Multi-Protocol Visualization

A Tool Demonstration

James Hall, Andrew Moore, Ian Pratt and Ian Leslie
University of Cambridge Computer Laboratory
JJ Thomson Avenue, Cambridge CB3 0FD
United Kingdom
firstname.lastname@cl.cam.ac.uk

ABSTRACT

This paper describes a system for the visualization of multiple protocols. The visualizer makes possible the identification of both intra and inter-protocol behaviour. This tool has become a critical resource in the development of our multi-protocol monitoring system; allowing the verification of the monitoring system, identification of new modes of behaviour and the easy visualization of potentially overwhelming quantities of information[1].

Keywords

Network monitoring, network event visualization, multi -protocol analysis, multi-protocol visualization, TCP, HTTP.

1. INTRODUCTION

This paper describes the visualization tool constructed as part of Nprobe [6]. It has long been considered that visualization provides a powerful aid to understanding and reasoning, e.g. [11]. Additionally, Nprobe — performing full line-rate capture — allows a unique perspective on the behavior of network protocols and specifically the interaction between different protocols in the network stack. The potential to incorporate a visualization tool within Nprobe was a clear and important choice.

Nprobe encompasses monitor, analysis and visualization tools, and manipulates network data derived from full line-rate capture. For example, an HTML-based browser relies upon the TCP/IP layers to transport its data. Thus the behavior of the transport layer directly impacts upon the behavior (and performance) of the application and Nprobe provides access to the data of multiple protocol layers.

To more-fully understand the behavior of a single layer and the interaction between layers we constructed a set of visualization tools. The bulk and scope of the data collected by the Nprobe monitoring system calls for tools which render the data and relationships that it contains in a compact and readily comprehensible form. The visualization tool described in this paper illustrates, not only the power of visualization, but also the depth and breadth of information provided by the Nprobe system.

Visualization systems for network traffic are not new: such a wide area has produced a wide range of tools. The Walrus tool, part of CAIDA's Coral Reef [5] was developed specifically to assist in the visualization of Internet topology. The MRTG system [7] is commonly used to graph daily traffic utilization trends, and the LBNL NetLogger NLV tool [10] presents data gathered across protocols from a variety of sources. More detailed, protocol-specific, visualization tools are less common, Tcptrace [8] is one example. Aside from these publicly available offerings, there are a number of commercial systems such as the Agilent N4212A Gigabit Ethernet Protocol Analysis Tool, although such systems are rare in research institutions due to their prohibitive cost. It is among this group of tools that we present the Nprobe visualization system.

We differentiate the Nprobe system due to its ability to allow visualization of multiple protocols from the network stack, simultaneously. Our approach is only made possible through the information provided by the monitoring tools built as part of Nprobe. The combination of on-line and off-line processing means that the relationship between packet, flow and application can be uniquely illustrated by our tool.

The Nprobe visualization tools are innovative in their integration with the post-collection analysis framework. Hence raw data contained in trace files (e.g. TCP segment sizes, flags, and timings) can be illustrated, but, more significantly, the *results* generated by analysis can be displayed in conjunction with the underlying data and their relationship made clear. The TCP visualization reproduces much of the functionality of Tcptrace, but additionally shows the relationship between application-level and TCP activity, and presents the results of the TCP modeling process described in [3]. The Web browser activity visualization tool presents both events and timings over the relevant range of protocols, together with the results of analysis which associates HTTP transactions and TCP connections to reconstruct the downloads of whole pages, and draws inferences about the browser's behaviour and use of connections.

Because Nprobe gathers data from the entire range of protocols of interest, and this data may be used to investigate the interactions between those protocols, the bulk and range of the data contributing to any specific analysis task is likely to be extensive, and the relationships to be identified may be both obscure and complex. Visualizations are normally invoked from analysis code in order to

[1]The density of information presented by the visualization tool that we describe dictates a reliance upon presentation in color, and the included figures are, therefore, less clear when printed in grey-scale. A full color version of the paper is available at `http://www.cl.cam.ac.uk/netos/nprobe/publications/MomeTools2003.html`

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM SIGCOMM 2003 Workshops August 25&27, 2003, Karlsruhe, Germany
Copyright 2003 ACM 1-58113-748-6/03/0008 ...$5.00.

present raw data, to identify relationships, and to examine and validate the results of analysis. The exact nature of the information shown will depend upon the protocols of interest, the granularity and type of data gathered, and the analysis task being conducted or developed.

Section 2 provides a brief overview of the Nprobe monitoring architecture. This section also describes the mechanisms that provide data to the visualization tool. Section 3 describes the current implementation of our tool. Within the section we strive to provide a broad overview of some of the abilities this tool can afford. Section 4 illustrates the tool in use and describes several further uses to which the tool has been put. Finally, alongside a summary of our work, Section 5 notes the ease with which the visualization tool may be extended in the future.

2. CONTEXT

Network monitoring has increasingly been considered a staple tool in the understanding of computer networks. From an improved understanding of the larger picture of routing and connectivity [9], to allowing the examination and construction of traffic models [12], to an understanding of the interaction between application and network [4], network monitoring has had an important role to play.

In a previous paper [6] an architecture for full-packet capture was described. The capture of data based upon access to the full packet can provide both a bonanza and a drawback. As part of the capture process tens of thousands of simultaneous TCP flows are reassembled. By recording information about how the TCP flows were represented in the network and only a targeted, application-specific, summarization of the data they transported, a monitoring architecture can achieve significant compression. It is this on-line summary of network traffic that allows compression of many hundreds of Mbps of data onto disk, without packet loss.

The online system, analyzes data only sufficiently to allow the correct interpretation of content during data extraction or abstraction, allowing aggregation of data into blocks representing one or more related TCP flows. The transformation of data to information requires the retrieval of data from the storage format, and in most cases further post-collection association and analysis is also required.

