Exposing server performance to network managers through passive network measurements

Jeff Terrell*, Kevin Jeffay*, F. Donelson Smith*, Jim Gogan†, and Joni Keller†
*Department of Computer Science and †ITS Communication Technologies
University of North Carolina
Chapel Hill, NC 27599
Email: {jsterrel,jeffay,smithfd}@cs.unc.edu, {gogan,hope}@email.unc.edu

***Abstract*—We describe a novel approach to managing the application-level performance of servers within an enterprise network given only passively-collected TCP/IP headers of packets from a link connecting clients to servers. The analysis is driven by constructing, in real-time, a source-level model of the request-response exchanges between each server and all of its clients. By continuously monitoring traffic, we generate a statistical profile of numerous application-layer measures of server performance. Given these measures, and their distributions over time, a network manager can quickly triage server performance problems in generic terms without any knowledge of the servers operation or *a priori* interaction with the server's operators. This approach is illustrated by using a continuous, 2-month dataset taken from a border router on the UNC campus to diagnose an actual problem with a UNC web portal.**

I. Introduction

"Network management" in an enterprise environment often carries an implicit responsibility for ensuring the performance of network resources attached to the network, such as servers and printers. For better or worse, users tend to view the "network" broadly to encompass the services they rely on as well as the interconnections between machines.

In this paper we report on a novel means of monitoring the performance of network servers through the use of continuous, passive network measurement. The goal is to enable network managers to triage server problems using easily obtainable data, without ever having to instrument the server itself and without having any knowledge of the operation or function of the server. We also believe our methods can be easily extended to enable managers to discover server problems in real-time.

The essence of our approach is to passively collect TCP/IP headers of all connections transiting a network link connecting users to servers. The TCP/IP headers are "reverse compiled," in real-time, into a source-level model of the request-response dialog between every client and server. That is, given only TCP and IP packet headers, and without any knowledge of the application or application-level protocol generating the packets, we construct and store for each connection a structural model of the request-response application-level dialog between a server and a client. The model is application-independent and consists of the number and sizes of requests and responses, along with the times between each chunk. (See Section III.)

Using these data, over time we can generate a detailed historical profile of the application-level performance of every server on a network. This profile is expressed in generic terms such as the distribution of request sizes, response sizes, number of request-response exchanges per connection, response time, connection duration, connection arrival rate, *etc.* With these statistics, network managers are now in a position to discover and triage server performance problems by looking for unexpected deviations in short or medium term distributions of these measures. Most importantly, these analyses are performed using application-level measures of performance but without ever having to instrument any server or even have knowledge of the operation or function of the server.

To illustrate the method, we present a case study, using our methods to understand an actual server problem that occurred on the UNC campus. We have been passively monitoring the main campus Internet link continuously since March 2008. The results presented here are based on 1.5 TB of data on the more than 1.6 billion connections established to UNC servers up to May 23, 2008. (See Section V.) On April 8, 2008, a performance problem was reported for a UNC web portal ("server is slow").

Using our data we were able to quickly determine the most likely cause of the problem (an unexpected increase in request volume) and rule out a number of equally likely alternate causes (*e.g.*, an increase in the content being returned on each connection). In addition, we were also able to inform the server operators of quite detailed micro-performance measures such as a marked increase in the response time for the transfer of the third, fourth, and fifth objects returned within a connection. This suggested a possible linkage between the user-observed performance problem and any back-end server or database responsible for generating common content for the third through fifth responses.

Although the performance problem we analyze turned out to be a rather mundane problem, we were able to diagnose the problem and provide sophisticated data to the server's operators without any knowledge of the servers operation or *a priori* interaction with the operators. We believe the methods we have developed and are developing provide a powerful new means for network managers to understand both the performance of their network in broad terms that now include the application-layer performance of arbitrary servers attached to their network.

II. Related Work

Other network research is built on passive and continuous measurement. Caceres *et al.* [1] discuss the AT&T trace collection project, involving an elaborate infrastructure and many types of active and passive measurements. Fraleigh *et al.* [2] collect 3.3 TB per day of packet header traces, taking care to synchronize clocks on distributed monitoring points. Hussain *et al.* [3] design a traffic monitoring infrastructure for collection of packet header traces. All three of these works involve a much more elaborate and expensive infrastructure than ours, and they require more work to setup the infrastructure. Packet header traces are nearly as detailed a data source as one can reasonably hope to get, yet we claim that most of the information of interest to a network manager is preserved in our data, in much less space (roughly 4-5 %).

