9674	10.97.31.40: 443	64		TXB NIC 3; PCB 687E1400 link E9571600h; VLAN 19;...	251
■	07:28:22.8685	2295	10.2.213.134: 443	10.2.213.132:53720	608	0.000029	TXB NIC 11; PCB E9571600 link 00000000h; VLAN 0; ...	253
□	07:28:22.8686	2296	10.2.213.134: 443	10.2.213.132:53720	608		RX NIC 11; PCB E8571600 link 00000000h; VLAN 0; c...	253
◆	07:28:22.8686	2297	10.2.213.132:53720	10.2.213.134: 443	64	0.000002	TXB NIC 11; PCB E8571600 link 00000000h; VLAN 0; ...	258
◇	07:28:22.8686	2298	10.2.213.132:53720	10.2.213.134: 443	58		RX NIC 11; PCB E9571600 link 00000000h; VLAN 0; c...	258

As usual, transit times can be plotted on the data-sequence (Trip Times) chart in place of round-trip times:



The above chart shows that the transit times of all ack packets were less than 10 microseconds, whereas most data packets had a transit time of 30 microseconds.

Nstrace records packets in memory pages of 8 Kbytes and won't let a packet cross a page boundary. Consequently most pages end with some unused space that must be skipped when an nstrace file is analysed. Analysis of one trace file has revealed that when switching from one page to the next, timestamps often regressed with clock jumps of up to 2.3 seconds. Although this is a small sample of nstrace output it casts doubt on nstrace timing accuracy.

23 Miscellany

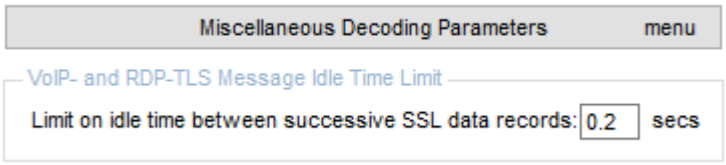
23.1 Characterising Transactions of Unknown Types

Encrypted transactions such as those using HTTPS are usefully characterised according to reversals in the direction of data flow and the lengths and types of the SSL records that form their request and response messages. However, if the protocol allows multiple transaction to run concurrently, and responses don't appear in the same order as their requests, it is not possible to always match responses with their requests. The HTTPS decoder can then record totally misleading response times.

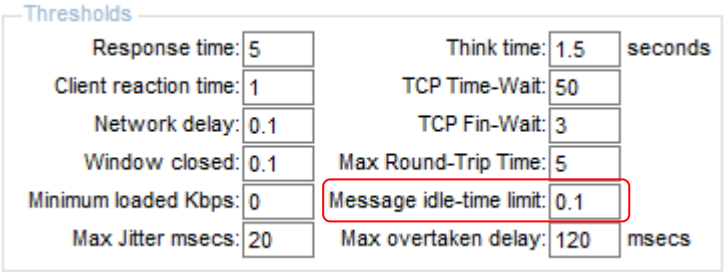
Another difficulty arises when not all messages form part of a request-response pair but flow independently in both directions. This is particularly the case for communication with 'thin' clients and in remote-control protocols such as Microsoft's RDP (Remote Desktop Protocol). For such protocols NetData uses a different technique that forms pseudo transactions from apparent round-trips that may start from either the client or the server, and unidirectional flows of data.in either direction, without responses. A short idle time such as a tenth of a second is assumed to mark the end of a transaction.

This scheme reveals patterns of dialogue activity on both the performance and timing charts for correlation with other system behaviour but avoids presenting pseudo transactions with misleadingly-large response times. Before accepting NetData's measurements from one of these pseudo transactions their credibility should be assessed by examining the transaction's packet-timing chart.

The transaction-controlling idle period can be changed. For protocols employing TLS the period is a miscellaneous decoding parameter under the heading 'VoIP- and RDP-TLS Message Idle-Time Limit' on the Decoding page of controls:



For non-TLS protocols such as Microsoft OMHS (Operations Manager Health Service), and the *unspec* decoder reserved specifically for handling unknown protocols, the period is set in the group of time thresholds on the Charting page of controls:



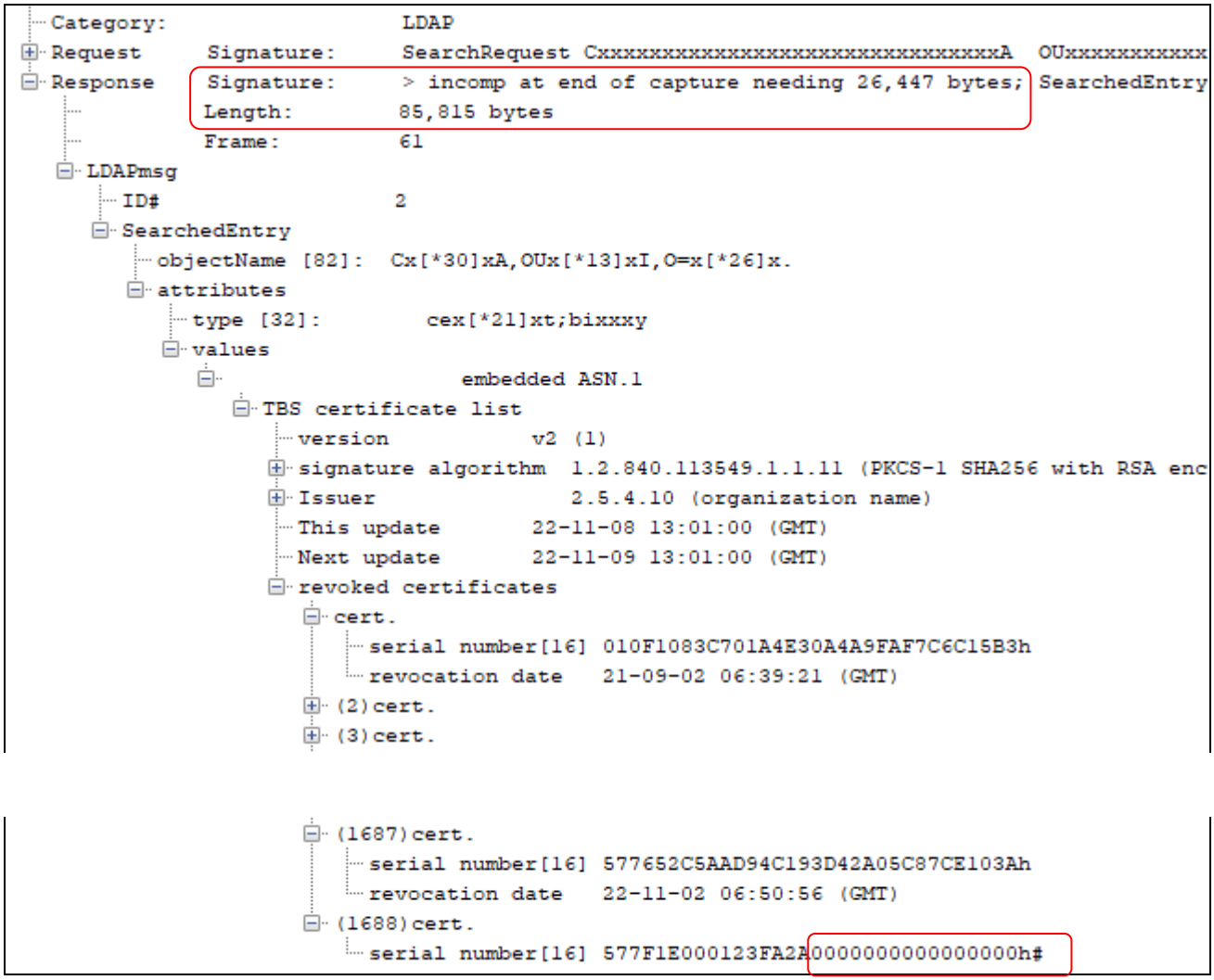
If the traffic of an unknown protocol is encountered, it is recommended that the generic *unspec* application type be assigned to its service on the dialogue chart. If the traffic does contain request-response pairs their response times will be correctly measured, and the transaction types will be usefully characterised according to their message lengths like the references to SSL records of TLS transactions.

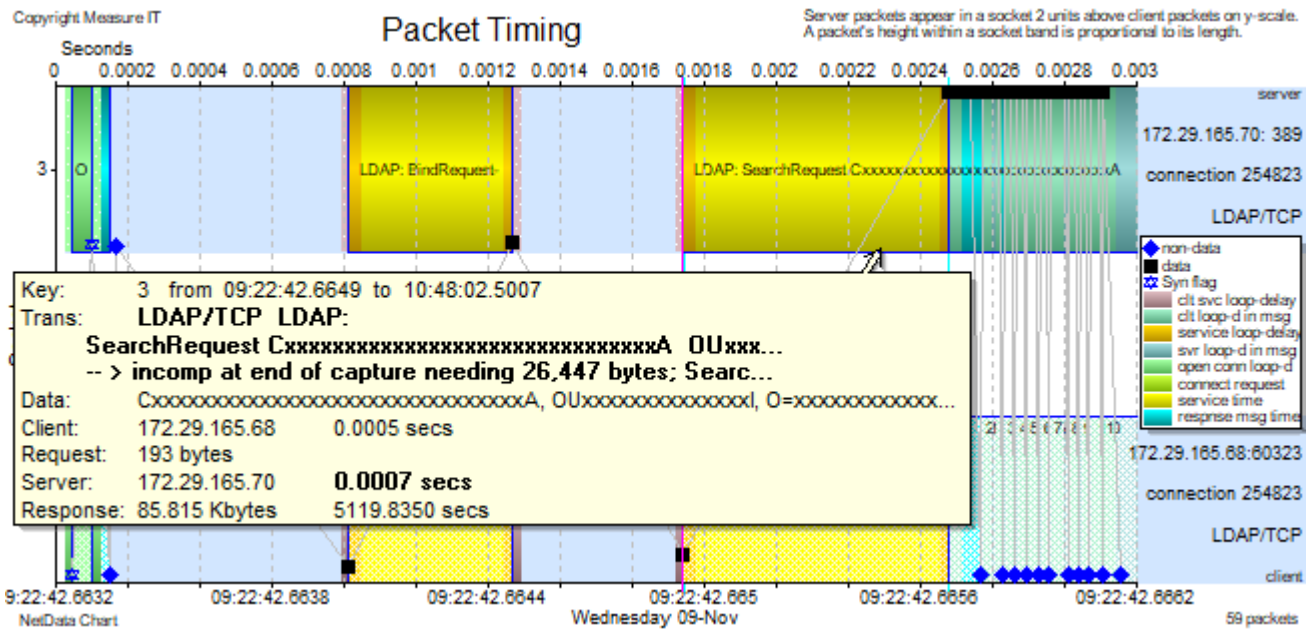
For an unknown protocol using TLS the equivalent decoder is invoked by assigning the generic *SSLrevS* application type. That decoder is automatically assigned to the traffic of a Splunk forwarder, a Mitel application server, a CyberArk PTA syslog, and Microsoft RDP.

23.2 Characterising Incomplete Transactions

If a transaction's response is not completed for any reason – perhaps because a server problem stopped all activity on the connection part way through the response message, or the capture was stopped too soon – earlier versions of NetData might record the transaction as having no response. NetData now checks whether part of a data block has been seen and, if so, records the expected length of the block as the transaction's response length, records the time when the response first appeared, and gives the transaction a response signature indicating that the response was incomplete. Either the transaction's key-data field or its response signature indicates the number of bytes needed to complete the data block, and, in order to describe the response, NetData decodes the partial response extended with null bytes to make up the required length.

In the following obfuscated example the partial response ended part way through the serial number of certificate 1688:

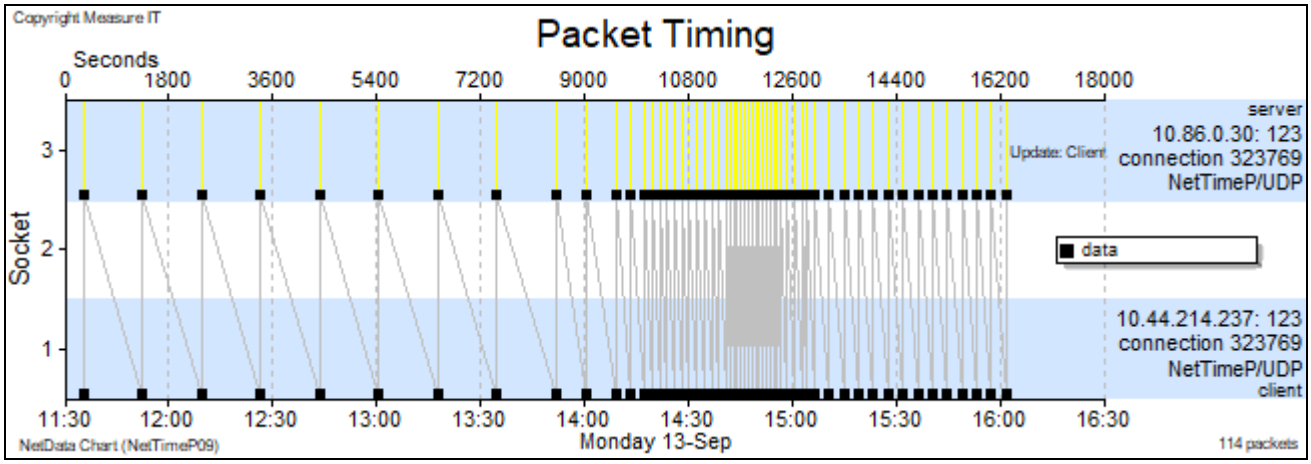



Copyright Measure IT
Packet Timing
Server packets appear in a socket 2 units above client packets on y-scale.
A packet's height within a socket band is proportional to its length.
Seconds
0 0.0002 0.0004 0.0006 0.0008 0.001 0.0012 0.0014 0.0016 0.0018 0.002 0.0022 0.0024 0.0026 0.0028 0.003
LDAP: BindRequest
LDAP: SearchRequest
server
172.29.165.70: 389
connection 254823
LDAP/TCP
non-data
data
Syn flag
clt svc loop-delay
clt loop-d in msg
service loop-delay
svr loop-d in msg
open conn loop-d
connect request
service time
response msg time
Key: 3 from 09:22:42.6649 to 10:48:02.5007
Trans: LDAP/TCP LDAP:
SearchRequest CxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxA OUxxx...
-- > incomp at end of capture needing 26,447 bytes; Searc...
Data: CxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxA, OUxxxxxxxxxxxxxxxl, O=xxxxxxxxxxxxx...
Client: 172.29.165.68 0.0005 secs
Request: 193 bytes
Server: 172.29.165.70 0.0007 secs
Response: 85.815 Kbytes 5119.8350 secs
172.29.165.68:60323
connection 254823
LDAP/TCP
client
9:22:42.6632
09:22:42.6638
09:22:42.6644
09:22:42.665
09:22:42.6656
09:22:42.6662
Wednesday 09-Nov
NetData Chart
59 packets

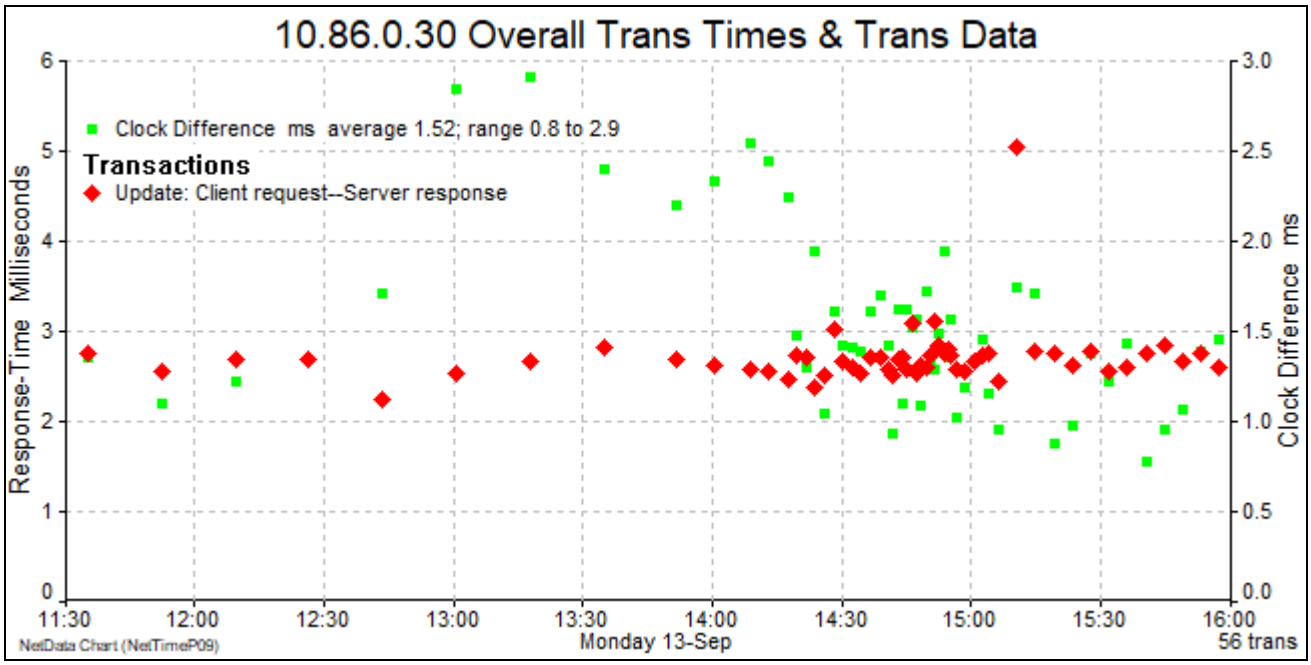
23.3 Network Time Protocol

The Network Time Protocol (NTP) is widely used to synchronise clocks between computer systems because it is designed to accommodate the variable latency of packet-switched networks. NetData describes all the fields in NTP packet headers, including the packet-extension fields of NTP version 4 (RFC 5905), and plots the response times of clock-update transactions.