An object-oriented approach is made to the handling of the data files. Thus the methods for accessing the data may easily incorporate functions to perform any necessary post-processing. This approach has meant that, prior to any visualization, any processing needed following collection is automatically performed as part of the retrieval process. This approach is only made possible through the rich depth of (partly processed) data provided by the monitoring system.

As an example of how collected data is ultimately presented to the visualization system, consider the display of the most simple HTML/HTTP transaction using HTTP/1.0 without any level of persistent connection. One HTTP transaction (a request and reply) is represented within the trace-data by an associated state record. This state-record in turn indicates the location of specific data entries that record the request and reply contents. Along with the HTTP state record, the record of request and reply indicates the relevant TCP packets that carried (and acknowledged) the data. Additionally the events of the set-up and tear-down of the relevant TCP flows are also available. Thus, by accessing the collected data, a visualization tool need only locate a particular HTML/HTTP transaction record to be provided with the relevant data flows, the network packets that made up the data and also set-up and tore-down the relevant flow. In addition to the data, the monitor provides a time-stamped record of the packet events allowing an interpretation of the behavior over time.

The data structures described for this simple example may be extended to represent more complex protocol implementations such as extended versions of HTML/HTTP as well as other application protocols.

Among the Nprobe analysis tools, the visualization tool is the most striking, both in its ability to bring obvious clarity to difficult protocol-relationships and in presenting a wide range and depth of data. However, the visualization tool is not the only analysis system and exists alongside test-based tools for summary and statistical interpretation. However, each part of the toolkit: visualization, summary and statistical analysis, is built using the same methods of data access. The exposed nature of this access interface allows the construction of any number of purpose-specific analysis. It is this aspect that reinforces the utility of the Nprobe architecture.

In addition to running in the preferred, on-line mode, the Nprobe tools are able to run off-line; converting a tcpdump trace-file into the format used by analysis tools such as the visualizer. This is not the preferred mode of operation as it subjects the collected data to the limitations of tcpdump[2].

3. IMPLEMENTATION

The implementation consists of three components. The first of these is a data plotting mechanism that provides interactive access to the underlying data. The second system allows the visualization of TCP/IP packet trains, while the third tool incorporates the visualization of TCP/IP packet trains in concert with HTML/HTTP, (application-layer), operations.

While Nprobe on-line monitoring code is written in C and assembler. The off-line data retrieval process is implemented using object-oriented Python, thereby allowing the easy association of off-line processing with data retrieval.

3.1 The Data Plotter

The data plotter may be used as a stand-alone tool but is normally invoked in order to display selected data sets from analysis results. The tool is designed to make the manipulation, examination and comparison of data sets possible without reloading the set, provides interactive support for raising, lowering or blanking out individual sets and has a zoom facility. Plotting styles can be varied as desired, and the data re-plotted as scatter plots, histograms, probability and cumulative density functions as appropriate to its type. Basic smoothing methods are also available.

The plotter acts as the principal data management tool allowing selected data sets (or sub-sets), or their derivatives (e.g. density functions) to be saved or printed. The most significant difference between the Nprobe plotter and other plotting programs is its facility allowing the user to examine the derivation of data points. A callback and tag mechanism allows the user to select a set of data points and to examine the underlying raw data and the process of its (re-)analysis in close detail.

3.2 TCP Connection and Browser Activity Visualization

The raw trace file data contributing to each datum generated during analysis are likely to be complex and bulky and to have complex relationships. Consider the data associated with each packet

[2]We describe tcpdump as 'limited' in the context of multi-protocol data collection because the high overheads implicit in multiple user/kernel space copies and the high volume of data generated by the verbatim capture of packets using a long 'snaplen' do not readily support the monitoring of high-bandwidth networks without packet loss or unfeasibly large trace files.

of a TCP connection, the number of packets that may be sent on a single connection, the complications due to packet loss and the number of connections that may be involved in a Web page download. The raw data, furthermore will be sparsely distributed among the other file contents. Although a trace file reader exists that is able to present trace records in a convenient form, while providing rudimentary facilities for selecting associated data, it would be time consuming and (in the case of large data associations) virtually impossible to fully assimilate and comprehend all items without assistance. The appropriate visualization tools will present data in a compact and comprehensible way which will assist in identifying the relationships that it represents.

Trace file data analysis is concerned with the distillation of information from the raw data. Visualization should, therefore, not only present the raw data but also, insofar as is possible the information generated by analysis and the relationships upon which it is based. The visualization tool is designed to meet this requirement.

3.2.1 The TCP Visualization Tool

The TCP visualization tool, illustrated in Figure 1, plots sequence numbers against time in a style similar to `Tcptrace` [8]. Data segments are shown as vertical arrows scaled to the data length carried, acknowledgments as points, the 'ACK high water' drawn and segments are annotated with any flags set. Other salient data (e.g. window advertisements) may also be shown.

The tool, however, has considerably greater functionality: Section 4 illustrates this by describing a technique for examining the dynamics of TCP connections, relating them to activity at higher levels of the stack and identifying application behavior and connection characteristics. Where connections have been subject to this technique its inferences are also displayed: the relationship between TCP and application activity, causal relationships between packets and congestion windows are, for instance shown; secondary plots of the number of packets 'in flight', inferred network round trip times and application delays are also presented. This data is derived using modeling techniques described by Hall *et al.* [3].

Figure 1 shows the tool's primary window displaying the activity of a non-persistent TCP/HTTP connection. Although the figure contains a great deal of interesting detail we will comment on a few features to illustrate the power of the visualization:

A The solid purple line to the left of the segment 'arrows' represents the server's congestion window as predicted by the connection model — the number of segments in each subsequent flight increasing as the window opens.

B Packet #13 is retransmitted (packet #18 — shown as a *red* arrow) at approximate time 540 ms.

C The server's congestion window shrinks following the retransmission, resulting in a flight of only two segments immediately afterward. The retransmission also causes the connection to enter congestion avoidance — shown by the broken congestion window line.