Malan and Jahanian's Windmill system [4] also collects continuous passive measurements. Like our work, theirs is a *one-pass* approach to processing the traffic stream, and there is a focus on application-level effects. Unlike our work, however, they use incoming packets to trigger events in application protocol "replay" modules, requiring significant work to create each such module. Instead, we measure application-level phenomena in a generic fashion, requiring no knowledge of any particular application.

Feldmann's BLT [5] describes in detail a method for capturing HTTP information, including request/response pairs. Like our work, it abstracts network and transport effects such as out-of-order delivery and retransmissions, instead focusing on the application-level data units exchanged between hosts. However, BLT focuses on HTTP and gleans information from the TCP payload, so it is not appropriate for encrypted traffic. Furthermore, it uses a multi-pass algorithm, and so is not suitable for high-speed links, which require a one-pass approach. Another approach, based on BLT, that has similar drawbacks relative to our focus is Fu *et al.*'s EtE [6].

Olshefski *et al.* introduce ksniffer in [7]. This work is similar to ours in approach in that it passively infers application-level response times on a high-speed link (therefore using a one-pass algorithm). Their goal is to measure, at the server, the time from a client's request of a web page to the receipt of all data by the client. Like other related work, however, the focus is on HTTP, leveraging knowledge of HTTP's functionality and requiring access to HTTP headers—which, again, is not possible in encrypted communication such as HTTPS. Furthermore, their work is not entirely passive, since the measurement occurs in the kernel of the server host and therefore steals cycles from the HTTP server application. Later work by Olshefski and Nieh on RLM [8] fixes this problem by performing the measurement on a separate machine; however, the machine is placed in-line with the HTTP server so that it can affect the traffic, so it also is not purely passive. Our work is purely passive, generic, and does not require access to TCP payloads, thus functioning equally well on HTTPS and HTTP (as well as any other sequential protocol).

An idea that we heavily exploit was introduced (as far as we can tell) by Barford and Crovella [9]. That idea was a user-level think-time, and it was used to realistically model HTTP users. We extend the idea to an *application-level* think-time, which functions the same whether the think-time is due to a user thinking or an application processing some input. This idea is the basis of our server-side *response time* metric, as introduced in Section IV.

III. Background

Our approach is based on a model of source-level interactions called the *a-b-t model*, described in [10]. The model focuses on application-level exchanges of data between hosts using a TCP connection. The application processes at the end systems exchange data in units defined by their specific application-level protocol. The sizes of these application-specific data units (called *ADUs*) are mostly independent of the sizes of transport-layer units (segments) or network-layer units (datagrams/packets). For example, in the file transfer protocol (FTP), a complete downloaded file is considered to be a single ADU.

There are three components of the model: the ADUs sent by the client (or connection initiator) to the server (*a*-units); ADUs sent by the server (or connection acceptor) (*b*-units); and *think-times* between exchanges (*t*). For our purposes, we will call the a-units *requests* and the b-units *responses* since most of the results presented here are for sequential, client/server applications. For example, consider HTTP. In the simple case of a non-persistent HTTP connection, there will be one request, followed by the (response) think-time, followed by one response. In the case of persistent HTTP connections, there will be a series of request-response exchanges, with server response times between the request and response and client think-times before each request. Each request, think-time, response, think-time pattern is called an *epoch*.[1]

The reason for using this model (and a key contribution of our entire approach) is that it reveals the *internal structure* of TCP connections from which application and user behavior can be inferred. In the results section we exploit this structure to examine the possible causes of observed poor server response times.

The key contribution of our approach is an online algorithm for constructing an a-b-t model of each TCP connection appearing on a monitored link in real-time. This online algorithm allows us to generate the models in an streaming fashion, requiring a single pass on a packet stream. An online approach is required for management purposes; otherwise, the data will become stale, and it will not be useful for urgent performance issues.