In the sample of NTP traffic below the polling interval ranged from 1024 down to 64 seconds.



The red markers on the performance chart indicate the round-trip times for request and response packets to show the variation in network latency. Another indication of latency is provided by the green markers that show the difference between the reference clock reported in server response packets, and the clock of the sniffer that captured the packets. These time differences appear in the key-data column of the transaction table and can be plotted by choosing the option 'Plot Data of Similar Trans' in the context menu of the transaction table browser.

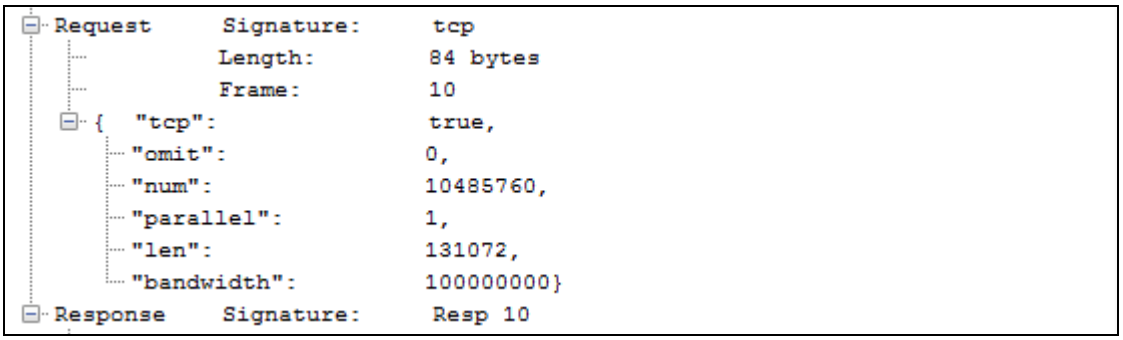


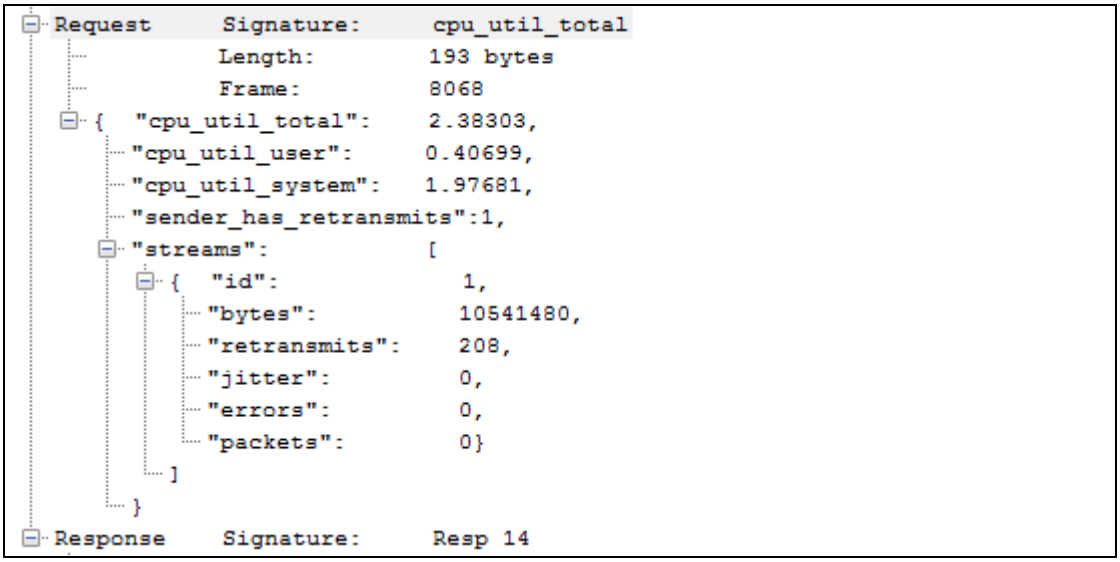
A full description of a clock-update transaction presents the values of every field in both the request and response packets. NetData clarifies the meaings of the last three timestamps in the basic headers that differ in request and response packets, whereas RFC 5095 describes only the timestamps in response packets. NetData adds the timestamp of the sniffer that captured the packet.

```
Overall response time:     0.002813 secs
                      Time of Day    From Midnight   Relative      Interval
-------------------------------------------------------------------------
 Request start:   13:34:50.612585  48890.612585     0          1023.994674 (preparation or
           end:   13:34:50.612585  48890.612585     0.000000
Response start:   13:34:50.615398  48890.615398     0.002813      0.002813 (server time)
           end:   13:34:50.615398  48890.615398     0.002813      0.000000
Start in capture file      NetTimeP09.PCAP
Client:                     10. 44.214.237
Server:                     10. 86.  0. 30:123
Connection ID:             0-323,769
Protocols:                 Network Time Protocol (NTP) / User Datagram Protocol
Key data:                  2.398 ms ahead of ref
Category:                  Update
Request     Signature:     Client request
          Length:          48 bytes
          Frame:           10439
  Header:
      Field Name                              Value
      ------------------------------------------------------------
      leap indicator                          no warning (0)
      version                                 3
      mode                                    client request (3)
      peer clock stratum                      secondary reference (6)
      poll interval                           1024 secs
      precision                               1 / 2^17 = 0.000008 secs
      root delay                              0.044678 secs
      root dispersion                         0.066101 secs
      reference clock ID                      10.0.6.1
      reference clock last set                13:17:46.618044 13/09/21, 2.86 ms after request
      origin    - last response dep. server   13:17:46.615000 13/09/21
      receive   - last response at client     13:17:46.618044 13/09/21, 3.044 ms ahead of ref
      transmit  - request departed client     13:34:50.612490 13/09/21
      sniffer clock                           13:34:50.612585
Response    Signature:     Server response
          Length:          48 bytes
          Frame:           10440
  Header:
      Field Name                              Value
      ------------------------------------------------------------
      leap indicator                          no warning (0)
      version                                 3
      mode                                    server response (4)
      peer clock stratum                      secondary reference (5)
      poll interval                           1024 secs
      precision                               1 / 2^10 = 0.000977 secs
      root delay                              0.041824 secs
      root dispersion                         0.059723 secs
      reference clock ID                      172.29.7.5
      reference clock last set                13:15:18.407000 13/09/21
      origin    - request departed client     13:34:50.612490 13/09/21
      receive   - request at server           13:34:50.613000 13/09/21, 0.51 ms ahead
      transmit  - response departed server    13:34:50.613000 13/09/21
      sniffer clock                           13:34:50.615398, 2.398 ms ahead of reference clock
```

23.4 iPerf Decoding

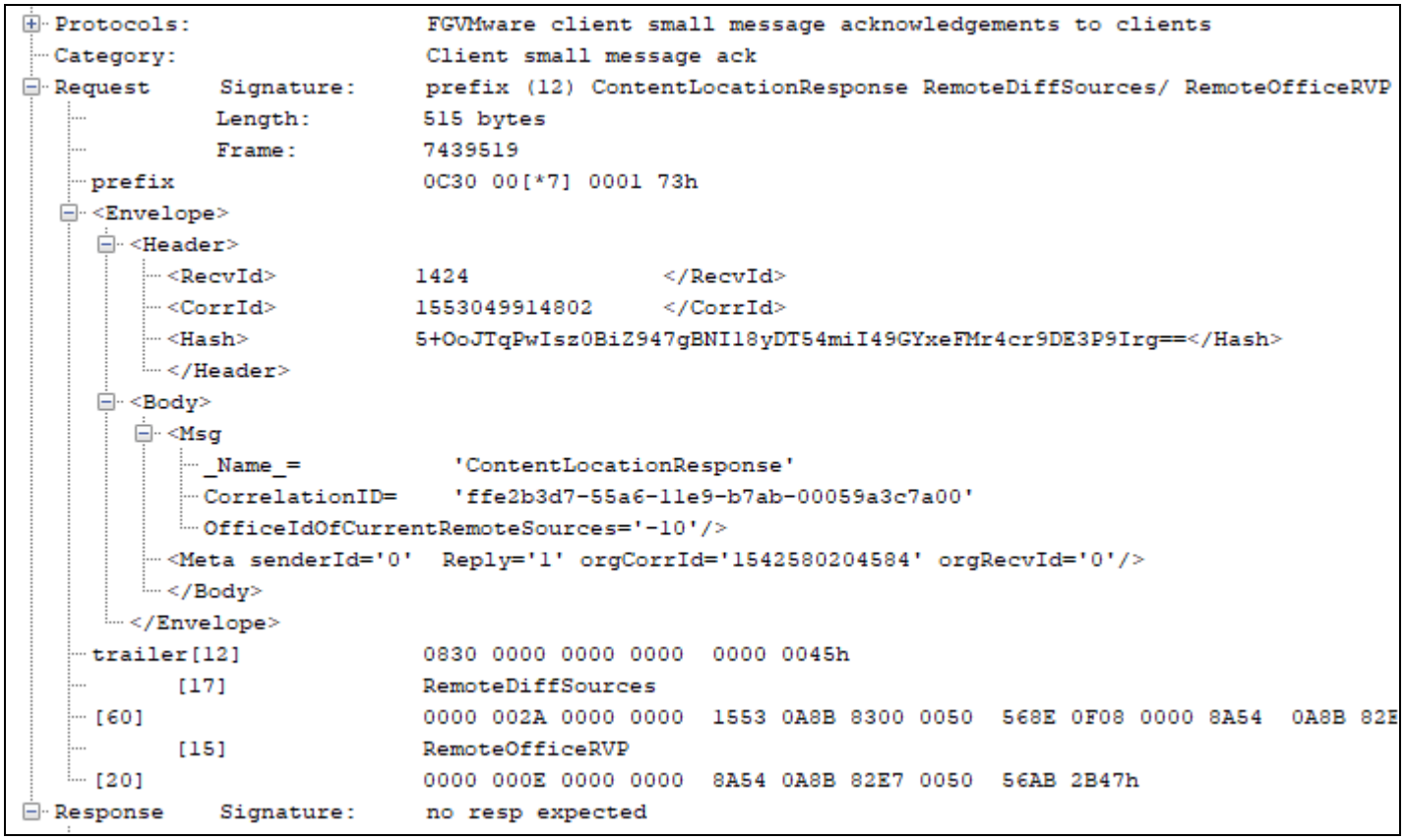
NetData now recognises the traffic of iPerf, a network speed test tool, and characterises round-trips at the start and end of a test that convey test parameters and test results:





23.5 VMware Small Messages Sent by Clients

NetData now recognises and describes small messages sent by VMware clients to other clients or a server using VMware's Background Protocol that endeavours to avoid sending messages when the network is congested. Messages use UDP and are always addressed to a port in the range 34322 to 34325. These four ports are dedicated to four message categories: for data and acknowledgements; to and from other clients or a server. Data messages and their acknowledgements are thus sent in separate UDP connections but can be matched by a common correlation ID carried in an XML document. The XML documents may have an optional prefix of binary data and an optional trailer containing binary and ASCII data.



23.6 Nessus Port Scanner

NetData recognises probing packets of the Nessus UDP port scanner.

23.7 Windows Update Delivery Optimization (WUDO)

Windows 10 has a feature that allows a PC to seek a peer on the same network from which it can copy Windows Update files. The dialogues with peers use TCP port 7680. NetData recognises this traffic which it tags as 'WUDO'. The first packets in a dialogue carry the words 'Swarm protocol':

23.8 Mapping of Airline Traffic over Internet Protocol (MATIP)

SITA (Societe Internationale de Telecommunications Aeronautiques) in association with several airlines and other service companies developed messaging protocols that were essential for the nascent commercial aviation industry more than 50 years ago. Basic messaging systems relied on teletype terminals, and online users such as reservations staff relied on block-mode video terminals exemplified by the IBM 3270 and Univac Uniscope families – sometimes known as green screens. Several old protocols are still in use although the video terminals are emulated by PCs and messages are encapsulated for transfer through TCP/IP networks. A widely used encapsulation scheme is Mapping of Airline Traffic over Internet Protocol (MATIP) defined by RFC 2351.

NetData decodes two versions of MATIP: the original version 1 that handled transactional traffic (type A) through TCP port 350 and messaging traffic (type B) through port 351; and version 3 conveying transactional traffic with emulated Uniscopes that complies with SITA protocol P1024C. Comparable traffic designed for 3270-compatible terminals is covered by SITA protocol P1024B. NetData is unable to decode the payloads of version-1 traffic because they are not specified by MITAP, but NetData fully decodes the escape sequences and control codes in Uniscope messages to provide an effective indication of the text that would appear on a Uniscope screen:

```
Request     Signature:     :0 SI>ps*,nam
            Length:        80 bytes
            Frame:         479
   channel                 02F0 808E 8172h
   address <SOH>           1./X28PSB/0666/D/MH/MH/AU/ADL/   /24080//
   text    <STX>           }S{
                           <ESC VT row:0 col:0 SI>ps*,namxxxx
   tail    <ETX>           9395 90h

Response    Signature:     <ESC VT row:1 col:0 SI><RS start entry>PS: MH.../..JUN ADL,NAM
OP/NAM
            Length:        387 bytes
            Frame:         495
   channel                 02F0 808E 8172h
   address <SOH>           01/X28PSB/0666/D/
   text    <STX>           <ESC VT row:0 col:0 SI>
                           <ESC VT row:1 col:0 SI><RS start entry>PS: MHxxx/xxJUN ADL,NAMxxxx
                                   OP/NAM<ESC b: erase to end line>\^
                           333/B     GTD/18   POS/GATE BDT1300 SD1430 ED1430   SA2045   FT0745
                                   <ESC b: erase to end line>\^
                             1. 1xxxxxxxx/JOAN+  BI2    SN12C    O KUL O PSM ASR TKNE DOCS ONE
                                   <ESC b: erase to end line>\^
                             2. 1xxxxxxxx/MERC+         SN14D    O KUL O PSM ASR TKNE DOCA DOCS
ONE
                                   <ESC b: erase to end line>\^
                           CAT/000/000 ***SVC DESK 542***<ESC b: erase to end line>\^
                           <RS start entry><ESC b: erase to end line>
   tail    <ETX>           9395 90h
```

In a traditional Uniscope message address information is enclosed by SOH and STX control codes, and the message text is enclosed by STX and ETX. A block-check follows the ETX but in this emulation it is always the same three bytes. The P1024C protocol inserts extra header information and provides a longer address text instead of the Uniscope's three characters for Region, Station and Device IDs (RID, SID and DID). In NetData's descriptions of messages, escape sequences and ASCII control codes are enclosed by angle brackets. A common sequence sets the row and column of a cursor position, starting with ESC VT (Vertical Tab) and ending with SI (Shift In) that for a Uniscope sets screen writing in Unprotected mode. To avoid the need to specify all the characters on the screen, lines of text are often terminated with an <ESC b> sequence that clears all characters to the end of the line, prior to a new-line sequence (displayed by NetData as \^).

