Observing Web Browser Behaviour Using the Nprobe Passive Monitoring Architecture (Extended Abstract)

James Hall, Ian Pratt and Ian Leslie
University of Cambridge Computer Laboratory
Contact Author: James Hall[1] (James.Hall@cl.cam.ac.uk)
University of Cambridge Computer Laboratory
New Museums Site, Pembroke Street
Cambridge CB2 3QG, UK
Tel: +44 (0)1223 334600, Fax: +44 (0)1223 334678

Abstract—**In this paper we will introduce a novel passive network monitoring architecture which enables us to capture and integrate data from different levels of the protocol stack using a probe which can be placed at any arbitrary point in the network. We subject the data gathered to off-line analysis and show how, by modelling the dynamics of the protocols observed, we are able to extract information which would not be available using existing means. In this paper, we show how the technique can be used to observe the network behaviour of Web Browsers, and the way in which they use TCP connections.**

I. Introduction and Motivation

The strengths of passive network monitoring techniques are well recognised, and have been comprehensively described in many previous works.

The entire set of data concerning the network's traffic and functioning is potentially available to the user of such techniques, but the volume of data collected may become unmanageable, particularly as network bandwidths and the volume of traffic carried increase. Past work has perforce largely been limited to the examination of behaviour observed at one level of the network protocol stack e.g. TCP, or to a limited set of the total data e.g. Netflow.

Many phenomena of interest, however, dictate that the network, communicating processes and the protocols employed be viewed as an integrated system. To do this the capture and integration of data from a range of protocols is required, together with monitoring architectures capable of extracting and storing the required data without loss due to overload. A *data reduction* ratio must be achieved which allows the capture of an adequate but minimal set of data tailored towards its intended use.

More recently monitoring designs aimed towards keeping pace with today's network bandwidths include the ambitious OCxMon architecture [1] and the AT&T Labs PacketScope [2].

The collection and integration of data derived from multiple levels in the network protocol stack is the subject of a growing research field, including the Windmill monitoring architecture [3], and in the BLT project [4].

The Nprobe design is aimed towards use of 'off the shelf' components, flexibility, scalability and ability to keep pace with high bandwidths, high data reduction ratios, no packet (data) loss, and minimal data extraction/reduction and copying overheads. While similar to some of the work described in [1][2][3] and [4] it differs in many significant respects. We have, so far, concentrated upon the probe infrastructure and protocol data extraction modules which allow us to study World Wide Web traffic, although the capabilities now open to us are wide ranging and may in future include examination of streamed media. We have conducted a preliminary study of Web Server delays described in [5].

Much work has focussed upon the efficiency with which Web Servers deliver objects over TCP, but relatively little has examined the contribution made by the browser. Nprobe extends data extraction/correlation to the contents of the web objects seen, in the case of HTML pages extracting link URLs.

Research based upon transfers of the *sets* of web objects comprising whole pages from the perspective of the client is relatively new ground, and argues a shift of emphasis from interest in server behaviour to that of the client. Whilst suggestions have been made to improve the efficiency of multiple-object transfers using TCP/HTTP, e.g.. HTTP 1.1 persistent connections and pipelining, it has not hitherto been possible to comprehensively study the efficiency with which these mechanisms have been employed, or to exactly what extent.

[1] James Hall is a Graduate Student at the Computer Laboratory

In Section II of this paper we describe the Nprobe online monitoring architecture, and in Section III we illustrate some of its capabilities by describing current work investigating Web browser behaviour.

II. Monitor design

A. The Nprobe Architecture

The Nprobe architecture runs over the GNU/Linux operating system, with modifications made to the kernel networking and memory management code to support the efficient passing of network buffers to and from user-space. Any network interface supported by Linux can be used with Nprobe, though in some cases we have chosen to modify the device driver to improve performance or add instrumentation. Currently deployed systems are dual Pentium PIII 500MHz machines, with storage provided by a software RAID array of high-capacity IDE disks.

Nprobe is currently equipped with Alteon ACEnic 1Gbs Ethernet cards fed by port monitoring via a local switch. They may alternatively receive packets directed to the probe via a pair of passive optical splitters inserted into the link of interest, as when previously monitoring ATM links. The cards' firmware is modified to attach time-stamps to incoming packets with an accuracy typically of the order of a few microseconds.