D Horizontal dashed lines indicate causal relationships between packets. Here packet #25 from the server triggers an acknowledgment — packet #26 — from the client

E The last line of the legend at the top of the main pane indicates that the modeling process has explained the behavior the server's TCP implementations based upon a generalized base model of behavior with an Initial Window (IW) of two segments; the client's behavior is explained by a similar model, but the IW is unknown as less than one MSS of data has been sent. The term `SSTGT=1` denotes that the implementations enter the congestion avoidance phase when the congestion window *exceeds*[3] the slow start threshold.

To ascertain the patterns of TCP activity involved in even the relatively short and uncomplicated connection visualized in Figure 1 would require the close examination of 38 packet headers: A time consuming task which might, even then, fail to identify all of the features present. To fully comprehend the activity of a substantial connection, or one with complex features, rapidly becomes a daunting and error-prone task. Such difficulties are, however, minor in comparison with those presented by the need to relate the outcome of the connection modeling process to the packet level data contained in the trace — an essential step to ensure the accuracy of the model constructed. The figure illustrates that visualization offers a succinct and comprehensive presentation of both the original data and the information synthesized from it, in which features are readily identified and relationships made explicit.

The tool opens a secondary widow (not shown in Figure 1) which displays a textual representation of the trace file data associated with the connection and which may be toggled between packet and application level data. Displayed items can be selected and recursively expanded to show greater detail.

3.2.2 The Web Browser Activity Visualization Tool

The Web browser activity visualization tool presents the data extracted from the TCP, HTTP, and HTML levels of the protocol stack for all HTTP activity originating from single hosts or client and server pairs. Figure 2 illustrates part of a small page download.

The tool's main pane **[1]**[4] displays the TCP connections carrying HTTP transactions plotted against time as scaled horizontal bars with tics showing packet arrival times (at the probe) and annotated (in blue) with details of the connection. Client activity is shown above the bar, and server activity below it. HTTP activity is shown as requests and responses above and below the TCP connection line respectively. Request or response bodies are shown as blocks of color representing the object type, and are annotated (in black) with the transaction's principal characteristics (e.g. the object's `URL`, the request type and the server response code).

A secondary pane **[2]** may be toggled between a display showing a key to the symbols used in the main pane and a summary of the activity shown, or details of individual selected TCP connections in the style of the TCP visualization tool's secondary window. The TCP visualization tool may be invoked by dragging over a connection bar in order to closely examine individual selected connections. A further secondary map pane **[3]** shows an overall view of the entire set of browser activity at reduced scale (the level of detail in the [scrollable] main pane will generate a graph larger than the tool's window for large pages or extended browsing sessions and the secondary pane serves to locate the main pane contents).

It is possible to reconstruct the trace data; all activity associated with the downloading of entire Web pages; the tool presents the results of this reconstruction by showing the dependency relationships between objects as dashed lines. In-line links (e.g. those to contained images or frames or representing redirection or automatic updates) are differentiated from 'followed' links (i.e. those followed by the user). The objects downloaded as constituents of discrete pages are thus grouped together and the user's progress

[3] This is the default for the base TCP model — TCP implementations may optionally enter congestion avoidance when the congestion window *reaches* the threshold [1].

[4] This paper is in color, grey square brackets refer to numbers on the indicated figure.

