On the Performance of Middleboxes

Mark Allman
BBN Technologies
mallman@bbn.com

ABSTRACT

This paper presents a preliminary performance analysis of a complex middlebox infrastructure in a real-world production environment that serves several thousand people. While prevalent, middleboxes (firewalls, NATs, etc.) have yet to be systematically measured. This paper makes two contributions: (*i*) we outline several methodologies and metrics by which to measure middleboxes and (*ii*) we offer preliminary application-layer measurements of one particular production middlebox system. We show that the middlebox infrastructure in question offers a mixed bag of performance implications (both positive and negative). In addition, we quantify several failure modes introduced by the middlebox infrastructure.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems; C.4 [**Computer Systems Organization**]: Performance of Systems; C.2.0 [**Computer-Communication Networks**]: General

General Terms

Measurement, Performance, Experimentation, Security

Keywords

firewalls, middleboxes, TCP performance

1. INTRODUCTION

So-called "middleboxes", such as firewalls, address translators and proxies (among others), are prevalent in today's Internet architecture. [3] offers a discussion of the pros and cons of such devices. These smart network entities are used for a variety of reasons, for example:

- **Security.** Among the most common middleboxes is a firewall that is used to control traffic to implement security policy between networks. Firewalls range from simple devices that do not pass traffic with given characteristics (e.g., protocol number or port number) between the connected networks to complex devices that act as proxies for transport layer connections. Firewalls are not the only type of middlebox inserted into a path for security purposes. For instance, traffic normalizers [6] can be used to remove ambiguities from a traffic stream so that an intrusion detection system can better predict the effect of the traffic on an end host.
- **Performance.** A second class of middleboxes is used to increase the performance of standard networking protocols. For instance, web caches or content delivery networks (e.g., Akamai) are inserted into the network such that users do not have to retrieve content from its original source, but rather from a closer copy. In addition, various proposals and products allow boxes in the middle of the network to "assist" protocols (e.g., by retransmitting dropped segments on the sender's behalf [2] or controlling the sending rate by manipulating TCP's advertised window [7]). Finally, some middleboxes split an end-to-end transport connection into two (or more) connections in an attempt to shorten the control loop and enhance responsiveness on each stream (e.g., I-TCP [1]).
- **Address Translation.** A final common example middlebox is a network address translator [4]. These boxes change the network layer addresses and/or transport layer port numbers in traffic passing between two networks. One common use of this technology is to allow multiple internal hosts to share one external IP address. The need for this technique is sometimes due to the lack of global address space allocated to a particular network (and, hence, the desire to leverage that address space) or the desire to obfuscate internal addresses for security reasons.

While the reasons middleboxes have been added to the Internet's architecture are numerous the study of the impact of these entities in production environments has not kept up with deployment. For instance, several questions come into play when thinking about middleboxes, such as: What is the impact on performance (delay, application startup, throughput, etc.)? What is the impact on the reliability of TCP/IP protocols? What additional failure modes are introduced and how prevalent are these? In this paper we describe initial measurements of one particular middlebox infrastructure with the goals of: (*i*) defining a methodology

IMC'03, October 27–29, 2003, Miami Beach, Florida, USA.

for testing the impact of middleboxes using active measurements and (*ii*) gaining *preliminary insight* into the effect a large, production middlebox system has on traffic.

This paper is organized as follows. § 2 details our methodology and environment. § 3 outlines experiments involving small transactions traversing the middlebox infrastructure. § 4 discusses measurements of the delay involved in traversing the middlebox. § 5 discusses the persistence of the state required to be kept in middleboxes in the context of keeping TCP connections alive. § 6 outlines our experiments involving transmitting large data files. § 7 discusses measurements involving the File Transfer Protocol (FTP) [10]. Finally, § 8 gives our conclusions and sketches future work in this area.