A complete description of the online modeling algorithm is beyond the scope of this paper, but it is based on the concepts and methods that have been evaluated and used in prior work ([11], [12], [13]). Briefly, we maintain per-connection, per-flow-direction state that is initiated upon seeing a SYN

[1]For pipelined HTTP requests and responses, the entire pipeline is treated as 1 request ADU and 1 response ADU.

packet and terminated at a FIN/RST. Whenever the algorithm determines that an ADU has ended, it prints a corresponding record, with the size and ensuing think time. There are many complications to this approach, and it is difficult to get right for all cases. For example, the algorithm must deal correctly with transport-layer and network-layer effects such as segmentation, retransmissions, out-of-order delivery, and losses.

A thorough validation of our algorithms is also beyond the scope of this paper. However, we have compared the output of the carefully validated offline algorithm used in [12] with our method using 109,000 randomly selected TCP connections. We found only a few minor differences related to the streaming algorithm's inability to "look into the future", none of which affected the calculation of ADU sizes, number of epochs, or server response times.

One complication is worth mentioning in more detail. In the common case of a *sequential* connection, in which each host waits to receive a complete ADU before sending a subsequent ADU, any new data sent in one direction will acknowledge all data previously sent in the other direction, thus ensuring that there is only one direction containing unacknowledged data at all times. However, in *concurrent* connections (*e.g.* NNTP, BitTorrent), a host might send new data without waiting to finish receiving an ADU. All connections are assumed to be sequential unless proved otherwise. In concurrent connections, we cannot be certain that the server is actually responding to the most recent request; instead, it could be sending a new ADU independent of the request. When we consider response times in this paper, we ignore those from concurrent connections because the servers we monitor are dominated (*i.e.* 95%) by sequential connections. [2]

IV. Measurement

The measurement setup for our approach is remarkably simple, and this is one of the key contributions in our work. We have a single monitor system with a passive tap of the bidirectional network traffic flowing between UNC and the Internet. The monitored link (a 1 Gbps Ethernet) is close to the core campus network switches to which a number of servers are attached.

The monitor system consists of four components: a Dell (Intel architecture) server-class system, an Endace DAG card for packet capture, a custom program that we call `adudump` that reads and processes the output of the DAG card, and disks for long-term data storage.[3] Taken together, the measurement setup provides a reasonably comprehensive picture of the operation of the network at an affordable cost and with minimal deployment issues.

The `adudump` program reads the memory-buffered stream of packet headers and timestamps provided by the DAG card and builds a model of the application-level dialog exchanged over each TCP connection.

The metric of most interest to users (and therefore of most concern to network managers) is server response time, defined specifically to be the time between the last packet of a client's request and the first packet of the server's response. The server response time typically involves highly variable operations, such as the scheduling of the server processes, the use of disk or remote resources such as distributed databases, and the execution of application code, all of which are components that contribute to the performance of a server. Furthermore, note that the transmission of packets on the UNC core network switches between the monitor and switches, except in very rare cases, add less than a millisecond to the response time. Since we are interested in response times of a hundred milliseconds or more, we can consider the transmission component of the response times measured at the monitor to be negligible.

V. Data

We ran `adudump` on the monitor continuously, save outages, for a period of 10 weeks. Overall, we collected about 1.5 terabytes of data modeling 1.6 billion TCP connections. Table I shows the individual `adudump` processes, interrupted by outages. `adudump` does not track a connection unless it sees the SYN segment; thus, there is a short "startup" period at the beginning of each collection time. Furthermore, note that, for reasons of privacy and scope, only incoming connections (*i.e.* connections with the initial SYN packet originating outside of UNC) were tracked.

The data is a sequence of records. All records share two things in common: the timestamp of the segment prompting the record, and the 4-tuple ID of the connection to which the record pertains. There are several types of records:

- the first SYN segment seen
- the RTT estimated from the connection handshake
- the connection establishment (*i.e.* completion of the TCP 3-way handshake)
- the connection termination (with either FIN's, RST's, or a combination)
- each ADU with its corresponding size and ensuing response or think time

The records are stored in uncompressed, human-readable text, with one record per line and fields separated by a single space. Compared to recording a full trace of all packet headers from this link (which could be used in an off-line analysis) we achieve a compression ratio of about 25:1 (about 22 GB/day vs over 550 GB/day on average). Further, as we show in Section VI, this data has a wealth of information useful for forensic analysis. More importantly, since the data is produced in real-time, it can be used in an "early warning" system to alert network managers to impending issues.

VI. Results

We have mined the archived data looking for incidents of poor response-time performance by the top-25 most-used servers at UNC. Almost all of them had at least one (and

[2]More precisely, we ignore response times from connections that have *already* proved to be concurrent. Since we use a one-pass algorithm, we cannot know whether the connection will later be flagged as concurrent.