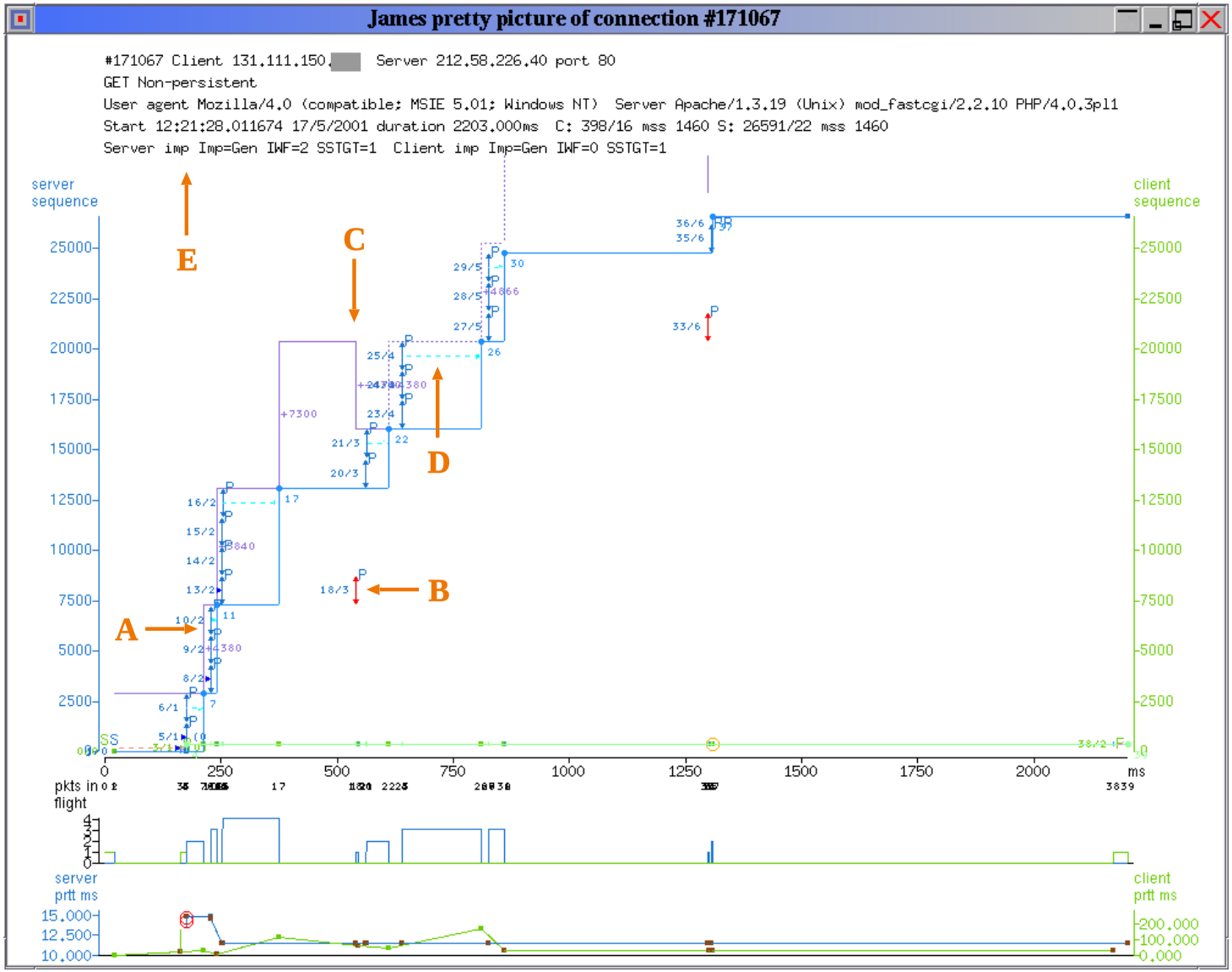


Figure 1: The TCP visualization tool primary window. For the sake of clarity a short connection carrying a single HTTP transaction is shown. Where more complex application-level activity is involved the plot would be annotated to show the connection between this and TCP-level activity (e.g. in the case of a persistent HTTP connection segments would be labeled to identify which carried the various requests and responses, and the pRTT (partial Round Trip Time) plot would show the calculated browser and server latencies.

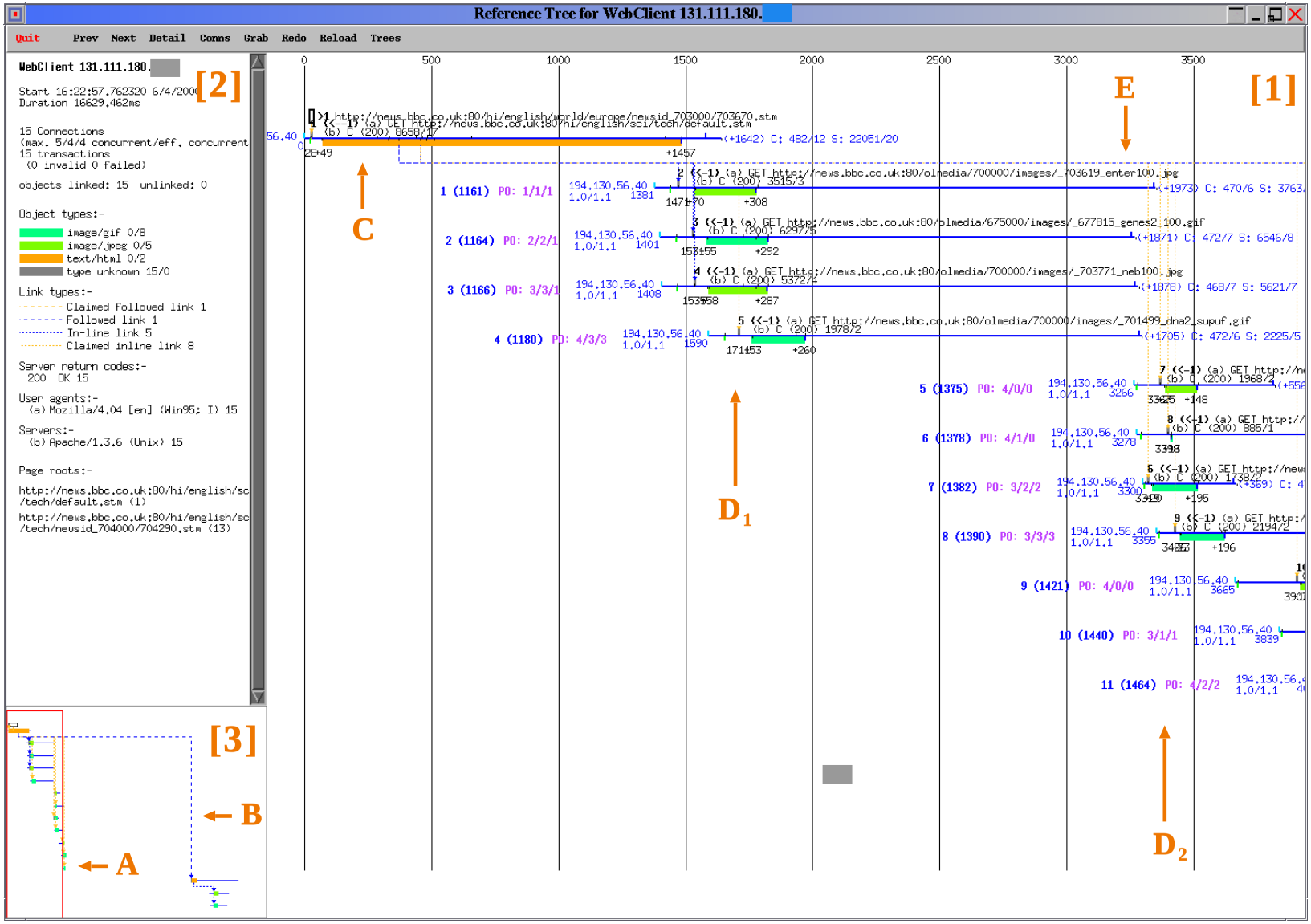


Figure 2: The Web browser activity visualization tool

from page to page identified. The way in which the browser uses TCP connections and the inferred relationships between connections and objects are illustrated in clear detail.

Figure 2 is annotated to demonstrate a sample of the key features of the example visualization:

A The red box in the map pane **[3]** identifies the section of the entire reference tree shown in the main pane **[1]** which can be scrolled by either dragging the box or the main pane itself.

B The map pane shows that the user goes on to visit another page following that currently occupying the main pane.

C The page's root HTML document identifies its referrer, not seen by the trace, which is represented by the small black box immediately above it.

$D_{1,2}$ The browser is configured to download subsidiary objects using four concurrently open TCP connections.