2. EXPERIMENTAL SETUP AND METHODOLOGY

To measure the performance of a set of middle boxes at one facility we setup two client hosts at *Site1*[1], which is a production network serving several thousand people. In addition, we setup a server host at *Site2*, which is roughly 550 miles from *Site1*. The "inside client" is located inside the middlebox infrastructure (MBI) at *Site1* while the "outside client" is on the WAN side of the MBI. The clients are identical Intel Pentium 4 machines running Linux 2.4.9, while the server is an Intel Pentium 3 running FreeBSD 4.6. The server is located on the WAN side of the firewall infrastructure at *Site2*. This initial study only considers TCP [9] traffic. The networking stacks of all machines are left in their default configuration (e.g., TCP option usage, advertised window size, etc.). All experiments outlined in this paper were run concurrently inside and outside the MBI at *Site1*.

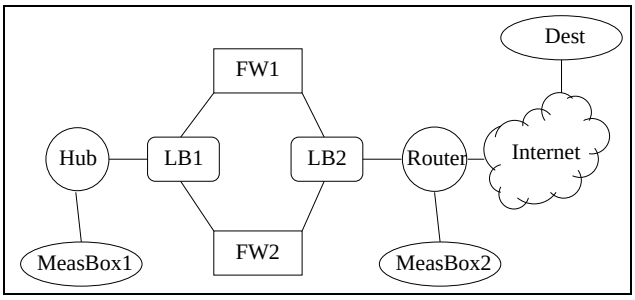


Figure 1: Simplified diagram of middlebox infrastructure.

Figure 1 shows the rough setup of the network (with "MeasBox1" and "MeasBox2" being the inside and outside clients, respectively). The MBI at *Site1* consists of several identical firewalls ("FW" boxes in the figure) attached to the local and wide-area networks by load balancers ("LB" boxes in the figure). The firewalls are stateful and proxy TCP connections. Therefore, the load balancers are also required to keep state (to always route the same TCP connection to the same firewall). We measured *Site1*'s MBI from October 14, 2002 – January 27, 2003 (roughly 105 days). The focus of this paper is not on the raw measurement values obtained in our experiments, but rather on the comparison between the measurements taken inside the MBI and those taken outside.

Also, note that in the descriptions of the experiments in the following sections we have chosen a number of constants (e.g., timeouts). These constants were chosen to be reasonable, but are largely arbitrarily. We believe this is fine because both clients share these constants and so the impact of the choice is likely to impact both clients. By choosing timeouts that are too short we may be biasing the measurements (e.g., if we had watched a little longer we might have seen a measurement complete, but rather we recorded it as a failure). However, we believe the chosen constants are large enough that we are not likely coercing a large number of these sorts of situations.

3. TRANSACTION DELAY

The first experiment's goal is to assess the impact of the MBI on the *transaction time* of a small request/response protocol. To measure the transaction time we use the *finger* protocol [11]. In our experiments the client opens a TCP connection to the server and sends a carriage-return and linefeed. The server then responds by sending two characters back to the client. The client verifies the returned characters and then closes the TCP connection. If the client does not receive a response within 30 seconds this is noted and the transaction is terminated. We use custom-written client and server code that timestamps each event in the transaction. We insert a delay between *finger* transactions of roughly 2 minutes (the actual time is determined using a Poisson process with a mean of 2 minutes).

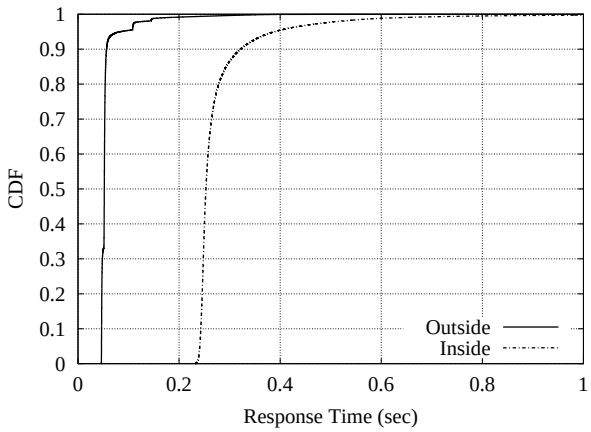


Figure 2: Distribution of response times for *finger* transactions.

Figure 2 shows the distribution of the response time for both clients. The dataset contains over 75,000 transactions for each client. The median transaction time outside the MBI is roughly 52 msec, while it takes roughly 253 msec from inside the MBI. The next section shows that the MBI adds minimal delay to data flowing through an already open TCP connection. Therefore, the five-fold increase in the *finger* transaction time shown in figure 2 is likely caused by the time required to instantiate connection state within the MBI.