The modified firmware also provides a simple filter, based upon hashing of XOR'd source and destination IP addresses, which can determine to which analysis process packets should be delivered, or whether they should be dropped without transfer to the host. In this way we are able to exploit processor affinity of separate user level analysis processes, each dealing with a different subset of traffic. By employing an n-valued hash we accommodate the possibility of using a cluster of monitoring machines each handling a specified sub-set of the traffic, hence providing scalability to enable us to monitor very high bandwidth links.

The major components of the Nprobe design are shown in Figure 1.

Incoming packets are delivered into a input buffer pool by the network device drivers and are not processed further in the kernel, but by user-level modules as described in Section II-B. To avoid the overhead of a copy into user space the buffer pool is mapped in to the analysis process' address space and analysis/data extraction carried out in place. A large pool is provided in order to accommodate burstiness in incoming traffic and the inevitable variation in packet analysis times due to variations in packet length and complexity.

Current work has not involved connection to highly utilised networks, but Nprobe has been used to monitor

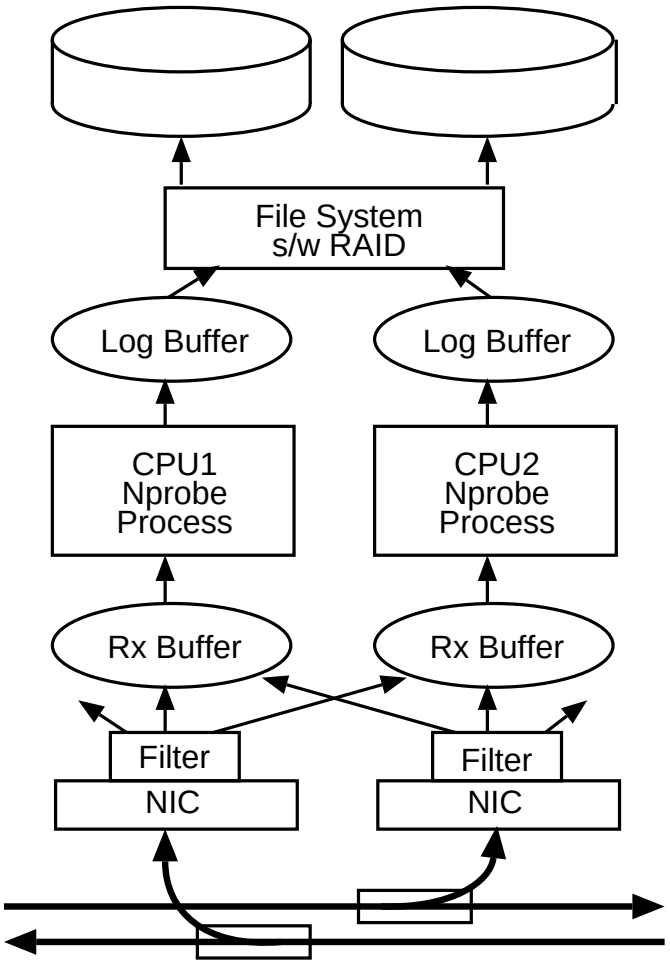

Fig. 1. Major Elements of the Nprobe Architecture

links where bandwidths of up to 120Mbs (including up to 96Mbs of HTTP traffic) have been observed without packet loss and with the consumption of less than 20% of available CPU cycles. When extracting data from TCP packet headers, HTTP headers and HTML object contents a data reduction rate of approximately 12:1 is achieved, rising to over 50:1 when less TCP header information is collected. The storage system should be capable of handling these logging data rates even if the input traffic were well in excess of a gigabit per second.

B. Analysis and Data Extraction

Packets collected by Nprobe are analysed 'on the fly' by user-level processes which extract the required data from the contained protocol headers and payloads — hence achieving a desirable rate of data reduction while not discarding potentially useful data – and save it for post-collection analysis. Each incoming packet is dealt with in its entirety and the containing buffer immediately returned for re-use by the input driver unless temporarily retained for TCP sequence ordering.

The analysis software is implemented as a series of

modules, providing flexibility and allowing monitoring and analysis functions to be composed and customised as required. Each module extracts protocol specific data, and where necessary de-multiplexes higher level protocols and delivers packets to their own processing module in an appropriate manner.