23.9 SITA Baggage Source Messages (BSM)

NetData now decodes baggage-source messages that are sent for each item of airline baggage from the departure airport via a central server to the destination airport. The following message example changes an earlier baggage message sent from the central server to an airport:

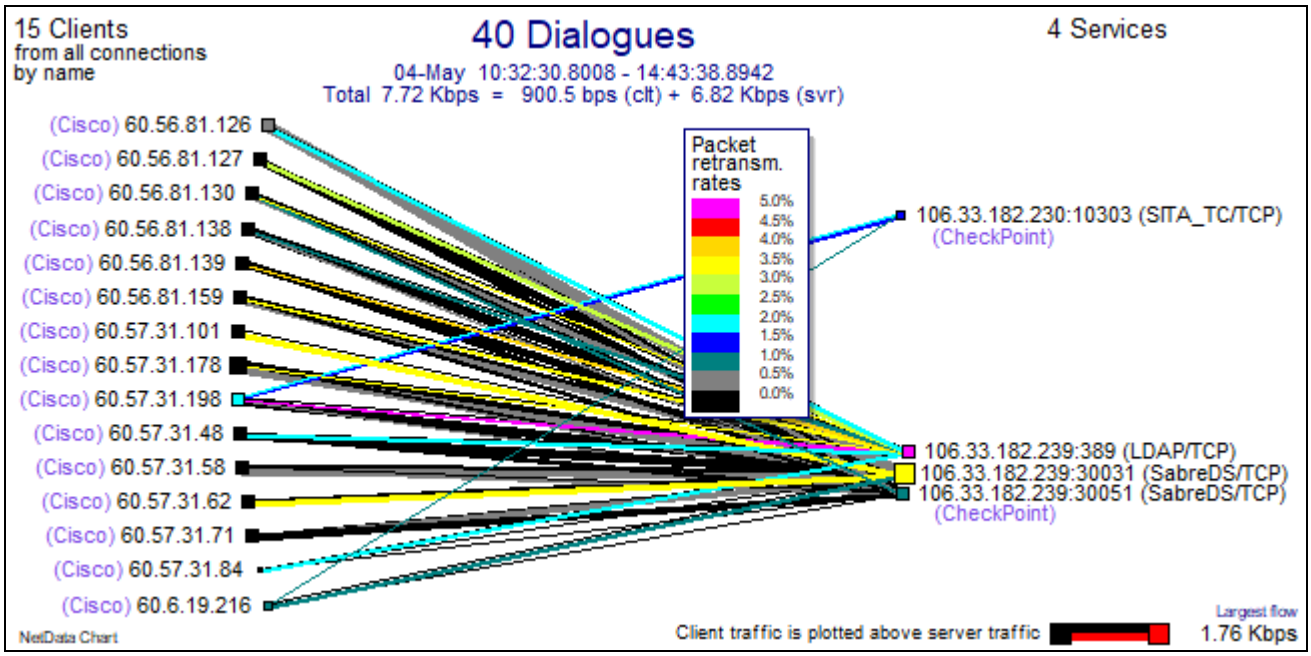
```
Request      Signature:      2-5 BSM\^CHG
             Length:         170 bytes
             Frame:          6229
   LONASI01       type       2-5
   sequence                  7792  0
   Message:                  AM25
   BSM
   CHG
   .V/1LADL
   .F/QFxxxx/xxJUN/SYD/J
   .N/008124xxxx001
   .S/Y/01F/C/041/041///A
   .P/xxxxx/xxxxxMR
   .R/PBT:0000000xxxx72-008124xxxx
   ENDBSM
```

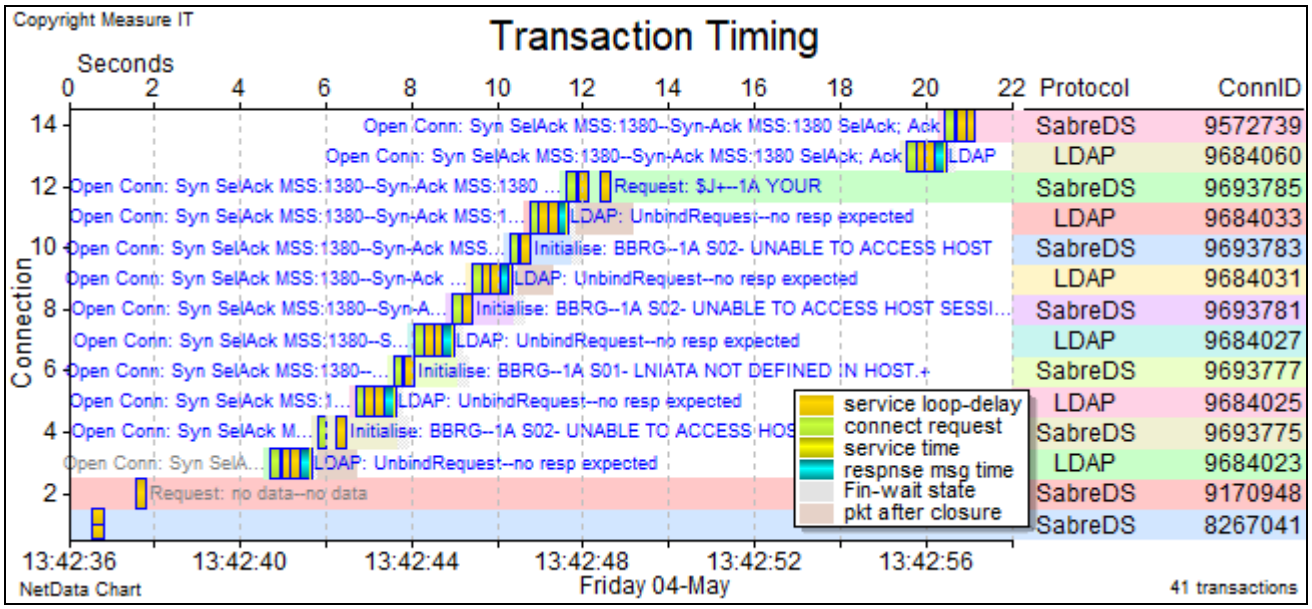
23.10 SITA Airport Messaging and Sabre Data Source

NetData decodes the messages and transactions of two more SITA protocols likely to be found in baggage-handling areas of airports. In one protocol, all messages begin with the two characters ‘TC’ and NetData tags this traffic as ‘SITA_TC’

In the second protocol all messages begin with the two bytes FE01h. Except for a connection’s first message that initialises a server session, all headers contain 52 bytes of binary codes with an embedded four-letter transaction ID in EBCDIC. The payload also uses EBCDIC and conforms to the message formatting rules of Sabre Data Source (SDS).

Sabre says that SDS provides an alternative to parsing data to and from screen displays (such as emulated IBM 3270 displays), enabling more effective extraction and utilisation of Sabre data within local applications. NetData tags this traffic as ‘SabreDS’.





On this chart a client made five attempts to open a Sabre session, each preceded by an identical LDAP query requiring four round-trips.

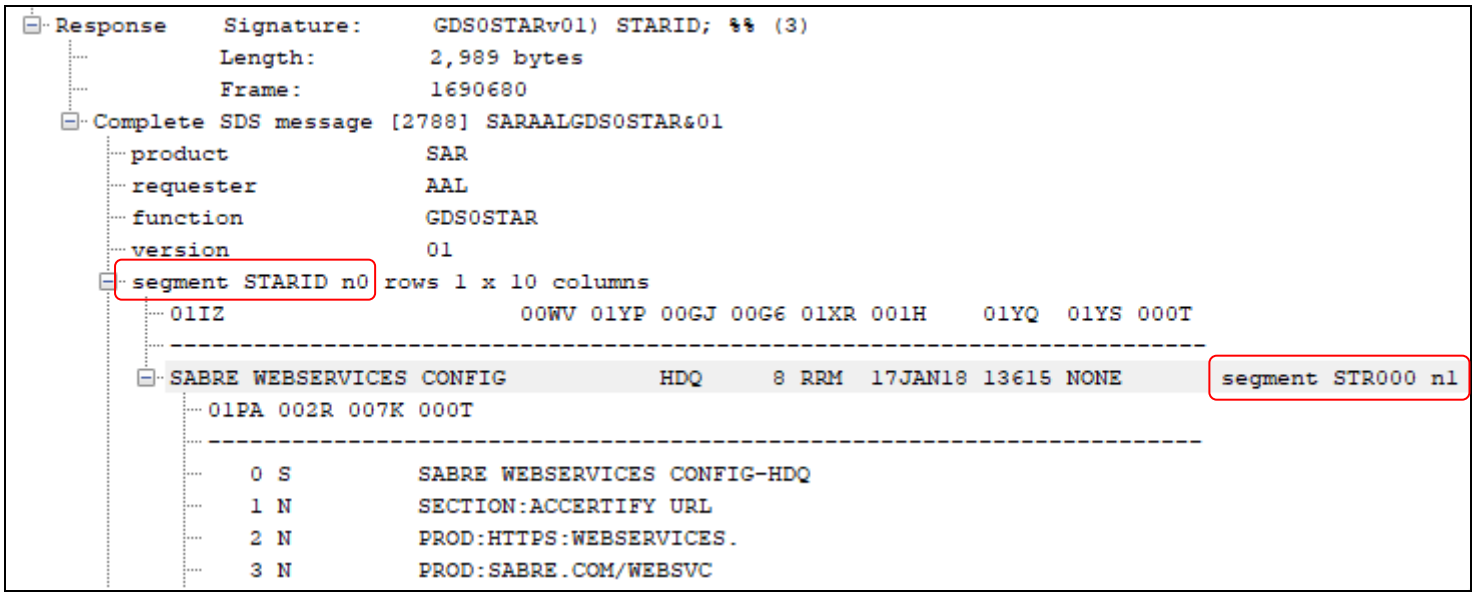
```
Response     Signature:      SDBAGTYPv03) BTTHDR; %%
             Length:         525 bytes
             Frame:          1753371
  Complete Message Definition Record (MDR) [458] AIRAALSDBAGTYP&03
    product                  AIR
    requester                AAL
    function                 SDBAGTYP
    version                  03
    segment BTTHDR n0     Optional  repeatMax 1;   elements 1
    segment BTTTBL n1     Optional  repeatMax 999; elements 8
    segment BRCHDR n2     Optional  repeatMax 1;   elements 1
    segment BRCSHD n3     Optional  repeatMax 999; elements 5
      ElementID Type MinLen MaxLen M/O RepeatMax
      ----------------------------------------
        00CP      C    1      1      O   1
        01C6      C    1      1      O   1
        01C6      C    1      1      O   1
        01C6      C    1      1      O   1
        01C6      C    1      1      O   1
    segment BRCTBL n4     Optional  repeatMax 999; elements 2
    segment BTAHDR n2     Optional  repeatMax 1;   elements 1
    segment BTYDAT n3     Optional  repeatMax 999; elements 1
    segment BTADAT n4     Optional  repeatMax 999; elements 1
```

Metadata specifying the format and content of SDS messages is provided in the form of Message Definition Records (MDRs) on request from workstations. An MDR has the same identifier as its related SDS messages and lists the types of segments that may be found in a message. Each segment has a list of elements whose four-character IDs are base-63 numbers (as above).

NetData may not find a relevant MDR in a capture before it encounters an SDS message. However, during analysis NetData builds a library of MDRs and at the end of analysis saves the library in the file *SabreMDR.ini*, stored in the same folder as *Discover.ini*. It reads the MDR file when it starts analysis, with the result that the decoding of SDS messages is informed by MDRs read during previous analyses of the same or different captures.

NetData displays an SDS message with a structure that indicates the nesting level of each segment and displays a sequence of similar segments as a standard NetData table. Any row in a table may have

nested segments forming sub-tables, like the list of STR000n1 (nesting-level 1) segments within segment STARIDn0, below.

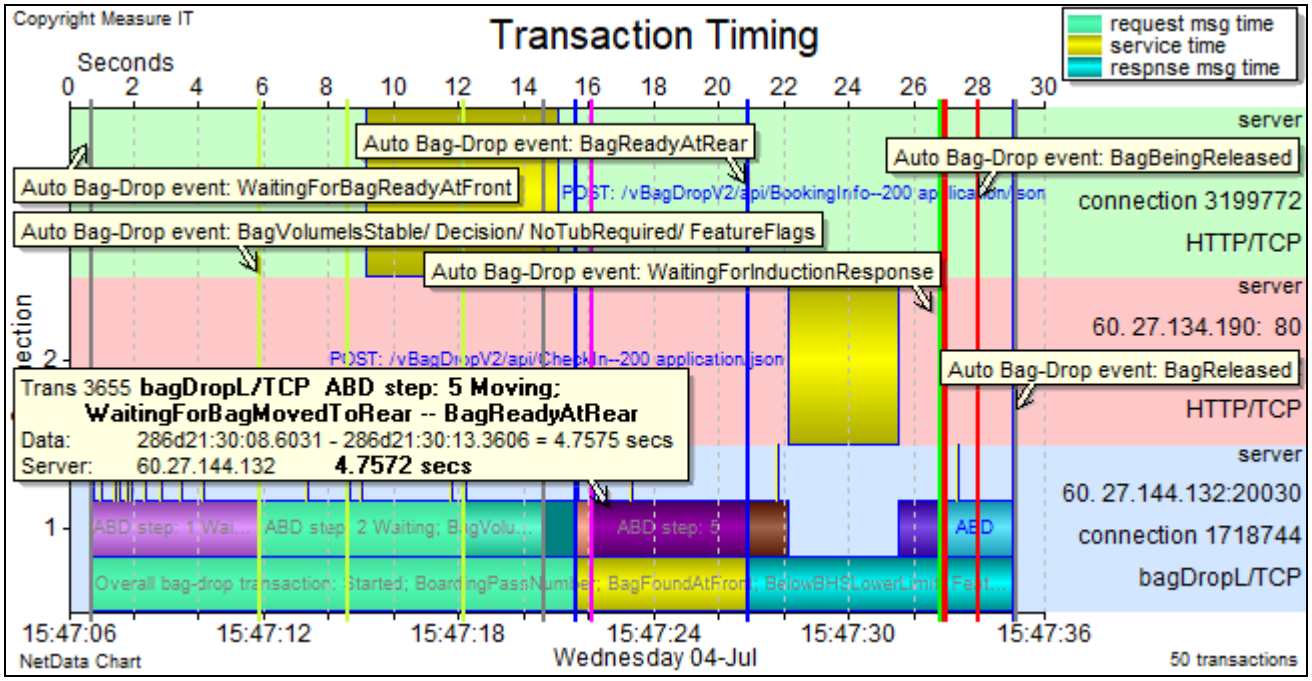


23.11 Visualising Automated Bag Drop Performance

Airport automation has brought rapid development of passenger self-service kiosks for check-in and now baggage-handling functions in Automated Bag Drop (ABD) equipment. Such equipment forms only the most visible part of large IT systems that link passengers to airport operators, airline operators and international booking services through integrated service platforms such as the SITA Transact CUSS platform. Interworking between the equipment and systems of different providers is fostered by the Common Use Self Service (CUSS) specifications of IATA (International Air Transport Association).

When performance problems arise in such systems NetData proves a uniquely powerful tool in analysing all the traffic handled by ABD systems, reconstructing transactions, and presenting comprehensive views of system behaviour over time. NetData already decodes many Sabre and SITA protocols, and to support a recent investigation added decoders for the logging messages issued by ABD equipment from Daifuku BCS Airport Technologies and ICM Airport Technics. After reading the MIB definition files from vendors, including those from SITA, NetData is able to describe SNMP Get requests and SNMP Trap messages that also indicate ABD events.

The ABD decoder extracts high-resolution timestamps and descriptive texts from event logging messages sent to a BCS server. The response times of logging transactions are not significant, and in fact the server doesn't respond to many messages. However, their time of occurrence and contents provide valuable clues to the time spent in various ABD steps, and when combined on timing charts that depict all ABD and backend transactions they highlight the critical dependencies and sources of delay. The chart below presents a complete bag-drop transaction that took nearly 30 seconds.