[3]The long-term storage system is actually on a distinct system from the monitor.

TABLE I
DATA COLLECTION. ALL TIMES LOCAL (EDT); ALL DATES 2008. DATA FOR MONDAY, MARCH 17, WAS LOST.

#	begin	end	duration	outage duration	size	records	ADUs	connections established
1	Fri Mar 14 22:25	Thu Apr 17 03:50	33d 5h 25m	1d 14h 11m	813 GB	11.8 B	8.8 B	820 M
2	Fri Apr 18 18:01	Wed Apr 23 07:39	4d 13h 37m	0d 3h 35m	106 GB	1.6 B	1.1 B	116 M
3	Wed Apr 23 11:14	Thu Apr 24 03:00	0d 15h 46m	0d 7h 38m	16 GB	234 M	161 M	19 M
4	Thu Apr 24 10:38	Fri May 16 11:19	22d 0h 41m	0d 7h 4m	530 GB	7.7 B	5.7 B	532 M
5	Fri May 16 18:23	Fri May 23 00:06	6d 5h 43m	n/a	108 GB	1.6 B	1.07 B	148 M
*			66d 17h 12m	2d 8h 28m	1.54 TB	22.9 B	16.8 B	1.6 B

often several) such incidents lasting from a few minutes to several hours during the 10 week period. Unfortunately, due to space constraints, we opt for depth rather than breadth: we restrict our discussion to a more detailed analysis of a user-reported performance issue on a particular server. We present this analysis using data and graphs that could be generated with minimal delay, just as if we were responding to it as it happened.

At 4:28 P.M. on Tuesday, April 8th, 2008, the UNC network services group was notified that a server on campus was responding slowly to requests. We will henceforth refer to the time of this first report as T. We will dissect the behavior of this server in order to illustrate the power of our approach to inform the decisions of system administrators and network managers. From the real-time analysis, we would have available data for the server on (at least) two time scales: the summarized historical data over a long past interval and the data for one-hour intervals during the past few weeks. The latter is included because we do not expect any aspect of the traffic to be completely stationary. Furthermore, this sort of data is available for *all* servers within the UNC network. In fact, all data related to the particular server involved in this incident makes up only 2.5 GB, or about 0.02% of the dataset.

The email stated that the server was responding slowly to requests. Figure 1 plots the distribution of response times over the immediately past hour against historical response times. Two historical distributions are plotted: (1) the entire data set, from the beginning of measurement on March 14th until T minus 1 hour; and (2) the hour between 3:28 P.M. and 4:28 P.M. on Tuesday afternoons, for the three Tuesdays in the data set prior to April 8th.

Clearly, the server is now taking much longer to generate responses to requests, just as the incident report stated. There are several reasons why this could be. First, let's determine whether the application and user interactions are qualitatively different. Figure 2 shows the distributions of request sizes over the same time periods. There is no discernible difference between the distributions. Figure 3 plots the distributions of response sizes. Again, there is little real difference between past responses and current responses.

Figure 4 plots the number of epochs seen in each connection. (Recall that an epoch is a single request/response exchange.) This plot confirms that there is little qualitative difference between the connection structures in the historical data for this server and the current data. Further confirmation is found in the similarity of the distributions of total request bytes and total response bytes per connection, which are not shown here.

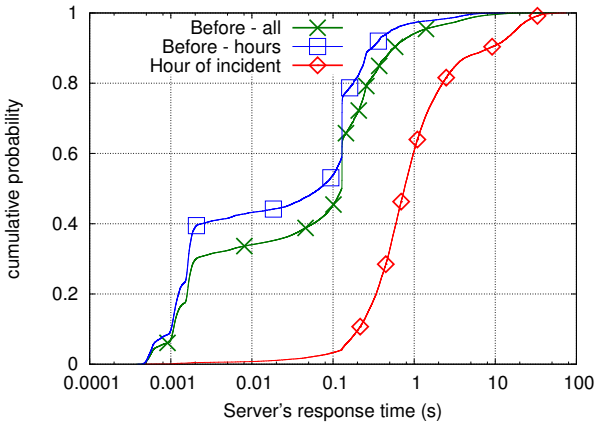

Fig. 1. CDF of response times for the server of interest. "Before - all" is all the response times in the data set until $T - 1$ hour. "Before - hours" is the three corresponding hour-long periods on Tuesday afternoons. "Hour of incident" is only the hour prior to the report of the incident.