The TCP module will, for instance, note out of order packets, sequence gaps and retransmissions, and store and reorder packets as necessary to ensure that they are delivered in order to the HTTP module. Timeout mechanisms are employed to avoid buffer starvation in the case of long queues following packets seen by the receiving host but not by the probe (i.e. not resulting in a re-transmission), or to detect prolonged quiescence of connections resulting either from loss of FIN packets or routing changes leading to loss of the packet stream.

Considerable ingenuity has been required in the construction of data extraction modules in order to make them robust in the face of malformed or misused protocol header fields and badly behaving hosts. Some of the difficulties met in this respect are described in [4].

Extracted data and state are maintained in a hierarchy comprising host endpoints, port endpoints, TCP connections and (for instance) HTTP transactions. A flexible definition of 'flows' can thereby be employed and data relevant to the protocol levels of interest can be correlated throughout the hierarchy. HTTP transactions are immediately associated with the TCP connections carrying them, and requests and responses paired even in the presence of persistent connections or pipelined requests. Collected data is output in binary form at a granularity of a TCP/UDP connection.

The extracted data is not written out by the analysis process(es) but is written into a large output buffer. An independent 'writer' thread copies data from the output buffer to disk; in this way the analysis process(es) are available to examine packets and return buffers to the pool without delay, and data output consumes processor cycles only when not required for higher priority input processing.

C. Off-Line Analysis of Binary Data Logs

The data harvested from the network is later post-analysed using software which extracts the pertinent data and passes it to analytic modules specific to the study being carried out.

The post-analysis code is currently written in Python chosen for its powerful high-level data-types, object orientation and suitability for quick prototyping. The format of the Nprobe output files is based upon the C-language structures used by the Nprobe on-line analysis code; Python's extensibility using modules written in C allows fast access to and interpretation of the binary data as object classes via appropriate interfaces generated using SWIG.

III. Observing Web Browser Behaviour

The pages fetched by web browsers become increasingly complex with the increasing count of 'in-line' components — style sheets, images, frames, script elements — and with server-generated content. Performance, as experienced by the user no longer depends upon the download time of a single object, but upon that of a *set* of objects (we commonly observe web pages containing in excess of forty images, sometimes in excess of one hundred). The timely issuing of requests, together with the efficient use of TCP connections, is therefore crucial.

The factors contributing to the download times of single objects have been studied, but are often less significant than the delays contributed by the browser. Data collected using Nprobe allows us to reconstruct the 'reference trees' representing whole pages, as well as the performance of individual TCP connections. As we know when the browser first 'sees' each reference we can therefore assess the part played by its behaviour in overall download times.

Figure 2 shows the post analysis reconstruction of network activity while fetching of a small page from the University's web server. Horizontal bars represent the TCP connections opened; packets from the client are shown as tics above the bar, and those from the server below it. Heavy tics represent packets carrying data. The dashed lines indicate the constructed reference tree, and indicate which packets carried the links subsequently followed. Connection (i) is used to request an HTML document, (ii) a style sheet, and (iv) – (vi) three in-lined *gif* images. The server response to the GET request of (iv) is a single packet containing an 'object not modified' server response. 'Connection' (i) comprises two UDP packets containing a DNS request for the server and its response.

It is interesting to note that, in this small example, both client and server signified their willingness to use HTTP 1.1 persistent connections, yet the browser issued its subsidiary requests on two pairs of TCP connections, each being used only for a single request. There is a delay of approximately 500ms between the close of the last connection of the first subsidiary pair and the almost simultaneous opening of the second pair. The browser leaves the first connection, used for fetching the primary object, open but inactive for over 1500ms before transmitting a FIN packet.

Larger reference trees exhibit more complex, and often more puzzling, behaviour and, even at this relatively early stage in the study, pose many intriguing questions. We see the use of persistent connections in only approximately