D_1 Connections #1 – #4 are used to download the first four in-line images in the page. Each request occupied a single packet shown as a large tic above the connection bar (colored grey for 'type unknown' as there is no object associated with the GET requests); dotted lines above the request tics connect them to the referring object (**C**) at the packet carrying the portion of the parent document containing the links. Although the links were seen in a packet at approximate time 450 ms the browser did not open connections upon which to request the objects until approximate time 1400 ms — hence introducing a delay of nearly a second into the page download.

E Although delivery of the first four image objects was completed by approximate time 1950 ms the browser does not *close* the relevant connections until approximate time 3300 ms, hence inhibiting the opening of the subsequent set of four connections, and introducing a further overall delay of about 1.3 seconds.

To manually identify and associate the activity involved in downloading the first page shown in Figure 2 would require the examination of 12 trace file records containing details of an equal number of transactions, and of 159 TCP packets — a complex and tedious task. To identify the links contained in the parent document would additionally involve the scanning of 8,658 bytes of HTML spread over 17 packets. If traditional `tcpdump` style traces had been collected the exercise would involve the manual examination of all 159 packets, their TCP and HTTP headers. The page illustrated is atypically small — pages with container documents measured in tens of kilobytes containing tens or hundreds of in-lined images are not unusual. It would be infeasible to manually examine the data describing activity in such cases, and to correctly interpret its internal relationships and meaning. The utility of the tool is amply demonstrated in the figure which presents all of the relevant data and the inferences drawn during analysis in an immediately accessible form. The overall pattern of activity is plainly discernible and features of interest clearly identifiable.

3.3 Software Organization and Extensibility

Although work with Nprobe to date has concentrated largely upon the study of Web traffic, the system is designed to be extensible to gather and analyse data from other protocols, and to allow tailoring of data collection within protocols to the needs of specific research projects. The contents of the Nprobe trace files are therefore variable in scope and are recorded in a self-defining format. Data is retrieved from trace files using a standard retrieval interface consisting of Python classes automatically generated from the C header files defining trace formats using the Simplified Wrapper and Interface Generator [2].

Data is analyzed using a set of reusable protocol-centric and cross-protocol analysis classes, and the analysis repertoire enlarged as required by the sub-typing of existing, or addition of new, classes. Analysis is controlled by Python scripts, optionally using a *StatsCollector* class or appropriate sub-classes. Results and an analysis log are summarized and presented by a further graphical tool, invoked from the script or StatsCollector, which allows a recursive descent through increasing levels of detail.

The data plotter is invoked from the summary tool in order to examine, manipulate, or save the data sets generated. The visualizations can be invoked from the summary tool for individual or multiple instances of the relevant raw and derived data (e.g. TCP connections and the results of their modeling, Web page downloads or activity at a page, browser, or server granularity). Hence grouped activity (e.g. all TCP connections to a particular host, all page downloads from servers ranked by use) can be examined, and individual instances (e.g. those recorded in the analysis log as exhibiting conditions of particular interest) are also immediately available.

Visualizations can also be invoked from the data plotter by selecting individual or multiple data points, hence allowing the derivation of those points to be examined, together with the analysis audit trail contributing to them. Such a facility is particularly useful when considering outliers — why are particular values highly dispersed, and do they represent activity of particular interest or analysis failures. Whenever visualizations are invoked the associated analysis of contributing data is re-run in verbose mode to allow close examination of the process.

The data plotter can take input as simple as a set of dependent variable values, but when used in conjunction with Nprobe analysis will normally be invoked with one or more parameterized data sets; meta parameters will identify callback methods to invoke the appropriate visualization class, and point parameters will identify the contributing raw data. The information presented by the visualization tools can be categorized as:

- Representing raw trace file data: e.g. TCP header information
- Generated by standard analytic classes: e.g. current TCP window sizes
- Generated by specific analysis tasks: e.g. Web server latencies

Because it is undesirable to clutter visualizations with unwanted information, and because the employment of analysis classes is task dependent, we have taken the approach that generalization, flexibility, and extensibility are not served by the provision of monolithic visualization classes. Instead basic screen manager classes are provided which create and manage the canvases used by the visualizations, calculate and draw scales, handle mouse events, etc., but the drawing of the required information is carried out by methods of the analysis classes generating the information. Raw TCP segment information, for example, is drawn by a method of the base TCP analysis class which is invoked with a canvas, origin, and scaling information as arguments, and the information generated by the TCP modeling analysis class is drawn by a dedicated method of that class. In this way the visualization tools expect no specific data format, and information-generating classes are not constrained in the type or range of information which is visualized.

As existing analysis classes are sub-typed to provide alternative or additional functionality their drawing methods can be similarly sub-typed to present the information produced. When new analysis classes are added to the existing repertoire (as, for instance, the Nprobe monitor is extended to capture data from further protocols), the functionality of the visualization tools is similarly extended by the provision of new drawing methods.

The annotations L_{Req}, L_{Resp}, and L_{Conn} in Figure 3, for example, have been added manually to assist understanding of the visualization shown, but the drawing method of a sub-class of the HTTP analysis class which calculated these intervals could trivially be extended to include them automatically. The TCP visualization shown in Figure 1 does not, similarly, flag changes in receiver window advertisements in the way that the NetLogger visualization tool might (rather, it shows the evolving sequence high-water determined by the combination of flow and congestion windows), but the TCP analysis class's drawing method could be trivially extended to do so.

3.4 Visualization as an Analysis Design Tool

Although visualization tools may be invoked to examine features of analysis results in detail, their main role is in the analysis development environment. The presentation of raw data in comprehensible and compact form assists in identifying relationships within the data and reasoning about those relationships when designing analysis algorithms.