Vertical stripes indicate significant ABD events described by BCS logging messages and recorded by NetData as network events. The bottom, grey band presents three levels of information derived from the messages: the top level (of the grey band) comprises short vertical strips that indicate the occurrence of all the logging messages; the three bars in the bottom level describe what NetData regards as a *user* transaction covering all steps in the bag drop; and the second level presents what NetData regards as *internal* transactions that measure the time for processing steps between the

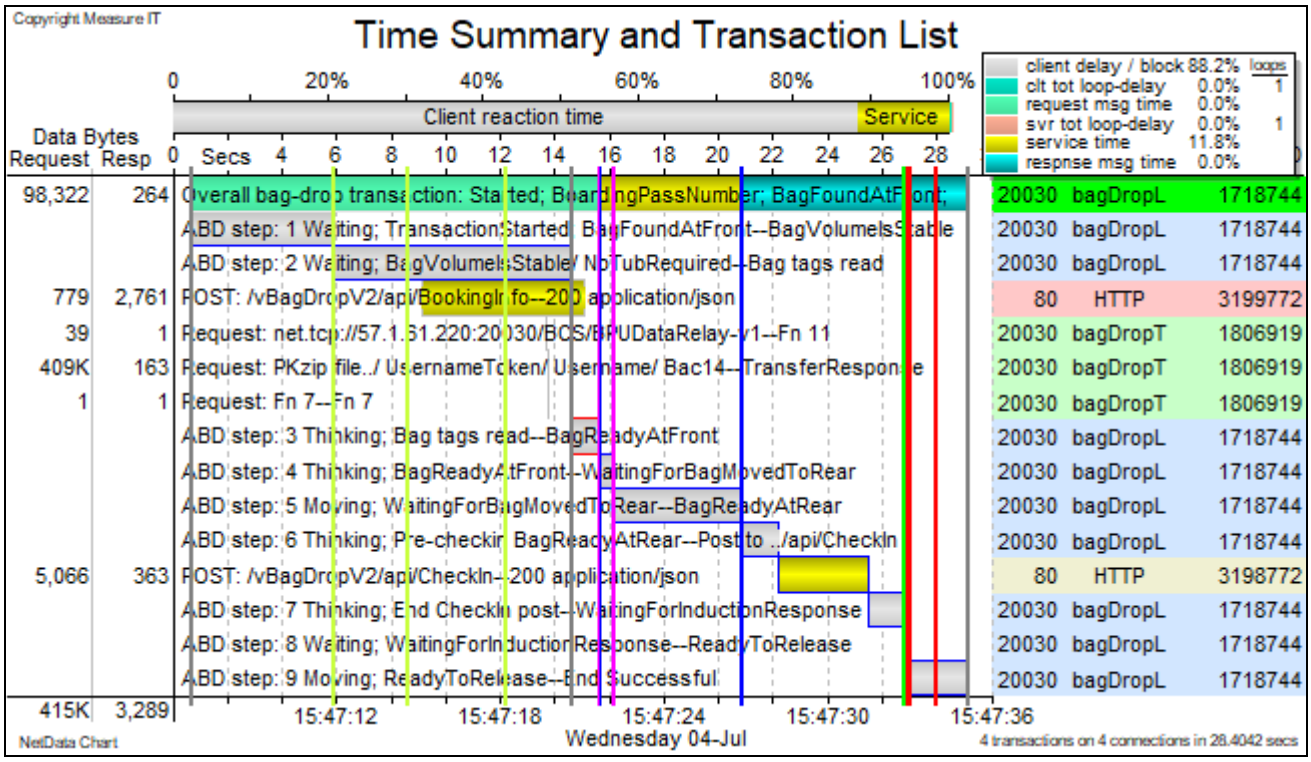
significant events. Eight processing steps are visible and leave a gap while the ABD conducted an HTTP Post transaction with a CheckIn service, as depicted on the middle (pink) band. The top (green) band depicts a Post to a BookingInfo service which in this case was not on the critical path.

The pop-up on the dark purple internal transaction (ABD step 5) displays the difference in logging-packet timestamps as server time, and also displays the two high-resolution ABD timestamps for comparison.

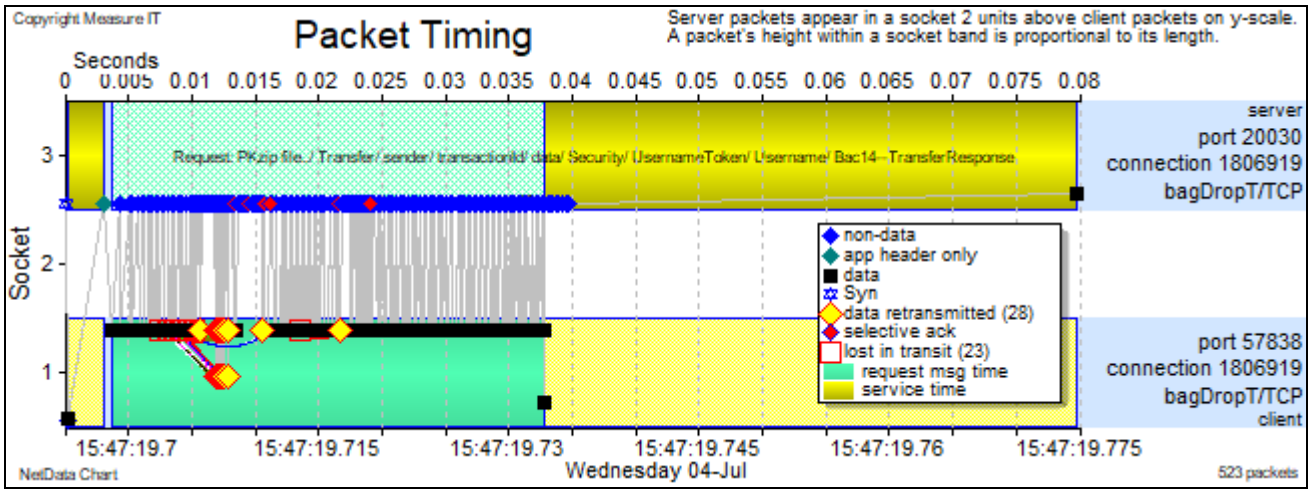
NetData assigns different colours to the various network events during analysis, and when loaded from the database they appear automatically on the performance and timing charts. Individual event stripes can be hidden, or their colours changed, in the event table.

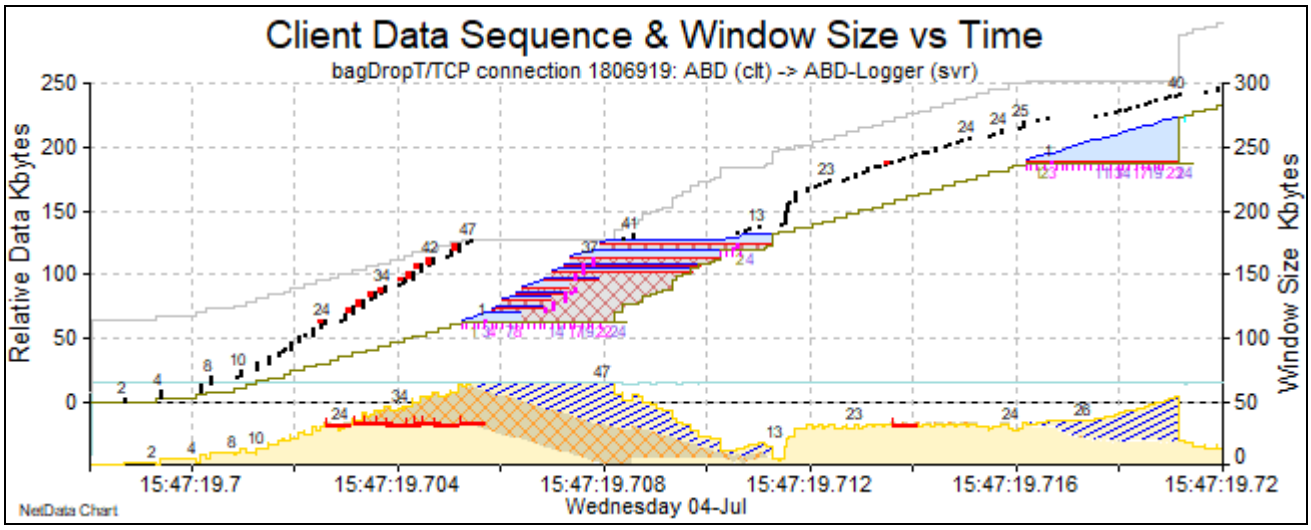
Internal transactions are assigned different colours when they are loaded for charting, but their transaction bars on the timing chart will revert to the standard yellow colour if the 'Colour families' box is unchecked. Different colours for the transaction bars, and for markers on the performance chart, can be assigned as Highlight colours on the performance chart. New options in the context menus of the performance and timing charts allow a Highlight colour to be assigned to all the transactions of the same type as the selected transaction.

Internal transactions can be assigned to a transaction family. The waterfall chart below depicts a family whose parent is the user transaction that embraces a passenger's complete bag-drop operation. NetData can select internal transactions for the critical path through parallel activities, but the internal transactions are painted grey on this chart because they contribute to client ('grey') time in the time-summary bars at the top of the chart. The resulting waterfall chart documents all the major steps and other activities in a bag drop, with timings displayed on a true time scale:



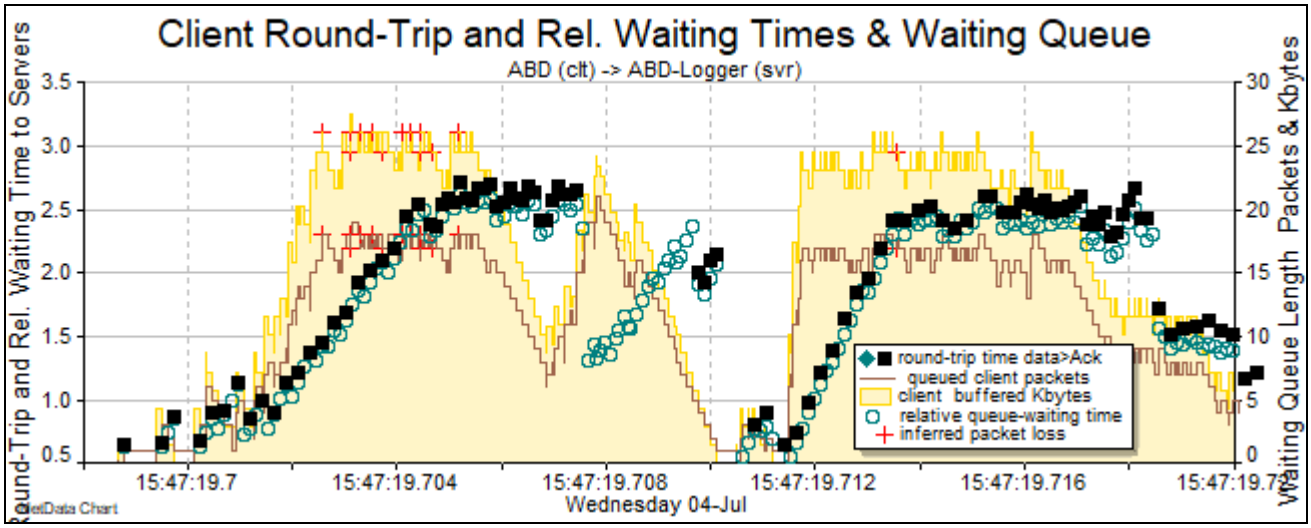
Two large transactions with external servers were not on the critical path: a 6-second Post that requested booking information, and a 400KB logging message. The large logging message was transferred quickly but revealed a potentially serious network weakness that in this case required 28 packets to be retransmitted.





Selective acks illustrate the magnitude of the problem. The window-size graph shows that the ABD client was able to keep the server's receive window full while the lost data was recovered, but subsequently halved its congestion window and invoked the slow-start mechanism.

The cause of the packet loss became clear when NetData modelled the queueing behaviour of packets that arrived in some device at gigabit speeds but queued behind a 100 Mhps egress link



The very strong correlation between the calculated queue-waiting times from the model (green circles), and the measured round-trip times (black squares), confirms the accuracy of the model which

shows that packets were dropped when the packet queue occupied only 25 Kbytes, with no more than 19 packets.

The purpose of the large logging message became clear when NetData recognised a complete PKzip file combined with various text strings and binary data. NetData automatically saved the extracted zip file to the project folder and could display its full contents. The panel below displays the structure of the zip file by identifying the names of all eight included files, the contents of their file headers, and their positions in the zip file.

23.12 Riverbed Out-of-Path Optimisation Traffic

NetData detects Riverbed out-of-path WAN optimisation traffic that uses TCP port 7810. The TCP payload starts with a header of 8 bytes which is normally followed by an SSL record.

23.13 Inter-Juniper Accelerator Traffic

The traffic between a pair of Juniper WX or WXC WAN accelerators includes many control dialogues that use both TCP and UDP with port numbers from 3577 to 3580. NetData recognises and tags most of the different types of flows between accelerators and will display packet contents, but can interpret very little of the contents. Most of the data flows are unidirectional and NetData doesn't recognise any request-response pairs.

A description of some protocols can be found in the Juniper Networks whitepaper *Troubleshooting WX Tunnel Flaps.pdf,* which summarises port allocations in the following table:

Source Port	Dest Port	Protocol	Description
3580	3578	UDP	WAN performance probe unidirectional
3578	3578	UDP	Backup heartbeat every 5 sec. (backup device initiates heartbeat; active device acknowledge to backup device)
ephemeral	3578	UDP	Heartbeat every 5 sec when data is there, heartbeats from DC to CC; path-MTU-discovery probes; router validation probes
ephemeral	3577	UDP	Compressed data (tunnel mode: UDP)
ephemeral	3577	TCP	Self registration, communicate between registration server and secondary registration server, over SSL
ephemeral	3578	TCP	Tunnel setup, compression subnet (Netmap) exchange; dictionary info
ephemeral	3579	TCP	Cluster peer communication

23.14 BlackBerry Enterprise Server (BES)

NetData now decodes UDP management packets sent to the UDP port 52311 of mobile devices. This port number is also used for HTTP services of BigFix management products that are now incorporated in the IBM Tivoli Endpoint Manager.

23.15 Micros Systems 9700 HMS Point-of-Sale Equipment

NetData decodes SCC (cluster controller) and CAL (Client Application Loader) traffic of Micros Systems Model 9700 HMS (Hospitality Management System) point-of-sale equipment. SCC traffic uses TCP ports between 5011 and 5018, while CAL traffic uses TCP port 7300.

23.16 Bentley Systems ProjectWise

NetData detects and partially decodes the traffic of ProjectWise, project collaboration software of Bentley Systems for engineering and related professions working with the design and construction of infrastructure projects. ProjectWise traffic normally uses TCP port 5800 in application, file and other types of servers. New TCP connections start with an SSL version 3 ClientHello record, to which the

server responds with the word ACK and either an SSL ServerHello record or, if data is not to be encrypted, with the word NOSEC. NetData tracks the SSL handshaking that is interspersed with short text messages, and the encapsulation of SSL application-data records. If data is not encrypted, NetData partially decodes the headers of data messages, and measures the response times of all server round-trips.

23.17 Eclipse Test & Performance Tools Platform (TPTP)

The Test and Performance Tools Platform (TPTP), a project of the Eclipse open-source community, provides frameworks and services for building test and performance tools that can be readily integrated with other system tools. The Eclipse Project was originally created by IBM and is now supported by a consortium of software vendors. TPTP, formerly called *Hyades,* is used in the IBM Rational suite of test tools.

NetData now decodes TPTP data-loader and agent-control messages, all of which begin with the magic number 82656780h. In the Rational tools many of the agent-control messages convey Java objects, and only those Java messages form request-response pairs which NetData records as server transactions. This traffic is tagged as either *TPTPagnt* or *TPTPagtJ (*if messages contain a Java object).

The data-loader messages flow only from client to server, generally conveying execution and trace data expressed in XML, and this traffic is tagged as *TPTPdata*.

The Rational tools use a third protocol to transfer files. Messages in this protocol don’t begin with the TPTP magic number but at the start of each transfer they refer to Java objects seen in agent-control messages. This traffic is tagged as *IBMRload*.

23.18 AMF in Adobe Flex for VMware Management Consoles

AMF (Action Message Format) is widely used in Adobe Flash Remoting and Adobe Flex to convey and reconstruct serialised ActionScript object graphs. There are two versions, AMF0 and AMF3, documented in Adobe specifications: *amf0_spec_121207.pdf* and *amf3_spec_121207.pdf*.

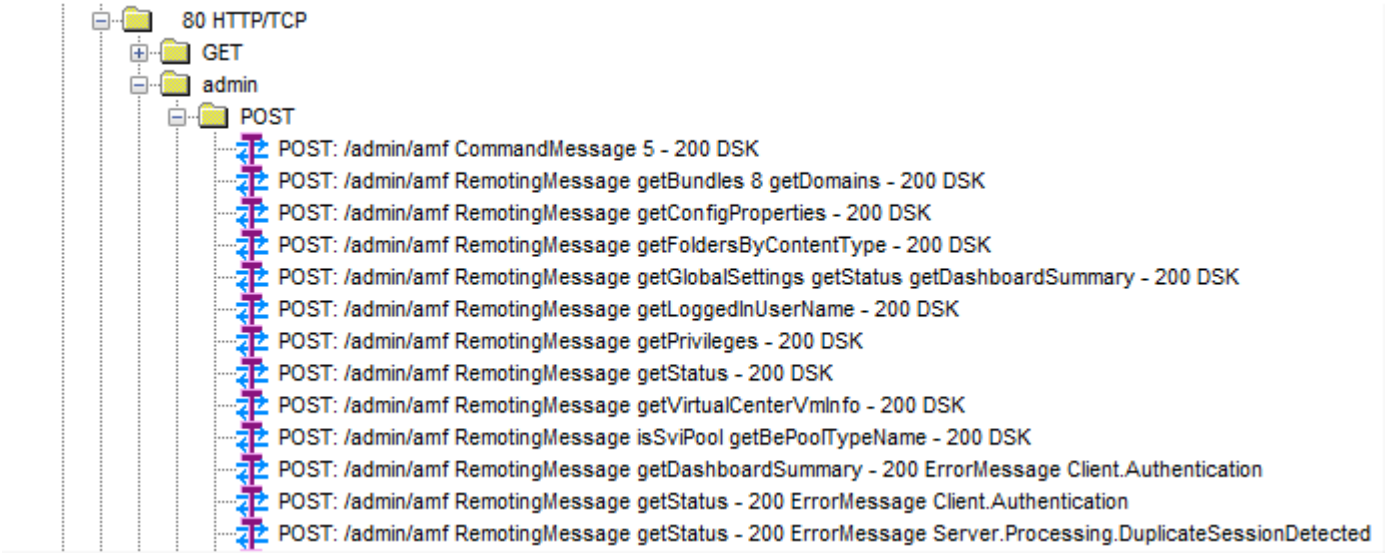
AMF is a compact binary format that uses variable-length integers and precedes values with a single-byte *type marker*. When names are associated with values in dynamic objects and some types of arrays, they are encoded as AMF strings but not preceded by a string marker.