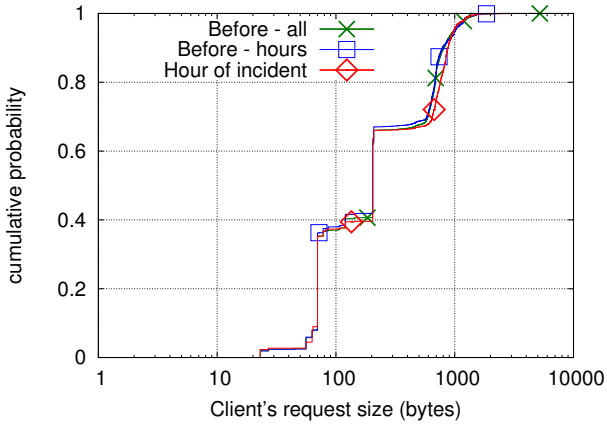

Fig. 2. CDF of request sizes for the server of interest.

At this point, given the unique structure of the request and response distributions, and the fact that the distributions are unchanged during the event, we can conclude that the application profile for the server is not qualitatively different than it is at other times. We emphasize that none of this data required any instrumentation of the server in question and can be derived solely from passive analysis of network packets in real-time.

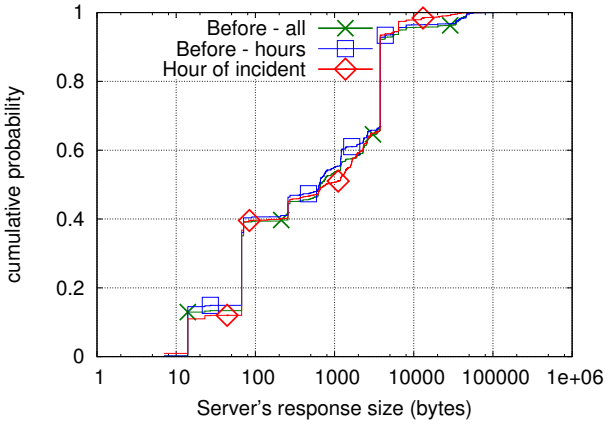

Fig. 3. CDF of response sizes for the server of interest.

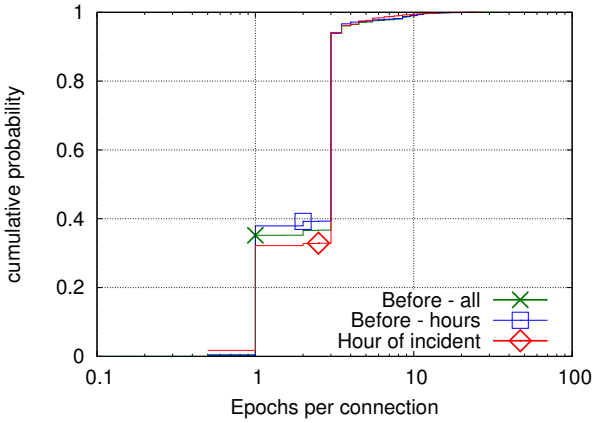

Fig. 4. CDF of epoch counts per connection for the server of interest.

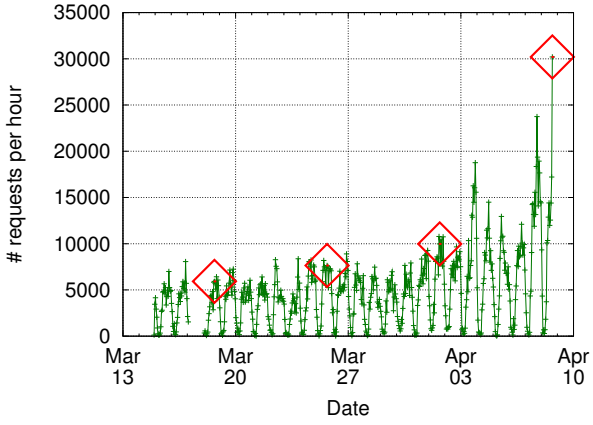

Fig. 5. The request count, per hour, up until the current incident. Hours end at :28 after the hour, which corresponds with the time at which the incident was reported. Tuesdays at 4:28 P.M. are marked.