Sections 3.2.1–3.2.2 illustrate the power of the visualization tools and comment upon the difficulty of comprehending and interpreting large masses of associated data without a mechanism for its succinct presentation — an essential precursor to analysis design. Figures 1 and 2 demonstrate the way in which visualization identifies and clarifies relationships, features and patterns of activity within the data — also essential to the design process. Both sections illustrate the volume and scope of trace data which may contribute to a single analysis result datum (e.g. the download time of a single Web page) and by implication the complexity of the analysis involved in its calculation.

The presentation of analysis output in conjunction with the data upon which it is based allows the relationships between raw data and analysis output to be examined and verified in detail, and the facility to repeat the analysis underlying selected results supports the close examination of the analysis process. Iterative design is supported by the facility to reload and re-execute analysis software on selected sets of data. The visualization tool may be instructed to *redraw* its displays: all analysis modules are *re-imported*, the underlying raw data is re-read from the trace file and re-analyzed with full tracing enabled and the display redrawn. Analysis code can therefore be modified *during the analysis process* and immediately re-run, and the modified analysis process examined in detail.

Although the visualization tool described is specific to a particular protocol set it is constructed in such a way as to contribute to a generic framework. As described in Section 3.3 functionality is divided between protocol-specific and display management classes (i.e. one set of classes provides and manages canvases and interactive features, another — given a set of raw data and results — knows how to draw them into the display provided). In some cases entirely new drawing classes may be required, but in general sub-typing will provide the appropriate functionality: the TCP tool's drawing class could be trivially sub-typed (with much removed functionality) to display the activity of UDP flows and the browser visualization tool's drawing classes sub-typed to accommodate other higher level protocol activity.

4. EXAMPLE: RECONSTRUCTION OF PAGE DOWNLOAD ACTIVITY

WWW traffic forms by far the largest single category carried by the current Internet. As such, its study motivates a considerable and wide body of research. This section describes how the data contained in Nprobe traces can be visualized, reconstructing the activity involved in the download of whole Web *pages*. While we assert that the ability to represent Web pages versus single TCP connections is a significant contribution of our Nprobe work. The details are beyond the scope of this paper.

Page downloads must be examined in terms of the objects downloaded (HTTP transactions); the TCP connections used and their dynamics; and client and server activity. In order to quantify the contribution of these elements, each must be identified, but the interactions between them can only be fully understood in the additional context of:

(a) The arrival time of each in-line link at the browser[5].

(b) The full set of objects and their dependency structure (i.e. how do the objects relate to each other, which objects contain links causing others to be downloaded).

The dependency structure within a page can be represented by a sparse directed acyclic graph (DAG) with objects as nodes and links as edges — it is convenient to refer to this structure as a *reference tree*. Because pages may be arbitrarily linked to one another, the reference structure of multiple pages does not necessarily form an acyclic graph, but again may be represented as such if the progress through a cycle is regarded as a return to a node visited earlier (to do so reflects the reality of re-using cached documents and page components). Because components (e.g. page decorations) are often shared between sets of pages the overall reference structure forms a set of DAGs with a number of shared nodes.

4.1 Reconstruction Using Data from Multiple Protocol Levels

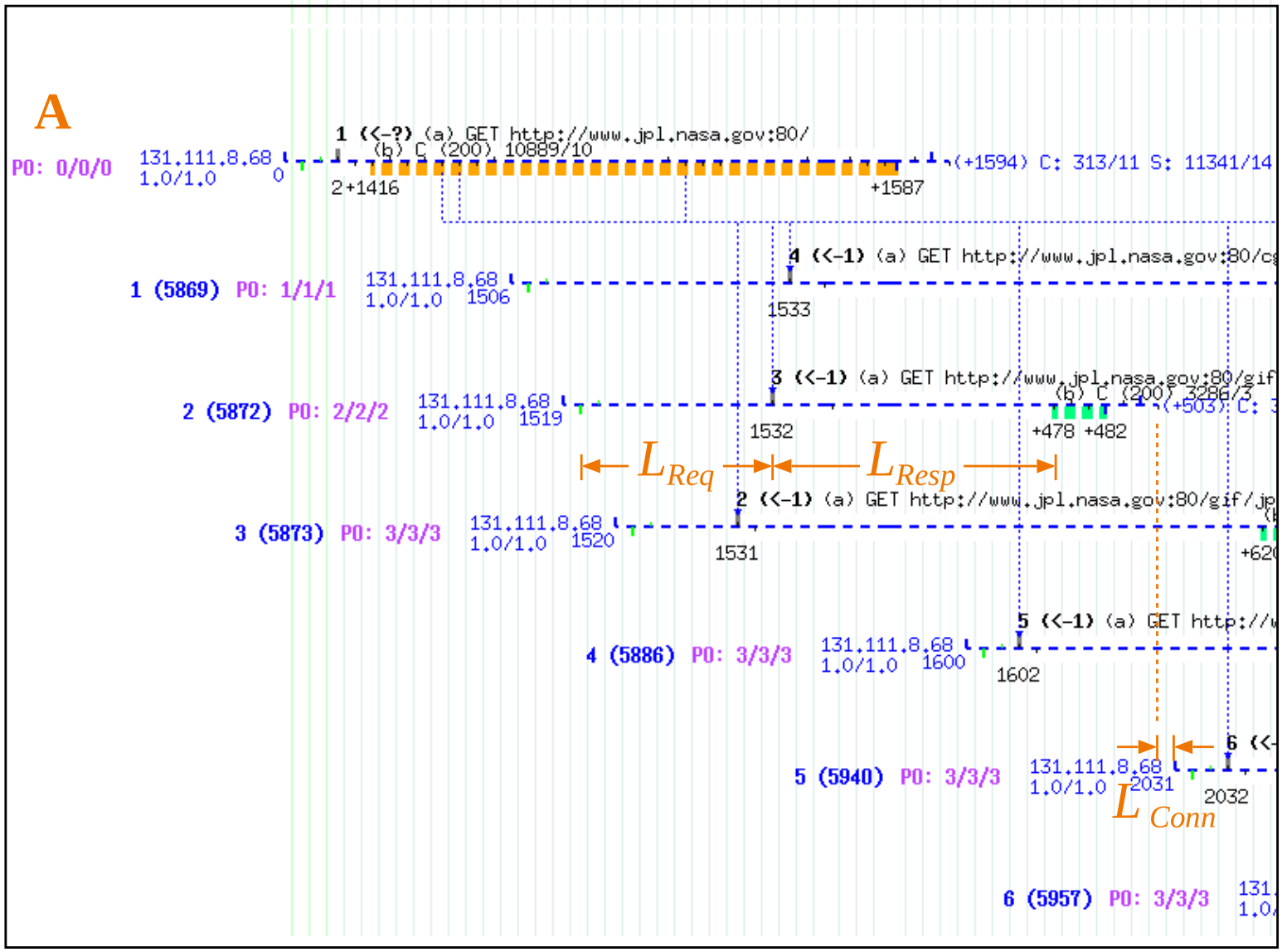
Using techniques yet to be published, object downloads are analyzed for page association and reconstruction. TCP connections are assigned to each object by client IP address; the structure of Nprobe trace files immediately associates each HTTP transaction with the TCP connection carrying it.