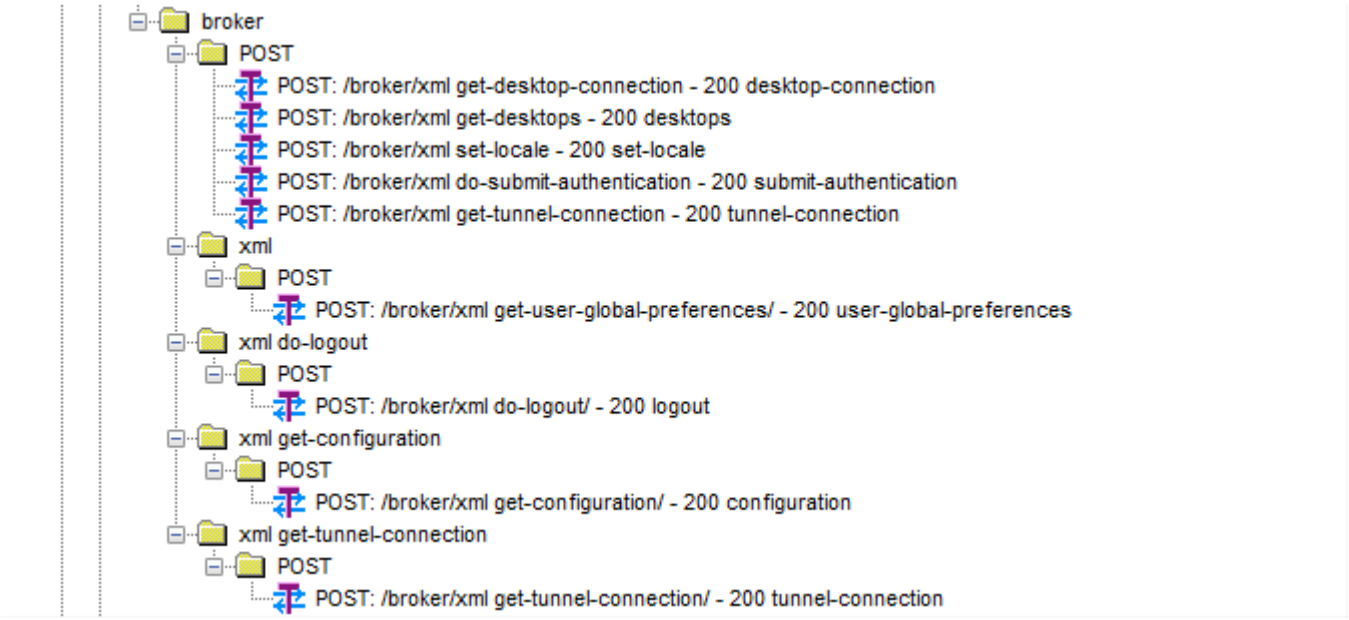
In AMF3 redundant information can be avoided by sending message components by reference to an earlier occurrence of a component. A reference is an integer usually in place of the component's length indicator, and forms a zero-based index to an implied table of components in the message. There are three different component tables, for strings, complex objects, and sets of object-trait definitions.

NetData decodes AMF and has been tested on AMF messages conveyed in the HTTP Posts and responses of an Adobe Flex package for VMware management consoles. It presents a thoroughly detailed picture of management activity in the context of user activity and signs of system stress.

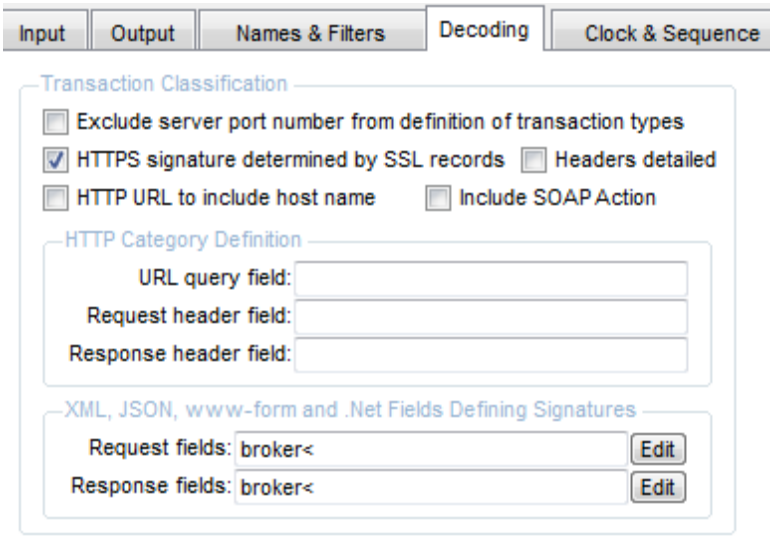
AMF messages are self-defining except for objects with *externalizable* traits whose serialisation format is known only to the sender and receiver. When NetData encounters an object with external traits it assumes that those traits contain a single AMF component. If this assumption is wrong, or NetData encounters what it regards as an invalid type maker, AMF decoding is terminated and the remaining data is displayed with NetData's normal treatment for raw data.

From every AMF message NetData extracts the name of the first object and the values of any strings named 'operation' or 'faultCode' for inclusion in the transaction's signature. The resulting signatures distinguish different types of transactions, as seen in the following extract of a transaction-class tree:





The same port 80 also handled Post requests addressed to a broker, and to distinguish the different types of requests and responses NetData controls were set to insert in their signatures the name of the first XML tag that follows a tag containing the word ‘broker’:

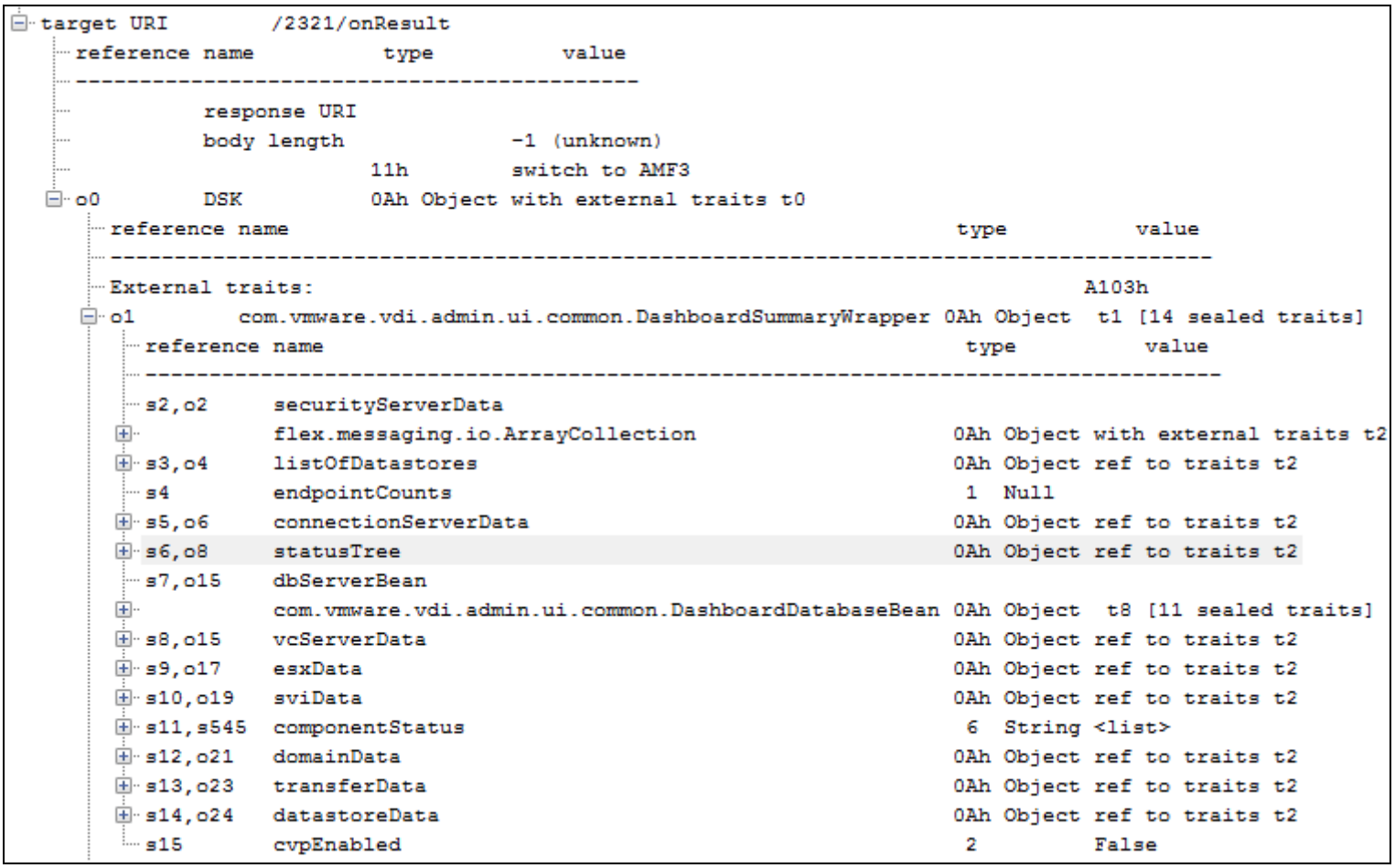


As illustrated in the following panel, NetData displays every sequence of AMF message components in a table with four columns:

- component’s index to its implied reference table;
- name if it has one;
- data type; and
- value.

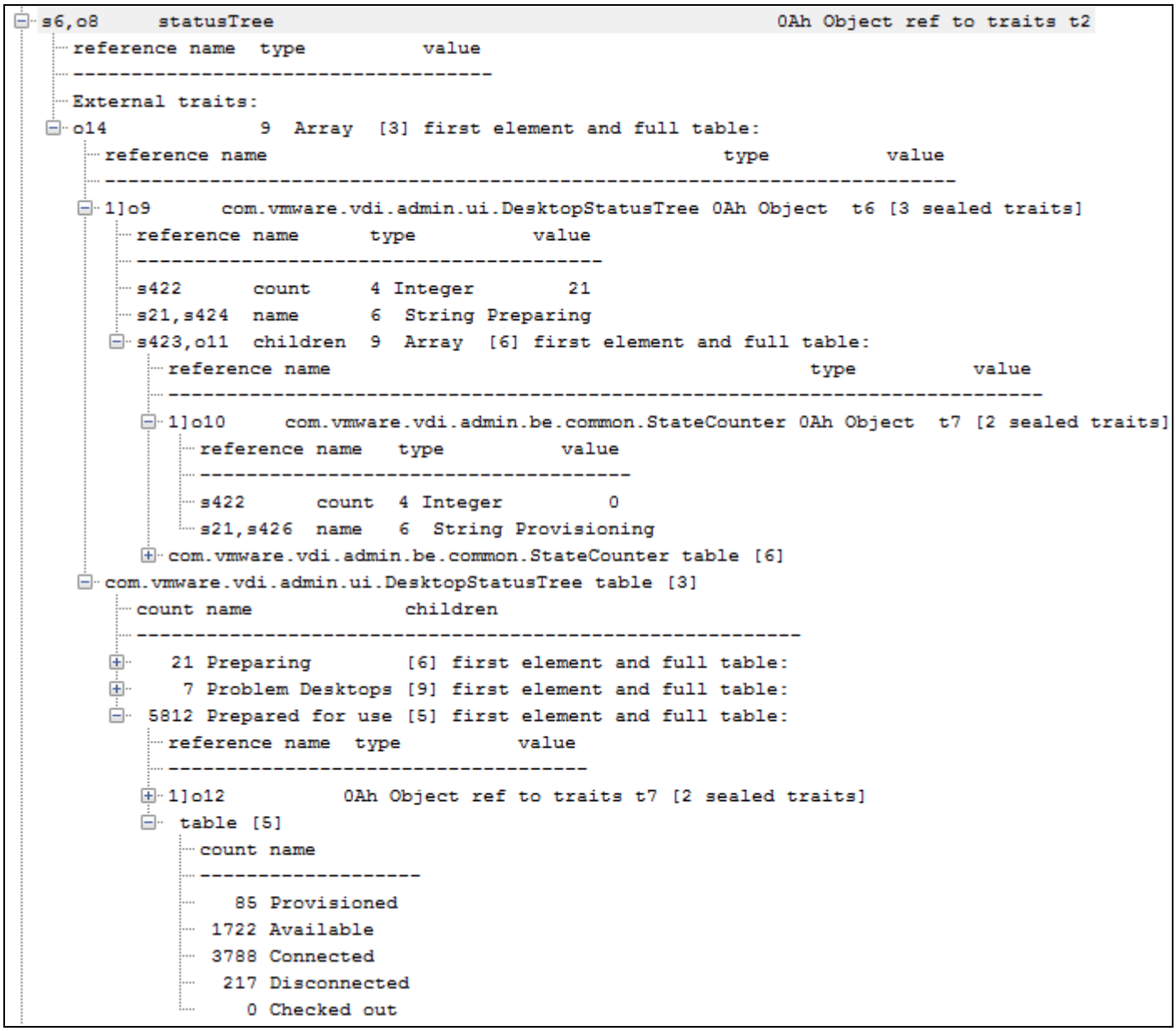
One component may have two indexes, one for its name string and the other for its value. Each index number is preceded by a letter to identify its reference table:

s strings
o complex objects (including arrays, byte arrays, dates and XML documents)
t sets of object traits



The header of every message contains target and response URIs, and a body length that may be -1 if unknown. This panel outlines a message conveying a dashboard summary. The body is an object with external traits, and its major trait is a typed object ('DashboardSummaryWrapper') with 14 'sealed' traits. Most of those traits are objects with an external trait (set t2), and that external trait is usually an array of typed objects.

NetData displays the highly recursive structure of such messages in its familiar tree format, and it is quite common for an AMF message to present tables within tables as illustrated by the following expansion of the above dashboard *statusTree* property.



The object with a single external trait appears to be a formatting artefact and if that is overlooked the panel indicates that this statusTree is an array of three *DesktopStatusTree* objects, and the third, *children* property of those objects is an array of *StateCounter* objects. The StateCounter object has two properties named *count* and *name*.

NetData describes the first component of every array in its usual four-column format. If the array has more than one component, it is also presented as a standard NetData table whose column headings are the property names of the component objects. Expand the first description to read the AMF data types and component reference numbers; and expand the table description for a concise display of the array values. Like all NetData tables in a tree structure, double-clicking the parent branch pops up a table-browser window that can be scrolled, sorted, filtered and searched. The table browser is valuable when exploring the dashboard's very large tables of desktop states and data-store states.