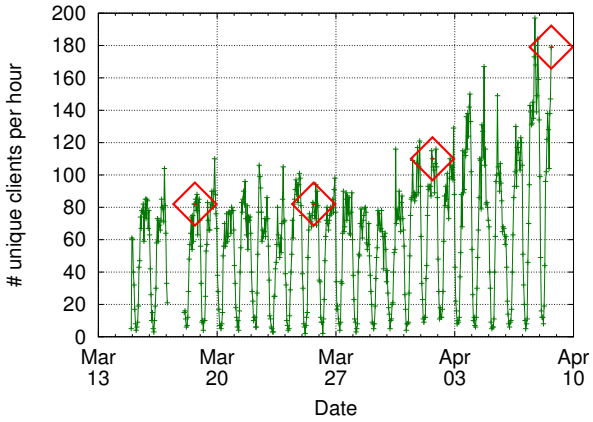

Fig. 6. The unique client count, per hour, up until the current incident. Hours end at :28 after the hour, which corresponds with the time at which the incident was reported. Tuesdays at 4:28 P.M. are marked.

Another obvious reason that the server might be responding slowly is that it could be simply overloaded with a higher load than normal. Figure 5 shows the number of requests to the server per hour. Note that the number of application requests per interval is a more direct performance metric than network-centric volume metrics such as the number of bytes or number of packets per interval. Since the server application is required to do processing per request, this metric is a better indicator of the server load than raw byte and packet counts.

There is clearly a large spike in the request count at the time of the incident. Therefore, we conclude that the server's performance woes are likely related to a request higher load than normal. One possibility is that the request count has increased because of a change in the content served by the server. For example, a web server could change its index page to one that references more objects, and we would expect that the requests (one HTTP request per object) would thus increase without a corresponding increase in the number of clients. However, Figure 6 shows that the number of unique users also increased along with the number of requests. Unless the content structure changed, the increased number of requests is likely a spike in users (a small "flash crowd").

The a-b-t model also allows us to look more deeply into the structure of the application data exchanges. Figure 7 shows the median response time for each epoch of connections by its order in the connection (persistent connections have > 1 epoch). In other words, the median of the i^{th} epoch's response time is plotted for each value of i. The median is plotted with range bars representing the first and third quartile (Q1 and Q3) of the response times. For example, the diamond with error bars at $x = 3$ is the median of all the 3^{rd} response times in the latest hour, along with the Q1 and Q3 of the same distribution. The count of response times is plotted as well (right axis). (Note that the count always decreases as i increases.)

There are many interesting things in this figure. First, it shows that the first two response times are not much different between normal operation and the hour containing the incident. However, the striking difference is in the third response time and beyond. The third, fourth, and fifth response times have a median roughly an order of magnitude greater than during regular load. Also, their variation (in terms of the inter-quartile range) is several times greater during high load. Note that

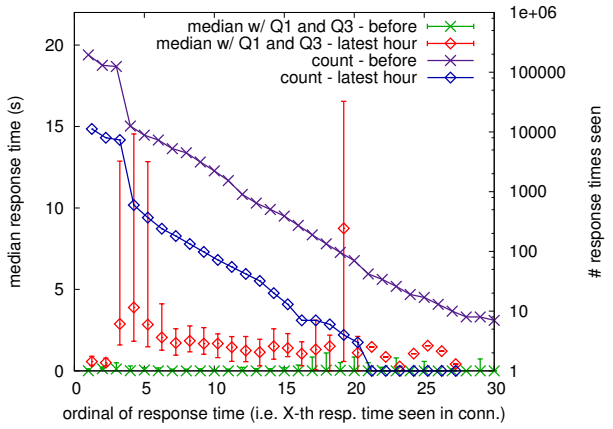

Fig. 7. Response times, by their order within a connection, along with the number of connections lasting i exchanges or more.

beyond the third response time, the number of responses drops precipitously. Thus, if we wanted to improve the average response time of the server during high load, we should focus on the third request/response exchange.

This information is, necessarily, rather generic. We do not know why the third request/response pair is distinguished, nor why most connections end after three exchanges. These questions can only be answered with some application-level knowledge, such as the application protocol or, in the case of web servers, the content structure. However, we claim that such information could be a useful clue when provided to the administrator responsible for the server. For example, perhaps the third request to a web server is typically for dynamically generated content, possibly with accesses to a distributed database. The administrator could then investigate whether the generation process could be optimized, cached, or otherwise improved.