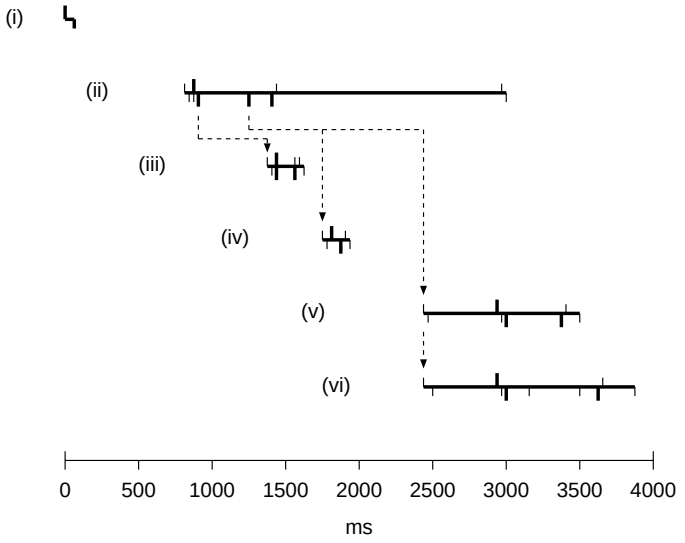

Fig. 2. The Reference Tree for a Small Web Page

10% of requests, and the use of 'pipelining' hardly at all — why should this be. Where multiple persistent connections are used we often note that requests are issued only after the receipt of outstanding responses on any of the connections, rather than concurrently. Reference to Figure 2, which although small is not untypical, shows that delays attributable to the browser are of significant magnitude in comparison the download times of small objects.

As we monitor traffic at some arbitrary point in the network we do not, of course, know exactly when packets arrive at either server or client. Fortunately this knowledge is not necessary — causality comes to our aid. We can measure the time interval between seeing pairs of packets travelling in opposite directions and which are associated by cause and effect, the arrival of one has made possible the transmission of the other. The measured interval consists of two parts: (a) a *partial* round trip time, i.e. the time taken for packets to traverse the network from the monitoring point to a communicating host and for returning packets to traverse the reverse path, and (b) any delay introduced by the host. By identifying causal pairs known not to involve any delay attributable to the host we are able to establish partial round trip times, and hence deduce the delay component for other pairs.

The establishment of causality is a non-trivial exercise, particularly as it may be determined at multiple levels in the protocol stack. We use our record of the TCP packets traversing a connection to construct a dynamic model of the connection which incorporates the changing state and behaviour of the two end-points. Such models are complex as they must allow for differing implementations and for packet losses both up and down stream of the monitoring point. Our modelling is similar to that described in [6], although to a different end, and is more fully described in [5]. A similar, but less complex model of HTTP behaviour is also constructed using data extracted from HTTP headers and is integrated with the TCP model. The output of the integrated model enables us to differentiate and quantify delays due to the server, client, network and protocol operation.

IV. Conclusions and Further Work

Nprobe has fulfilled its design in collecting datasets more comprehensive in scope than those previously available. Although it has not yet monitored traffic which would strain its capabilities we have been pleasantly surprised at the ease with which it has performed so far, as it effectively make three passes through the HTTP packets monitored; to verify the IP checksum, to parse HTTP headers and HTML documents, and to calculate an MD5-based signature of all objects seen.

The richness of the data gathered and the post-collection analysis techniques that we are developing promise rewards in several areas, including suggestions for the improvement of web browser and server performance, downloads of large and complex web pages, and cache performance.

References

[1] J. Apisdorf, K claffy, K. Thompson, and Rick Wilder., "Oc3mon: flexible, affordable, high performance statistics collection," .

[2] N. Anerousis, R. Caceres, N. Duffield, A. Feldmann, A. Greenberg, C. Kalmanek, P. Mishra, K.K. Ramakrishnan, and J. Rexford, "Using the at&t labs packetscope for internet measurements, design, and performance analysis," November 1997.

[3] G.R. Malan and F. Jahanian., "An extensible probe for network protocol performance and measurement.," in *Proceedings of ACM SIGCOMM'98.*, pp. 215–227. August 1998.

[4] Anja Feldmann, "BLT: Bi-layer tracing of HTTP and TCP/IP," *WWW9 / Computer Networks*, vol. 33, no. 1-6, pp. 321–335, 2000.

[5] James Hall, Ian Pratt, and Ian Leslie, "Non-intrusive estimation of web server delays," Tech. Rep., University of Cambridge Computer Laboratory: http://www.cl.cam.ac.uk/ Research/SRG/netos/netx/publications/sdel.html, May 2001.

[6] Vern Paxson, "Automated packet trace analysis of TCP implementations," in *Proceedings of the ACM SIGCOMM Conference : Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM-97)*, New York, Sept. 14–18 1997, vol. 27,4 of *Computer Communication Review*, pp. 167–180, ACM Press.