The relationship between objects is established by construction of the page reference tree(s) using HTML link and HTTP header data. The reference tree determines the relationships between transactions, the connections carrying them, the connections themselves, and hence any interactions of interest between protocols — the DAG representing the reference tree also represents an analogous graph in which the nodes are TCP connections, and which establishes the relationship between connections.

Figure 3 shows the relationships between the container document and in-line image objects for the early part of a typical page download. Analysis and deconstruction of the connection identifies time components derived from connection activity using TCP/IP header and HTML-level data. Thus L_{Req} and L_{Resp} indicate the lags from which browser request (`SYN-ACK` → request) and server response (request → first packet of response) latencies are calculated for the download of transaction #3.

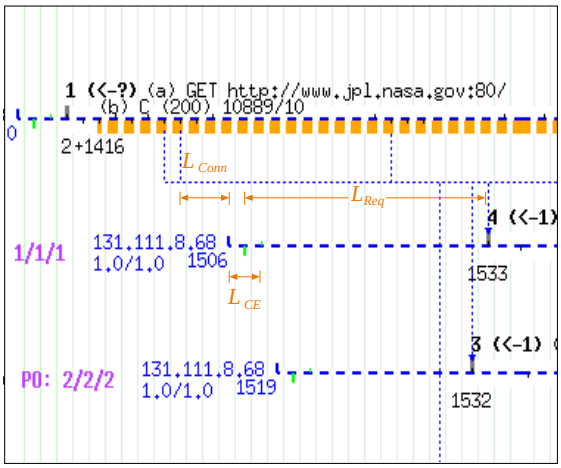
The notation 'PO: N/N/N' (**A**) denotes the number of connections 'presently open' as each new connection is opened, calculated by the three criteria (initial `SYN` → final `FIN-ACK`), (server's

[5]Analyzed in terms of lags and delays using techniques described in [3].



Note: the container document and early in-line images are shown. The visualization tool is in 'detail' mode; the fine vertical lines denote the missing 'slices' of time between events. Blue dotted lines represent the links associating objects, and show that HTML link data also identifies the data segment carrying the link and hence its *arrival* time at the browser. The reference tree also establishes the relationship between the TCP connections opened by the browser.

Figure 3: Detail from the Web activity visualization of a typical page download



Note: The request latency L_{Req} is measured from the receipt of the server's SYN-ACK and is consequently separated from L_{Conn} by less than the full TCP connection set-up latency L_{CE}.

Figure 4: Close detail of Figure 3 showing the browser connection latency

SYN-ACK → final FIN-ACK), and (server's SYN-ACK → first FIN) respectively — observation of many downloads suggests that all of these tests are applied by various browsers in deciding when a subsequent connection may be opened. For the connection shown the browser is maintaining four concurrently-open connections using the first criterion, hence connection #4 opens as a result of the full close of connection #0, and #5 follows #2.

The delay between the browser 'seeing' an in-line link and initiating a transaction to fetch the relevant object contributes browser connection latency. This value is quantified using HTML link and TCP connection data, and shown as L_{Conn} in the enlarged detail of Figure 4. Where connection open time is dependent upon the close of an earlier connection L_{Conn} is determined by connection timings as shown in Figure 3.

Connection ordering is normally determined by the ordering of the transactions carried, but it is interesting to note that this is not the case for the first three secondary connections (#1 – #3) shown. It is surmised that the browser has opened these connections speculatively as their request latencies are larger than those of following connections, which are ordered as expected. Figures 3 and 4 show detail of a page download with the visualization tool in *detail* mode: the ordering of events is maintained but time periods in which no events take place are 'sliced out' in order to allow detail to be shown — the horizontal time scale is therefore non-linear. In contrast, Figure 5 shows the entire page download in *time* mode; horizontal scaling is now linear, and the relationship in time between events is clear.

5. SUMMARY

This paper described a system for the visualization of network data containing multiple protocols. The visualizer can look both at protocol-specific behavior, such as that of a single TCP flow, and more-importantly, it is able to allow the visualization of interactions between protocols.

As was noted in Section 3, the current tool is designed with a specific set of protocols in-mind. However, a comprehensive use of type-classing and class-specific methods allow expansion of the visualizer for any number of network, transport or application protocols.

The tight integration between analysis code and visualization tools, and the way in which their invocation can be cascaded, provides a rich and productive design environment. The support for examination of the analysis process during design also facilitates validation of the analysis methods employed and hence underpins confidence in the results obtained.

This tool has become a critical resource in the development of our multi-protocol monitoring system; allowing the verification of the monitoring system, identification of new modes of behavior and the easy visualization of potentially overwhelming quantities of information.

Thanks to Richard Sharp and David Scott for their feedback on early drafts of this work.

6. REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens. RFC 2581: Tcp congestion control, April 1999. ftp://ftp.isi.edu/in-notes/rfc2581.txt.