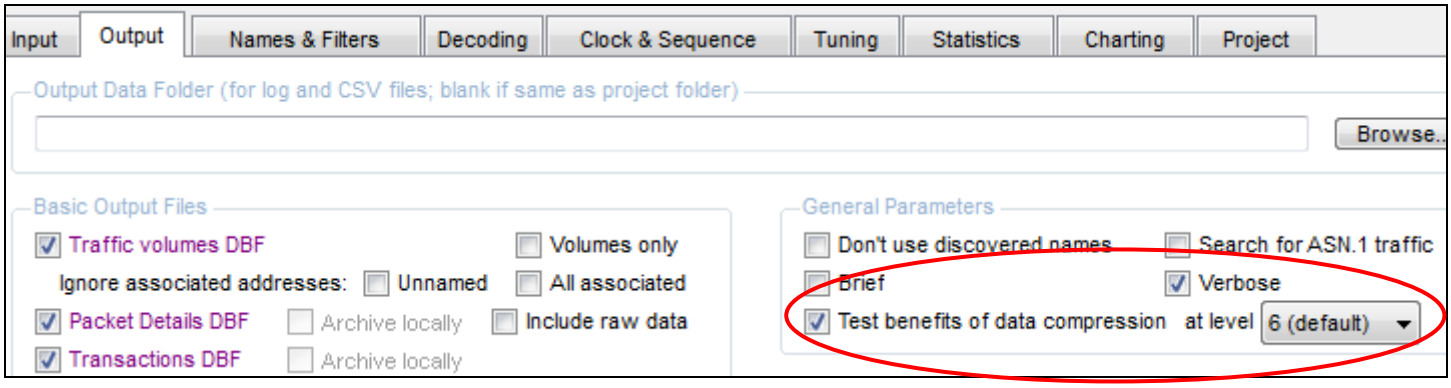
row	count	name	children
1	21	Preparing	[6] first element and full table:
2	7	Problem Desktops	[9] first element and full table:
3	5812	Prepared for use	[5] first element and full table:

row	count	name
2	1722	Available
5	0	Checked out
3	3788	Connected
4	217	Disconnected
1	85	Provisioned

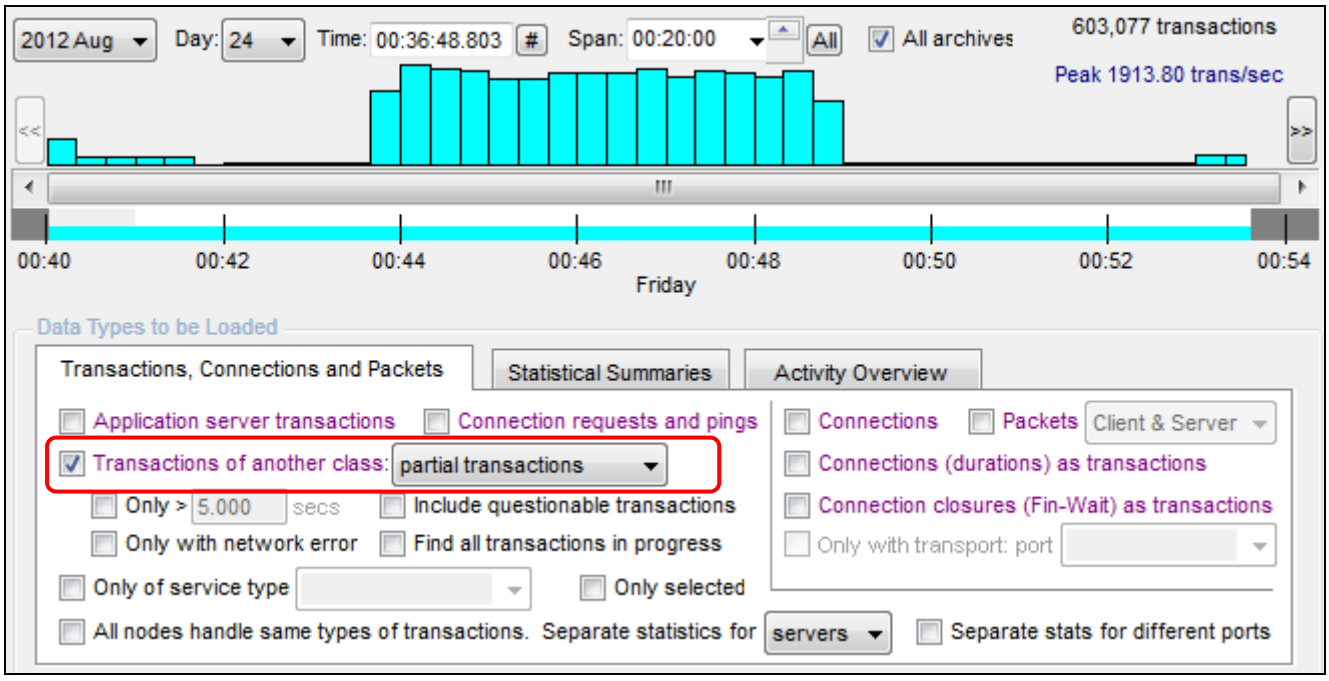
23.19 Data-Compression Benefits with Connect:Direct Traffic

When large files are transferred by Connect:Direct through a network with WAN accelerators it may be necessary to assess the WAN bandwidth required to handle the data after it has been compressed by the accelerators, even though traffic has been captured in its uncompressed form.

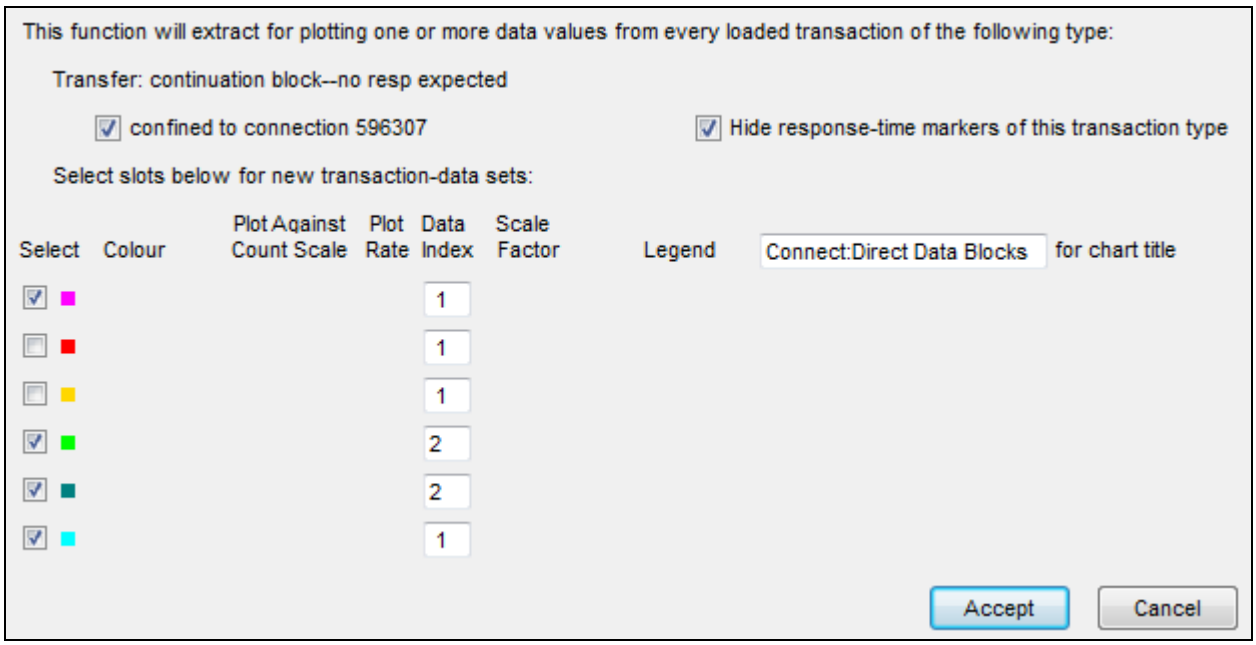
If NetData is set in Verbose mode for analysis it will record what it classes a *partial transaction* to characterise the transfer of every Connect:Direct data block. The block is regarded as the transaction's request message and its transfer time is the interval between successive blocks. These pseudo transactions don't have a server time or response message, but if test-compression has been enabled their data fields have two numbers: the length of the raw data in the block, and the length after compression by the gzip algorithm. NetData makes the latter measurement by compressing the data in every block. When those numbers are plotted as absolute values and as rates, with NetData's facility for plotting transaction data, they can provide a valuable insight to the causes of throughput problems.



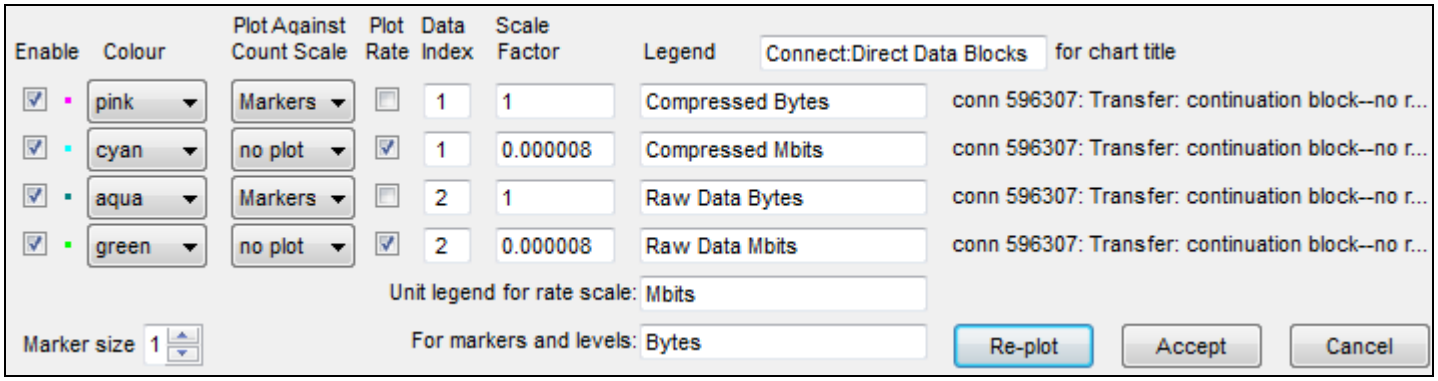
Check the Verbose box to record partial transactions, and check the next box to test-compress the contents of their data blocks. After analysis, for charting, load only partial transactions with the following settings in the load-data window:



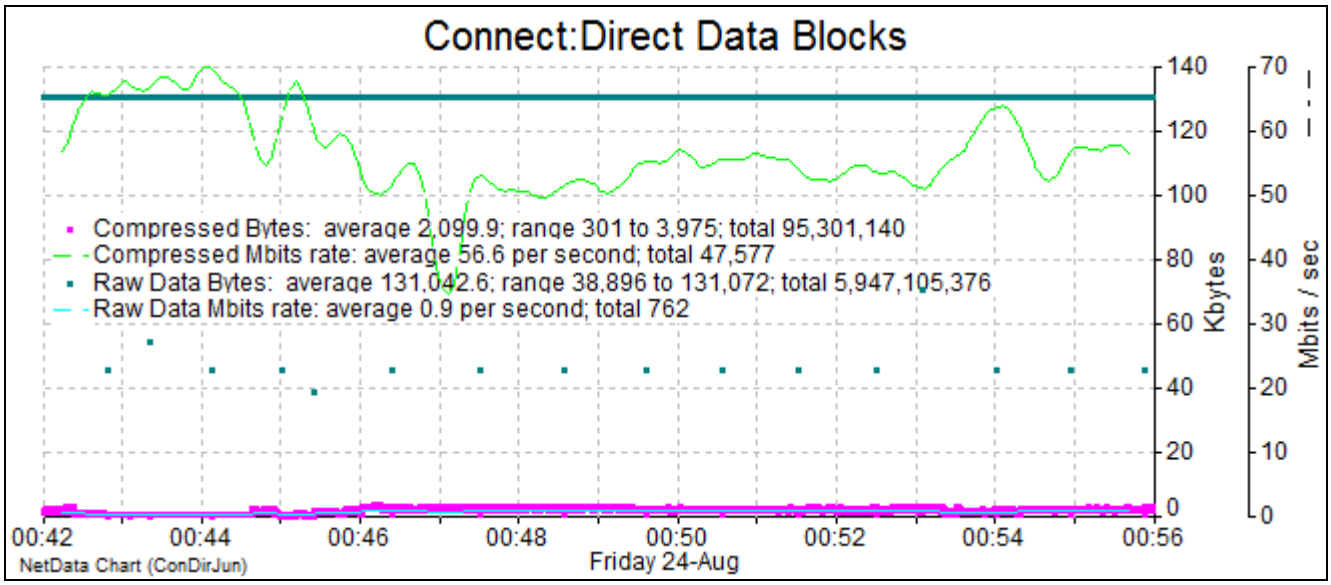
In the transaction table right-click the description of any partial transaction and select four colours to chart up to four data series formatted in the following successive windows:



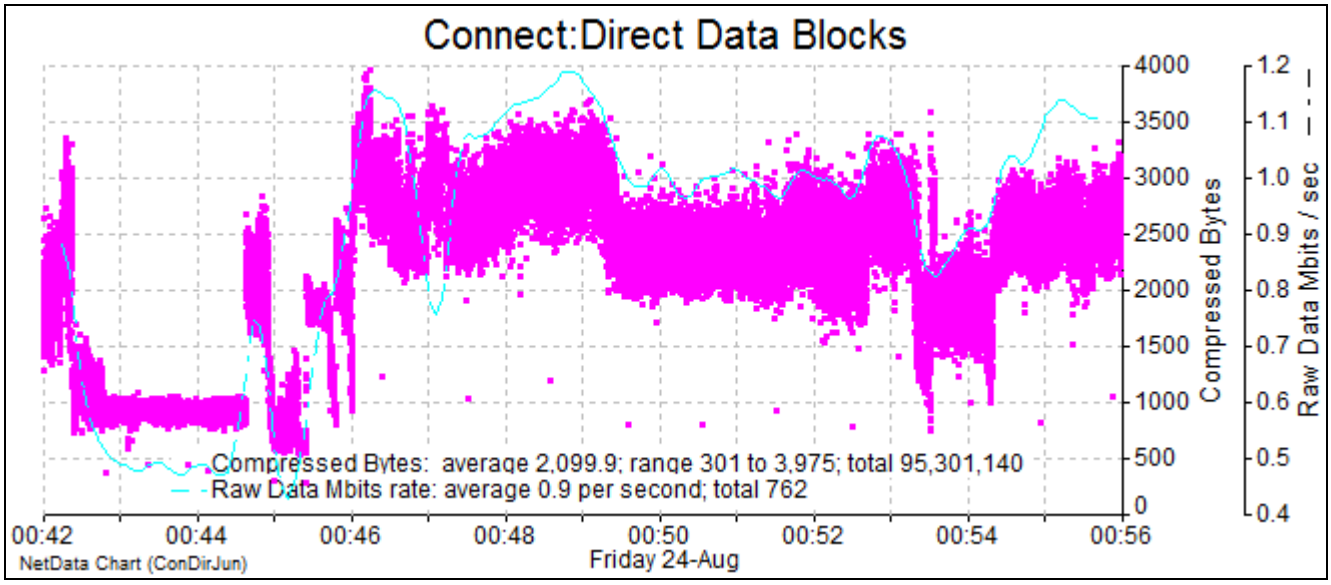
The first window displays the current occupancy of the six available slots for transaction-data sets. Initially they are all blank, but through this window new data sets may later be added to, or substituted for, existing sets.



The second window provides full control over transaction-data plotting, and can be recalled later by clicking the Edit button in the chart's format-control window. A data index of 2 refers to the second number in the data field, and a factor of 0.000008 converts bytes to Mbits. The resulting chart plots markers for the lengths of the raw and compressed data in each block, and the bandwidth required for the raw and compressed data:



The markers for the raw-data blocks of 131 KB form a line across the top of the chart, but the raw-data throughput ranges between 35 and 70 Mbps.



The charts reveal how raw-data throughput was reduced when the data was less compressible. Although compression reduced the bandwidth requirement by a factor of at least 35 and average of 60, the relationship between raw-data throughput and compressed size suggests that on this network throughput was often constrained by a bandwidth bottleneck.

23.20 TCP Port 4505

NetData now decodes TCP traffic using port 4505 and conveying message blocks that start with AABBh and end with AACCh. Until the application is better known, NetData labels the traffic as ‘AABBh’.

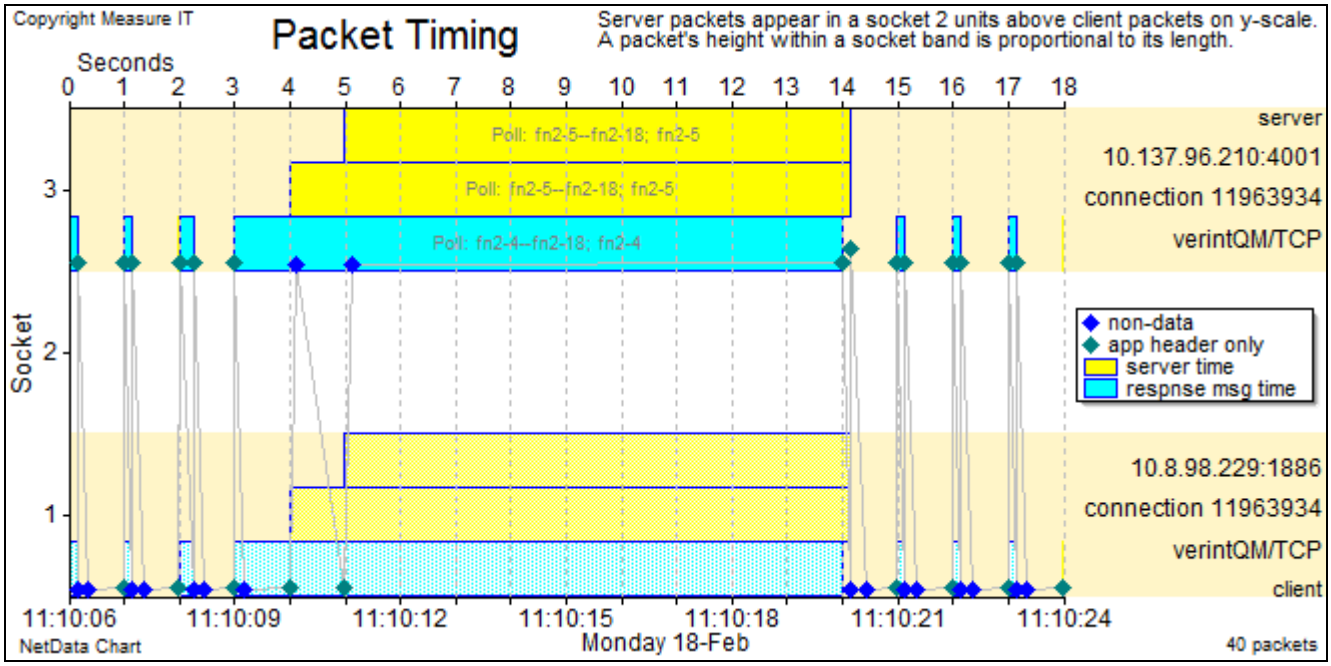
23.21 ZK Software Biometric Attendance Management

NetData can characterise the transactions of devices produced by ZK Technology and ZK Software for biometric measurements (finger-print recognition). This traffic usually uses UDP port 4370 in the devices.