Of course it is possible that all of this information could be retrieved for many applications by parsing server log files. Clever parsing, and maybe some augmentation to provide response times, could perhaps generate most of the results here. However, the contribution of our approach is that it works regardless of the particular server software used—or even the application-level protocol. Instead of gaining access to many servers, installing or writing code to parse logs (with augmentations as necessary), and combining the results across the servers, we simply monitor the network packets from a single vantage point and *infer* the application-level behavior; thus, our approach is more generic and elegant. Furthermore, our response times are *passively* measured are not subject to measurement artifacts from server instrumentation.

VII. Conclusions and Future Work

We have shown that by using an easily obtainable trace of TCP/IP headers, we can construct a source level model in real-time of all the client-server dialogs occurring on the UNC campus. The collected data enables a detailed characterization of the application-level performance of servers that is sufficient for diagnosing actual performance problems.

By continuously collecting data for the entire UNC campus, when a problem was reported, we were able to quickly extract historical performance data for the specific server and identify the most likely cause of the problem while simultaneously ruling out other equally possible causes. We were able to provide detailed information to the server's operators without any instrumentation of the server or even knowledge of the server's operation or function.

While the study reported here concerns an analysis of a server's performance after a report was received of a suspected problem, we believe that the methods we have employed are sufficiently generic that they could be employed in an automated fashion in an attempt to detect performance anomalies prior to their being noticed by a human operator. Such application-level performance anomaly detection is the subject of our current work.

VIII. Acknowledgements

We would like to thank John McGarrigle at UNC for his help understanding current management approaches and performance issues. This work was supported in parts by the National Science Foundation (grant CNS-0709081) and UNC Information Technology Services.

References

[1] R. Caceres et al, "Measurement and Analysis of IP Network Usage and Behaviour," in *IEEE Communications Magazine, 38(5):144-151*, 2000.

[2] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi, "Design and Deployment of a Passive Monitoring Infrastructure," in *Lecture Notes in Computer Science, 2170:556-567*, 2001.

[3] A. Hussain, G. Bartlett, Y. Pryadkin, J. Heidemann, C. Papadopoulos, and J. Bannister, "Experiences with a continuous network tracing infrastructure," in *Proc. SIGCOMM MineNet*, 2005, pp. 185–190.

[4] G. R. Malan and F. Jahanian, "An extensible probe architecture for network protocol performance measurement," in *Proc. SIGCOMM*, 1998, pp. 215–227.

[5] A. Feldmann, "BLT: Bi-Layer Tracing of HTTP and TCP/IP," in *Proc. of WWW-9*, 2000.

[6] Y. Fu, L. Cherkasova, W. Tang, and A. Vahdat, "EtE: Passive End-to-End Internet Service Performance Monitoring," in *USENIX ATEC*, 2002, pp. 115–130.

[7] D. P. Olshefski, J. Nieh, and E. Nahum, "ksniffer: determining the remote client perceived response time from live packet streams," in *Proc. OSDI*, 2004, pp. 333–346.

[8] D. Olshefski and J. Nieh, "Understanding the management of client perceived response time," in *ACM SIGMETRICS Performance Evaluation Review*, 2006, pp. 240–251.

[9] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *ACM SIGMETRICS Performance Evaluation Review*, 1998, pp. 151–160.

[10] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. Smith, "Tmix: a tool for generating realistic TCP application workloads in ns-2," in *ACM SIGCOMM CCR*, vol. 36, no. 3, 2006, pp. 65–76.

[11] F. Smith, F. Hernández-Campos, and K. Jeffay, "What TCP/IP Protocol Headers Can Tell Us About the Web," in *Proceedings of ACM SIGMETRICS '01*, 2001.

[12] F. Hernández-Campos, "Generation and Validation of Empirically-Derived TCP Application Workloads," Ph.D. dissertation, Dept. of Computer Science, UNC Chapel Hill, 2006.

[13] F. Hernández-Campos, K. Jeffay, and F. Smith, "Modeling and Generation of TCP Application Workloads," in *Proc. IEEE Int'l Conf. on Broadband Communications, Networks, and Systems*, 2007.