[2] D. M. Beazley. Swig: An easy to use tool for integrating scripting languages with c and c++. 4th Annual Tcl/Tk Workshop, Monterey, CA., July 1996. Extensive SWIG documentation can be found at `http://www.swig.org/`.

[3] J. Hall, I. Pratt, and I. Leslie. Non-intrusive estimation of web server delays. In *LCN 2001 The 26th Annual IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, March 2001.

[4] J. Hall, I. Pratt, I. Leslie, and A. Moore. The Effect of Early Packet Loss on Web Page Download Times. In *Passive & Active Measurement Workshop 2003 (PAM2003)*, Apr. 2003.

[5] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and K. C. claffy. The architecture of CoralReef: an Internet traffic monit oring software suite. In *Passive & Active Measurement Workshop 2001 (PAM2001)*, Apr. 2001.

[6] A. Moore, J. Hall, C. Kreibich, E. Harris, and I. Pratt. Architecture of a Network Monitor. In *Passive & Active Measurement Workshop 2003 (PAM2003)*, Apr. 2003.

[7] T. Oetiker. MRTG: Multi-Router Traffic Grapher, 2003. http://people.ee.ethz.ch/ oetiker/webtools/mrtg/.

[8] S. Ostermann. Tcptrace. Details are available from the Tcptrace home page at `http://irg.cs.ohiou.edu/software/tcptrace/index.html`.

[9] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot. Analysis of Measured Single-Hop Delay from an Operational Backbone Network. In *Proceedings of IEEE INFOCOM 2002*, New York, NY, June 2002.

[10] The Data Intensive Distributed Computing Research Group (DIDC), Lawrence Berkeley National Laboratory. Netlogger. Details are available via the NetLogger home page at `http://www-didc.lbl.gov/NetLogger/`.

[11] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphic Press, Cheshire, Connecticut, 1983.

[12] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the Characteristics and Origins of Internet Flow Rates. In *Proceedings of ACM SIGCOMM 2002*, Pittsburgh, PA, Aug. 2002.

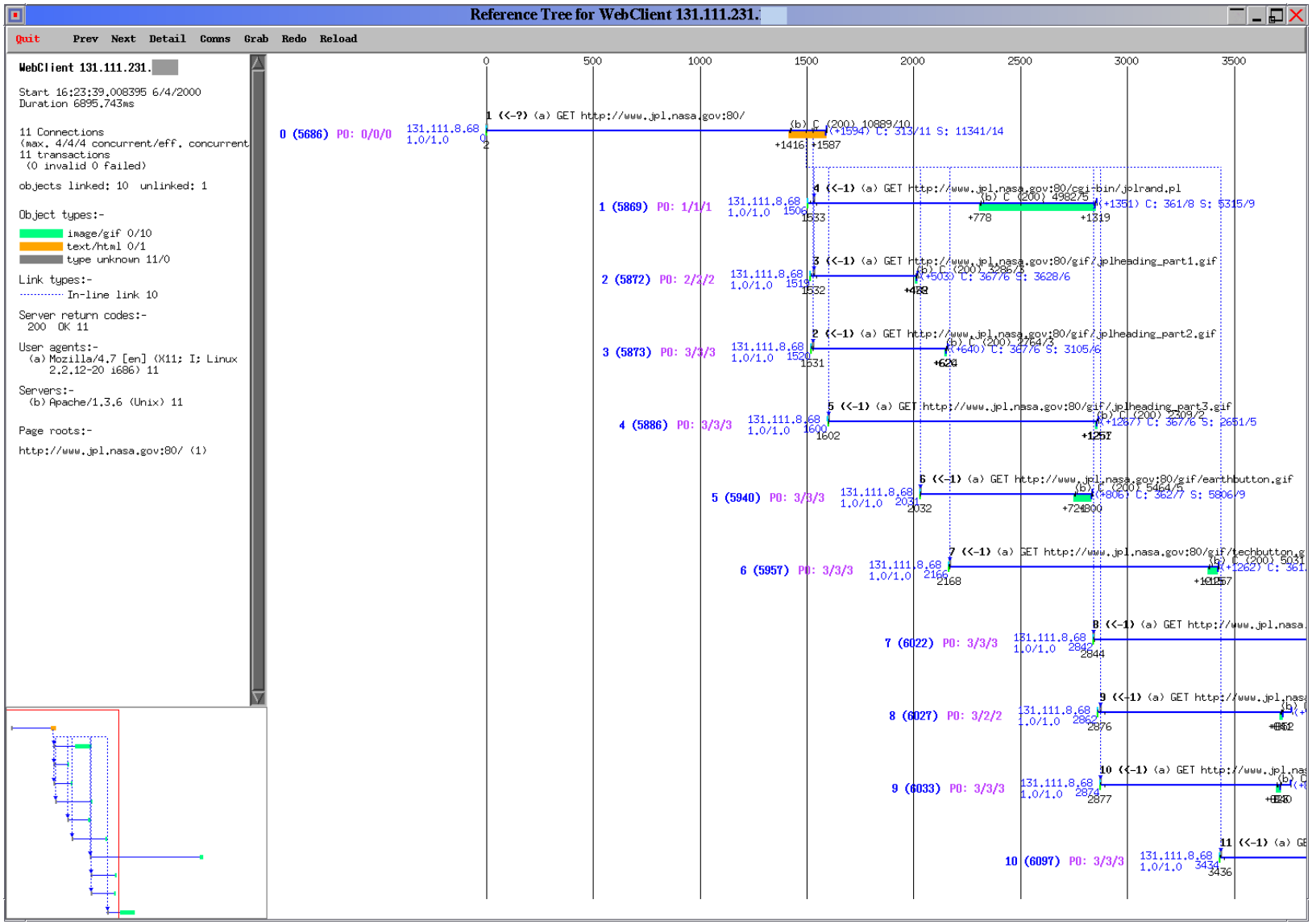


Figure 5: The whole of the page download shown in partial detail in Figures 3 and 4; the visualization tool is in time mode.