23.22 Verint Quality Monitoring

NetData measures the response times of regular polls to the workstations of contact-centre staff made by a Verint i360 Quality Monitoring module that records video of their desktops.

Polls are normally sent to TCP port 4001 at one-second intervals, but if the workstation is severely stressed the client appears to stop issuing polls when three polls are outstanding, as indicated by this packet-timing chart:



23.23 Memcache

Memcache is an open-source, high-performance object caching system designed to reduce the load on database servers in dynamic web applications. The system is distributed in the sense that it can cache objects in the memory of multiple servers.

Communication with caching servers usually employs a Telnet style of text-based protocol with TCP port 11211. Some systems also offer a UDP version or a binary-coded version of the protocol. Different types of transactions often run concurrently.

NetData now decodes all the messages of the TCP text-based protocol and characterises all memcache transactions.

23.24 Lotus Notes

A connection with a Notes server first opens a session with a database, and all subsequent procedure calls (NRPCs) are often encrypted. An option on the Decoding page of controls allows NetData to record transaction response times from the encrypted traffic, assuming that changes in data-flow direction define request-response pairs (as for HTTPS). This assumption is only likely to be valid for simple Notes clients. With traffic in clear, the decoder extracts text strings from messages to display them in the Data columns of transaction and packet tables, and recognises some of the more common procedures.

NetData decodes Lotus Notes Traveler Autosync notifications between a device and server which uses TCP port 8642.

23.25 Decoding RADIUS with PEAP

The RADIUS (Remote Authentication Dial-In User Service) decoder decodes PEAP (Protected Extensible Authentication Protocol). It establishes a secure session with SSL (TLS 3.1) to encrypt authentication information, and RADIUS encapsulates SSL records in EAP-message attributes. A long SSL message such as a chain of digital certificates is segmented at two levels, first into separate UDP packets, and then into a sequence of RADIUS attributes that are limited in length to 253 bytes. NetData handles the necessary re-assembly to reconstruct SSL records.

Authentications using PEAP involve 9 or 10 round-trips, and NetData characterises user authentication transactions that span all the round-trips.

23.26 Autonomy Qfiniti Agent Monitoring Engine (AME)

This call-centre agent monitoring system records calls and provides agent statistics. The system uses MS RPC for various administration functions, using ports 9005, 9006, and 9025. It uses proprietary protocols for setting up recorder ports and sending audio data to the recorder. NetData characterises and measures all the request-response pairs. The recorded stream transfers data in only one direction, like FTP, and although there is no response-time to measure, NetData parses the stream and identifies data blocks and sub-blocks.

23.27 IPFX (ex Performance Solutions) CTI

IPFX provides a suite of application systems that support the operation of call centres, and use a proprietary protocol for interaction between call-centre agents and a central server. Some request messages from agents include SQL statements to query a database. NetData's new decoder recognises and characterises request-response pairs among large volumes of unsolicited messages that distribute common state information to all agents.

23.28 Artsoft

NetData decodes the traffic of an unknown application featuring the word artsoft and seen using port 3333.

23.29 IBM Content Manager OnDemand

NetData decodes traffic of IBM’s OnDemand content manager widely used to manage and present images of account statements, through TCP port 1445.

23.30 IBM IMS Connect

NetData decodes IBM IMS (Information Management System) Connect transactions. The decoder has been tested mainly on implicit-mode transactions with text coded in ASCII and using the user-exit module HWSSMPL1.

23.31 Dropbox LAN Sync Decoder

NetData detects and displays messages in the Dropbox LAN Sync Discovery protocol which uses UDP port 17500.

23.32 SDI Decoder

NetData characterises transactions in a protocol labelled ‘SDI’. Little more is known about this protocol, other than it makes heavy use of EBCDIC and has been seen using ports 3181 and 3291.

23.33 BMC BEM to IBM Tivoli Netcool/Omnibus

NetData decodes event notifications from BMC Event and Impact Manager (BEM) sent to a Tivoli Omnibus ObjectServer, and also decodes the traffic of Tivoli Netcool/Omnibus heartbeat probes. Probe traffic comprises messages sent to event sources, without responses and usually at regular intervals. Event messages sent to an ObjectServer prompt a response message.

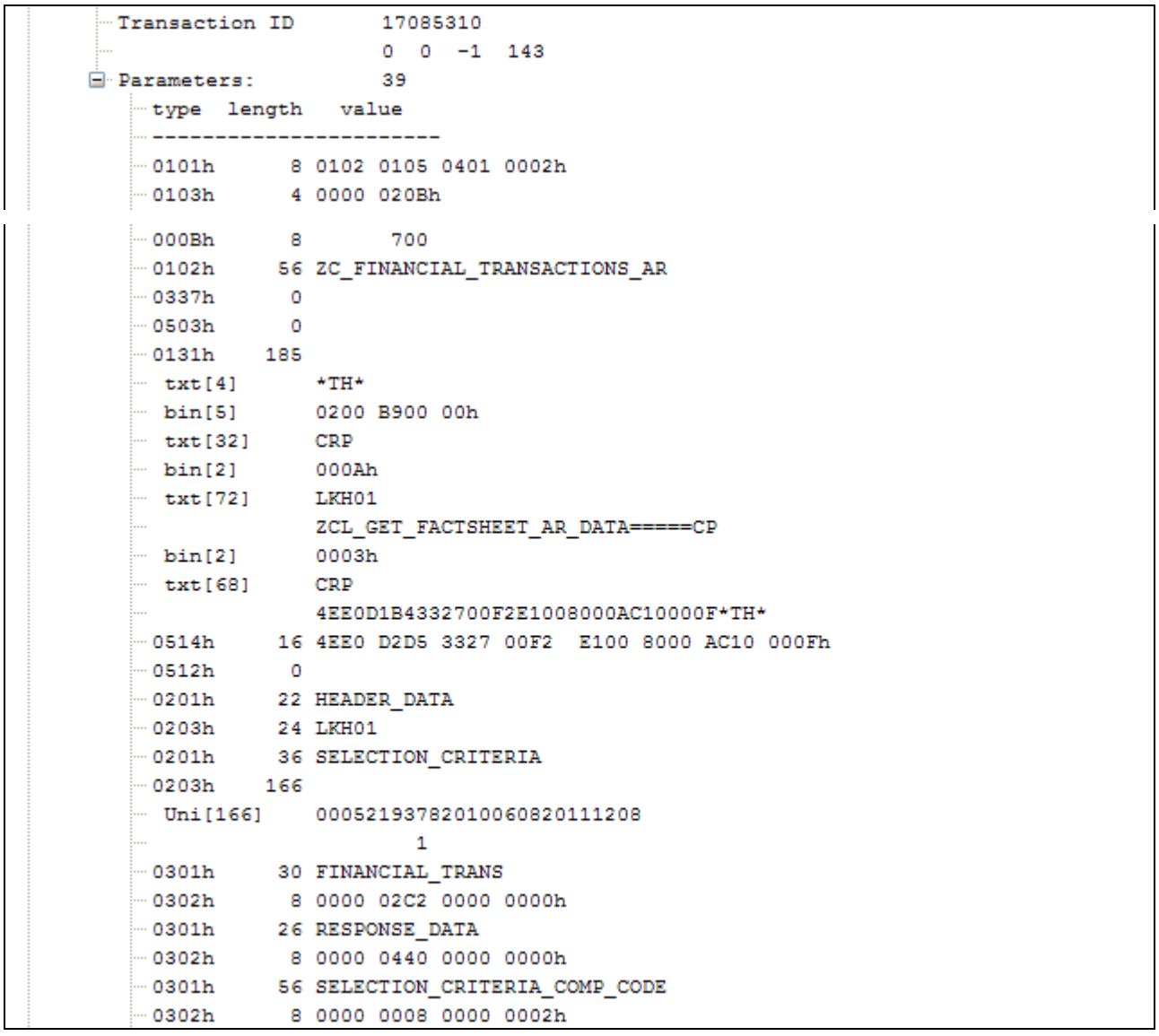
23.34 Unknown Protocols Using Ports 4892 and 12000

NetData detects the traffic of two unknown protocols that normally use ports 4892 and 12000. Port 4892 may be a logging service. Port 12000 multicasts event messages simultaneously to clients, and handles transactions with clients. NetData measures response times of both services.

23.35 SAP Gateway (RFC) Decoder

The decoder for Remote Function Call (RFC) transactions through SAP Gateways can display in tables the parameters of request and response messages. Parameters that name the function modules are added to the transaction request and response signatures to differentiate transaction types. Other parameters in the form of text strings are recorded with the transaction's key data, for display in the data column of the transaction table.

The gateway server can initiate transactions that NetData describes as 'Reverse Message Out'. The round-trip conveying the output message is always preceded by a preparatory round-trip described as 'Reverse Request Message Out'. Asynchronous RFCs involve the module ARFC_DEST_SHIP which transports data to a target system, and its message-out round-trip is usually associated with a concurrent round-trip from the client involving module ARFC_DEST_CONFIRM which confirms successful execution in the target system.

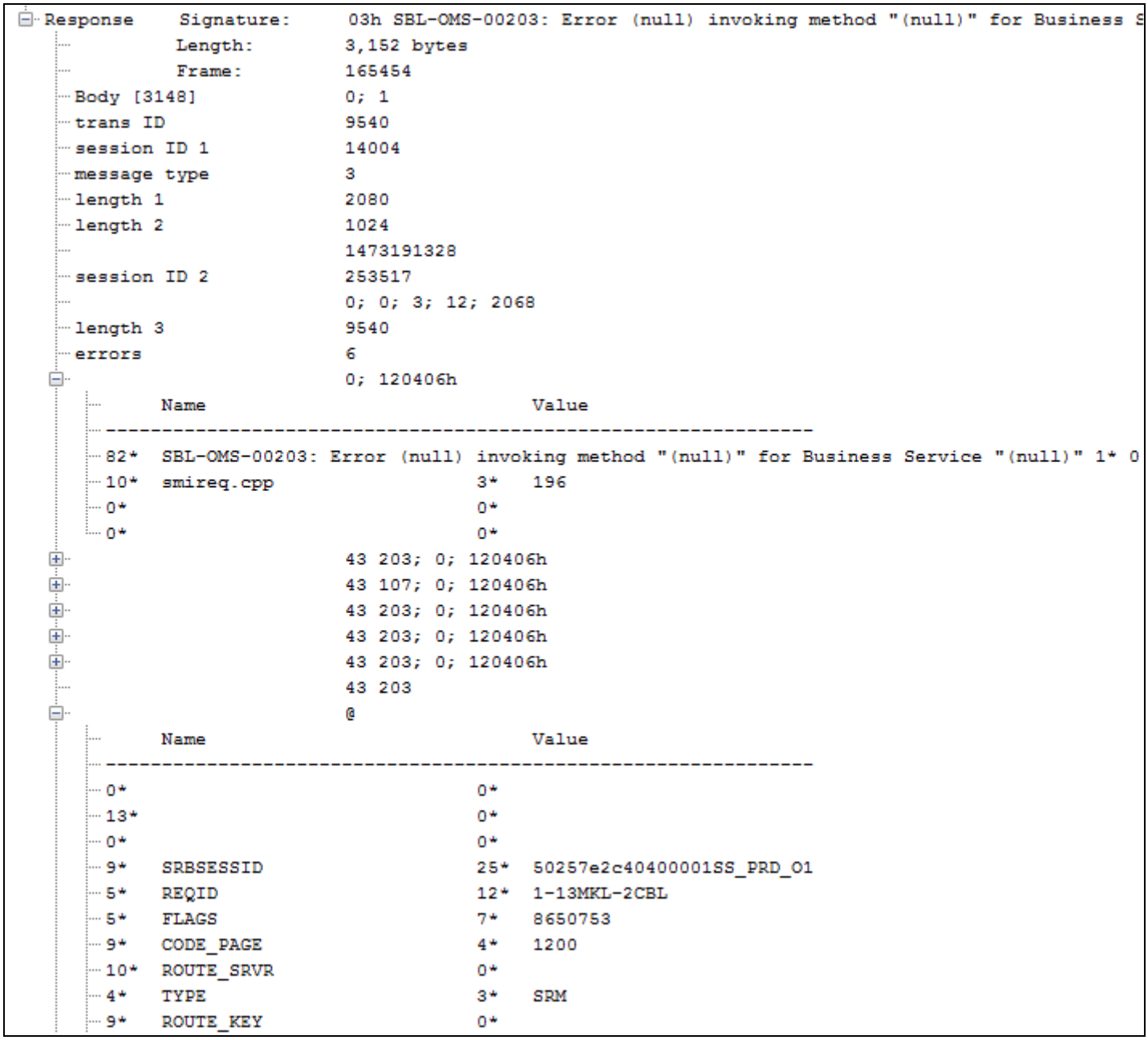


These two segments of a gateway-message parameter table illustrate the three columns for type, length and value. Values with binary codes are displayed in hex and end with 'h'. The values of the longer parameters comprise text and binary codes, and the text might be encoded with ASCII or Unicode. NetData separates the text from binary codes on different lines, prefixing them with a code indicator such as bin, txt or Uni and a length indicator in square brackets. Long strings of text are further folded into separate lines, without type or length indicators, at the ends of long strings of space codes.

23.36 Siebel Customer Relationship Management (CRM)

The contents of transactions between a web server with Siebel Web Server Extensions (SWSE) and a Siebel Server remain largely inscrutable but if the capture file contains the HTTP Post request that initiates a Siebel session then the significant name in that request's URL is included in the signature of all the subsequent transactions of that session.

The bodies of messages passed between Siebel servers contain mostly text (in Unicode) that convey tables of parameter name-value pairs which NetData displays as in the following extract from a transaction description.

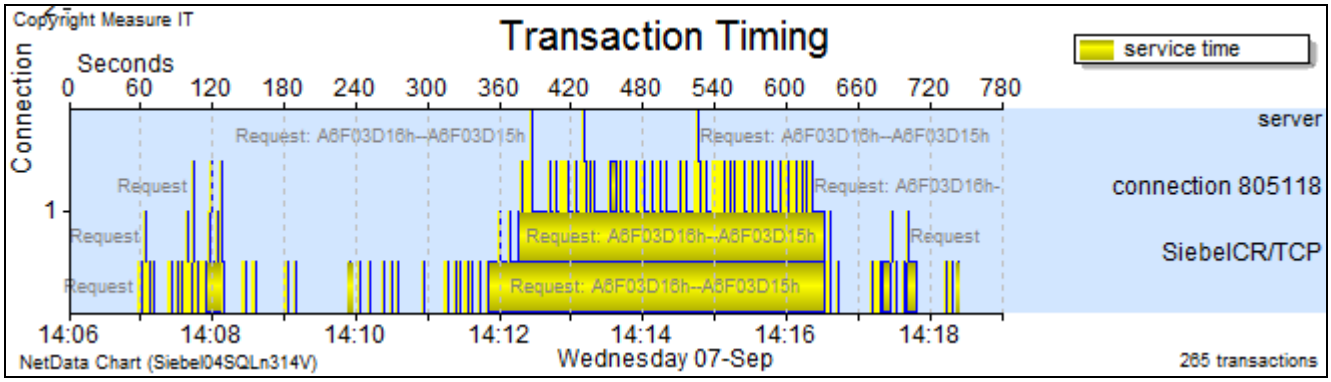


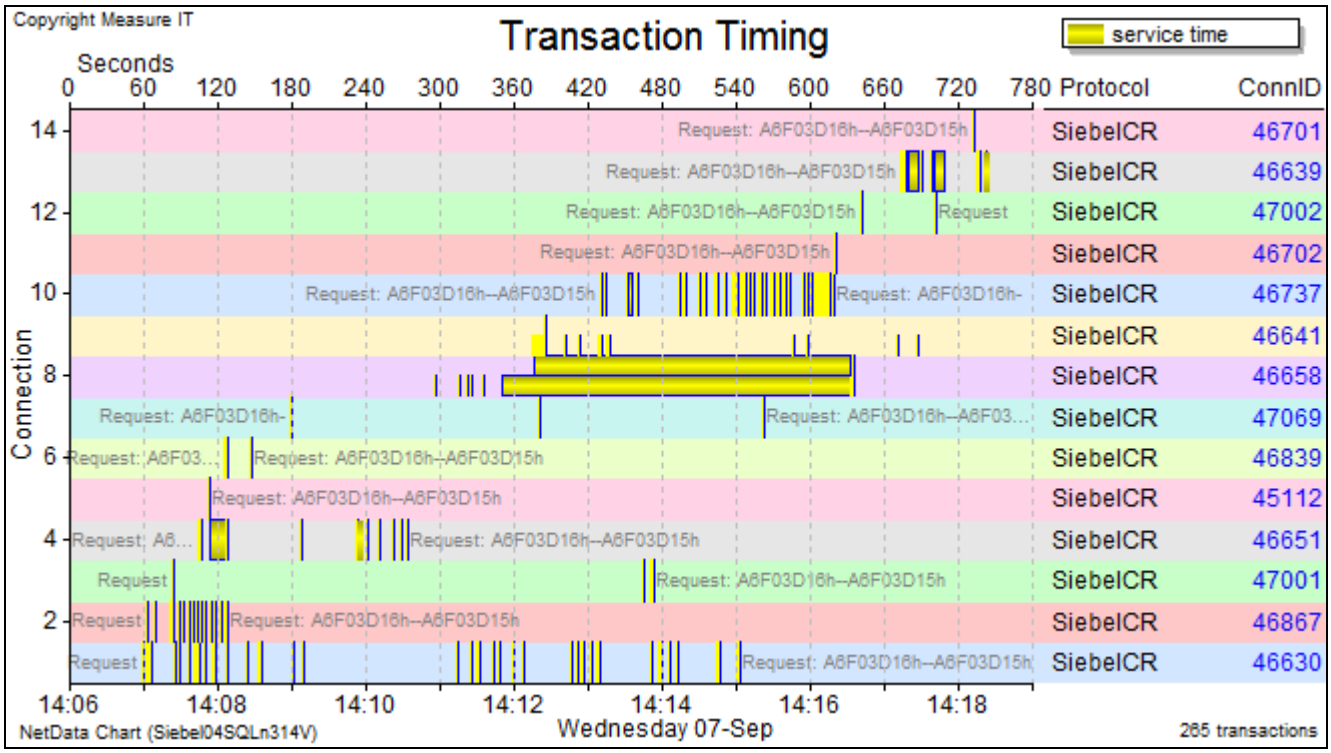
```
Response    Signature:      03h SBL-OMS-00203: Error (null) invoking method "(null)" for Business S
            Length:         3,152 bytes
            Frame:          165454
Body [3148]                 0; 1
trans ID                    9540
session ID 1                14004
message type                3
length 1                    2080
length 2                    1024
                            1473191328
session ID 2                253517
                            0; 0; 3; 12; 2068
length 3                    9540
errors                      6
                            0; 120406h
     Name                                Value
-----------------------------------------------------------------
82*  SBL-OMS-00203: Error (null) invoking method "(null)" for Business Service "(null)" 1* 0
10*  smireq.cpp                  3*   196
0*                               0*
0*                               0*
                            43 203; 0; 120406h
                            43 107; 0; 120406h
                            43 203; 0; 120406h
                            43 203; 0; 120406h
                            43 203; 0; 120406h
                            43 203
                            @
     Name                                Value
-----------------------------------------------------------------
0*                               0*
13*                              0*
0*                               0*
9*   SRBSESSID                   25*  50257e2c40400001SS_PRD_O1
5*   REQID                       12*  1-13MKL-2CBL
5*   FLAGS                       7*   8650753
9*   CODE_PAGE                   4*   1200
10*  ROUTE_SRVR                  0*
4*   TYPE                        3*   SRM
9*   ROUTE_KEY                   0*
```

Names and values are preceded by a length indicator that is terminated with an asterisk. Six of the parameters in the above response message contain error messages, and the first error message is appended to the transaction's response signature. NetData also includes in a transaction's signature any process, service or method name contained in a parameter table.

The Siebel protocol allows any number of transactions to run concurrently in a connection and NetData matches responses with requests by extracting a transaction ID from the header of every

message. The headers also contain numbers that identify a user, and NetData assigns secondary connection identifiers that allow the transactions of different users, and their packets, to be plotted on different bands on the timing chart. The resulting charts confirm that the Siebel server blocks concurrent transactions from the same user to ensure that they are processed strictly in the order of their arrival.





The above two charts of transactions in a single Siebel connection display the transaction bars of many concurrent transactions, and on the second chart they are separated into different bands for different users. One user (secondary connection 46658 on band 8) issued a second request while an earlier request was still outstanding, and the second request waited for the first request to end (after 281 seconds) before it could be processed. Thus these two transactions appeared to end at virtually the same time. NetData charts of database transactions at the same time revealed that the first request was held up by a single database query that eventually timed out.

23.37 Sapia Open Source Ubik

NetData decodes the traffic of Ubik, a framework developed by Sapia Open Source for handling and invoking Java distributed objects. Ubik uses BEEP as a multi-channel transport with Non-blocking IO (NIO). The payload of each BEEP message usually has two parts: a sequence of Java-like names interleaved with binary strings whose meaning is unknown to NetData; and a stream of tokens and data that form serialised Java objects. NetData displays the text and binary strings of the first part in consecutive rows, and displays Java objects, serialising tokens and object contents in a table with indenting to reflect class structures.

To form the signatures that characterise Ubik transactions NetData extracts the first four significant text strings in the first part of a message, and the first two class names of the serialised objects. Each name is shortened to the element that follows the last dot in the name.

The two panels below describe the two parts of a Ubik request message that form the transaction's request signature appearing at the top of the first panel:

First part:

```
Request      Signature:      MSG MksRmiContext/ InvokeCommand/ VmId/ OID/ Proxy
             Length:         1,249 bytes
             Frame:          3293
   MSG header                7 12 . 27398 1221
   Parameters:               0D0A 0Ah
           [43]              mks.ubik.transport.beep.AuthenticationToken
                             0000 0001 01h
           [7]               xxxxxxxx
                             0100 0000 1006 1FA9  738B BF0B D14A B23B  5E13 38C5 050Ah
           [21]              mks.rmi.MksRmiContext
                             0100 0101h
           [24]              FW1; -1; -1; -1; -1; -1; -1; -1
                             0Ah
           [50]              org.sapia.ubik.rmi.server.invocation.InvokeCommand
                             0000 0001 0Ah
           [30]              org.sapia.ubik.rmi.server.VmId
                             0000 0001 FFFF FEAC  E00E 4F5A 1F2F 6FC0  0000 0001 0Ah
           [29]              org.sapia.ubik.rmi.server.OID
                             0000 0001 0000 0153  5D7F 8754 57EE 4D54h
```

Second part:

```
0Bh Java objects [1221]
   token / name  length   name / value                                          handle serVrs
   ------------------------------------------------------------------------------------------
   y  Reset
   s} Proxy class    33 mks.frame.server.ExtensibleRemote
                     30 org.sapia.ubik.rmi.server.Stub
                     40 mks.frame.app.ui.IRemoteStatusListenerV4
   x  End block
   r  Super class    23 java.lang.reflect.Proxy                                     0 E127DA
      L  object       1 h                                                         1 t  Str
      xp End class block; Null super class                                        2 new cl
      ----- Derived object contents:
      sr h           37 org.sapia.ubik.rmi.server.RemoteRefEx                     3  1
         x  End class block
         r  Super class 35 org.sapia.ubik.rmi.server.RemoteRef                      4  1
            xp End class block; Null super class                                     5 n
         --------- object contents end
      --------- object contents end
```

23.38 Blocks Extensible Exchange Protocol (BEEP)

The Blocks Extensible Exchange Protocol (BEEP) is a framework for creating network application protocols that can use any transport protocol including TCP. BEEP provides a multiplexing function by splitting a single transport connection into any number of channels that serve as independent, full-duplex pipes. Application messages may use any format – textual or binary – but their syntax and semantics must accord with a pre-defined *profile* that is specified by a *protocol designer* and agreed by client and server when a channel is started. BEEP is documented in RFCs 3080 and 3081.

BEEP defines five types of messages – MSG, RPY, ERR, ANS and NUL – and each message is conveyed in one or more BEEP frames. The header of each frame includes a channel number, a message ID, a payload size indicator, and either a dot or asterisk to indicate whether the message is complete or there are more frames following.

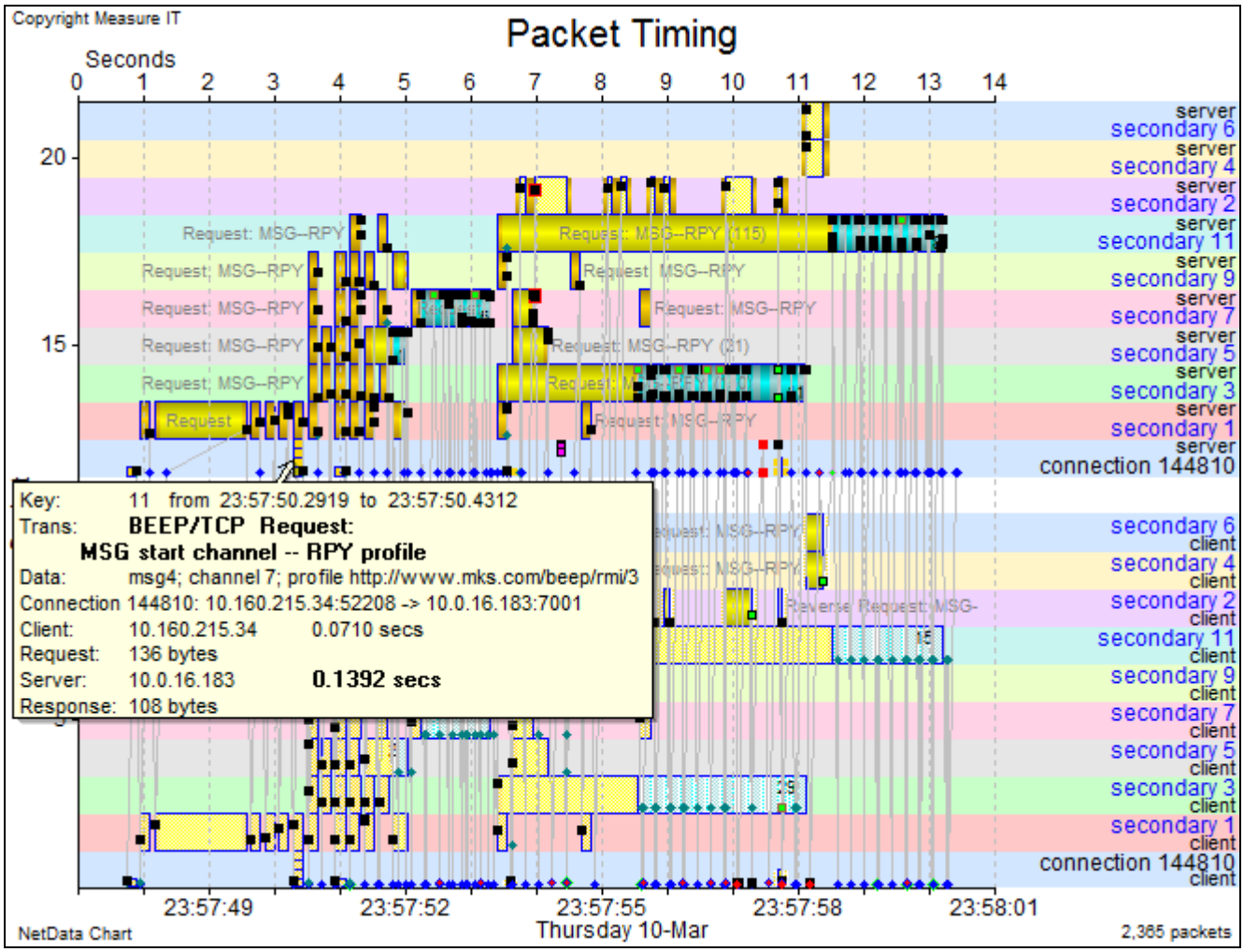
A MSG frame usually serves to convey an application request. A single positive or negative response is conveyed in either an RPY or ERR message, and multiple responses are conveyed in a sequence of ANS messages that is terminated by a NUL message.

Each channel has its own flow-control mechanism that is similar to that of TCP. Payload octets have individual sequence numbers and the header of each frame includes the sequence number of its first octet. Special SEQ frames serve to acknowledge octets and advise a window size. Sequence numbers and window sizes use modulo-32 arithmetic and, like all frame headers and management messages, are conveyed in the form of text.

NetData recognises BEEP frames carried by TCP; characterises application transactions; and fully decodes BEEP management transactions that use channel 0. The payloads of BEEP management messages accord with the MIME standard and conclude with an XML document that may consist of a single closed tag such as <ok />.

	Time Of Day	Seq	Source	Destination	2nd ID	AppType	Data	Function	Cont...	KeyData
■	23:57:53.557017	591	10.0.16.183: 7001	10.160.215.34:52...		BEEP	136	MSG start ch...	rev1	channel 2. seq710 [112]
■	23:57:53.563637	592	10.0.16.183: 7001	10.160.215.34:52...	-1	BEEP	1360			
■	23:57:53.563729	593	10.0.16.183: 7001	10.160.215.34:52...	-1	BEEP	1288	RPY	msg12	. seq7147 [2621]
■	23:57:53.564699	594	10.0.16.183: 7001	10.160.215.34:52...	-9	BEEP	1360			
■	23:57:53.564703	595	10.0.16.183: 7001	10.160.215.34:52...	-9	BEEP	511	RPY	msg6	. seq1094 [1845]
◆	23:57:53.613662	596	10.160.215.34:5...	10.0.16.183: 7001		BEEP				
◆	23:57:53.626642	597	10.160.215.34:5...	10.0.16.183: 7001		BEEP	17	SEQ		ack822 wdw65536
■	23:57:53.626658	598	10.160.215.34:5...	10.0.16.183: 7001		BEEP	108	RPY profile	rev1	profile http://www.mks...
◆	23:57:53.633658	599	10.160.215.34:5...	10.0.16.183: 7001		BEEP				
◆	23:57:53.633682	600	10.160.215.34:5...	10.0.16.183: 7001	-1	BEEP	18	SEQ		ack9768 wdw65536
◆	23:57:53.634764	601	10.160.215.34:5...	10.0.16.183: 7001		BEEP				
■	23:57:53.636626	602	10.160.215.34:5...	10.0.16.183: 7001	-5	BEEP	393	MSG	msg6	. seq2570 [368]
■	23:57:53.636859	603	10.160.215.34:5...	10.0.16.183: 7001	-7	BEEP	393	MSG	msg6	. seq5612 [368]

In the packet table the Context column displays message IDs, prefixed with ‘rev’ if the transaction was initiated by the server. The KeyData column displays sequence numbers, payload sizes in brackets, ack numbers, window sizes and the significant contents of management messages. NetData regards channel numbers as secondary connection IDs to the effect that a timing chart, with or without packet markers, can split the activity of a single connection into separate bands for each channel, as in the chart below. In the packet table channel numbers appear as negative secondary connection IDs. Packets 591 and 598 in channel 0 formed a reverse channel-start transaction that started channel 2.



The primary connection band, with ID 144810 in this case, carries the transactions and packets of channel 0, and markers for packets such as TCP acks and retransmissions that aren't associated with a particular channel. The connection IDs of the secondary connections are BEEP channel numbers. Because the client and server may start a channel at the same time, they avoid clashes by restricting client channels to odd numbers and server channels to even numbers. In the chart the top three bands have even channel numbers and all their transactions are described as *Reverse* requests to signify that they were initiated by the server.

The cream pop-up describes one of four concurrent BEEP transactions from the client, to start channels 3, 5, 7 and 9. The response in an RPY message specifies – in the form of either a URI or a URN – the profile that is to govern channel exchanges. This profile was selected from a list of acceptable profiles in the MSG frame.

23.39 Android Debug Bridge and Runtime Logging

NetData decodes Android Debug Bridge (ADB) USB transactions when conveyed over TCP, and the logging messages of Android Runtime (ART) or the older Android system called Dalvik runtime, two protocols often used when developing Android apps.

23.40 Protocol for Operations Discovery (pfod)

NetData decodes the Protocol for Operations Discovery (pfod) created by Forward Computing and Control for communication between system- or equipment-control devices and apps running in Android mobile devices.

23.41 Bloomberg Professional Services

NetData associates with Bloomberg Professional services any UDP traffic to ports in the range 48129 to 48137, and TCP traffic to ports in the three ranges 8194 to 8198, 8209 to 8220, and 8290 to 8294. NetData recognises two Bloomberg TCP protocols, one using SSL (tagged *BlmBrgS*) that handles data in both directions, and another (tagged *BlmBergD*) that seems to handle data only from a Bloomberg server to a client, and is neither compressed nor encrypted.