We recorded 42 failures[2] on the inside client and 12 fail-

[1]The sites involved in the measurements conducted for this paper have requested anonymity.

[2]There are several ways to look at the "failures" reported in

ures on the outside client. All the problems outside the MBI are failures to connect to the server. Meanwhile, nearly all the problems inside the firewall are recorded as timeouts. Since the firewall is proxying TCP connections this likely translates into problems connecting to the server or internal problems in the MBI[3]. Since there are 3.5 times more failures inside the MBI we believe the majority of the problems are actually caused by the MBI. Finally, note that the overall failure rate both inside and outside the middle box infrastructure is quite low (under 0.1% in both cases).

4. FEEDBACK TIME

The last section focuses on small transactions and on the costs associated with initiating such transactions. This section focuses on delay through the MBI once a connection has been established. To accomplish this we wrote a "TCP ping" client and server. The client sends a small message with a sequence number to the server, which echoes the message back to the client via an already-established TCP connection. The client records when the request is sent and when the response arrives. After each request/response a delay is inserted before the next request is transmitted. Our dataset consists of five parallel pinging connections each using a different mean delay, x: 0, 30, 300, 1800 and 3600 seconds. The actual delay is determined using a Poisson process with a mean of x seconds. In addition, if a response does not arrive within 20 seconds the connection is closed and a new connection started in its place. In this paper we focus only on the experiments involving the connection that uses $x = 30$ seconds due to space constraints. The connections that used different granularities show the same basic results. We note that the measured values are not necessarily round-trip times, but rather application-level feedback times (FT). Since the pinging process is using TCP as its transport the requests and responses are sent reliably and therefore may incur retransmission delays if lost.

Figure 3 shows the distribution of FTs from inside and outside the MBI with a mean inter-ping time of 30 seconds. The dataset consists of over 303,000 pings from each client. As the plot shows, the FT experienced on either side of the MBI is nearly the same. The outside client experienced a roughly 1 msec shorter FT on median and just over 2 msec shorter FT on average then the inside client. This increase in delay through the MBI is likely explained by the additional number of local hops required to traverse the MBI.

Finally, we found only two types of errors: setting up the TCP connection and the connection being closed due to a timeout. We discuss the timeouts in the next section. The inside client failed to make a connection 51 times in our dataset compared to 46 times on the outside client. Since the instances of failure connection failure is roughly the same in both clients the likely explanation is that the network path between *Site1* and *Site2* became temporarily unusable.

this paper. We lump all problems experienced by the client together as "failures". However, this does not take into account that some failures are expected (caused by routine and scheduled maintenance of the MBI). However, we feel that such maintenance is part of the cost of using middleboxes and thus whether the failure was expected or not is not reported in our data.

[3]The MBI at *Site1* completes TCP's three-way handshake with the client before ensuring that a connection can be made with the server. Therefore, the inside client can finish its portion of the transaction before the MBI finishes the TCP 3-way handshake to start a connection to the server.

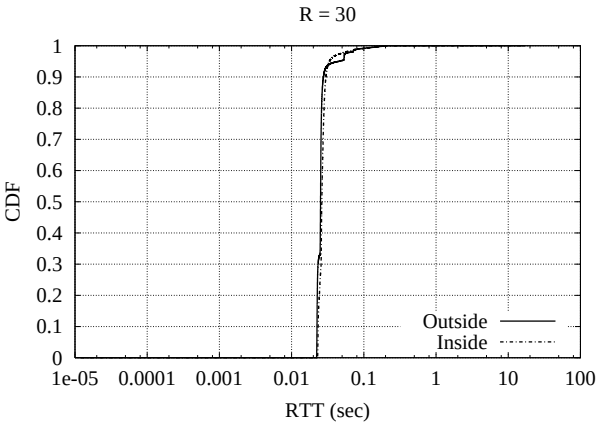


Figure 3: Distribution of feedback time when sampling roughly every 30 seconds.

Metric	Inside	Outside
Number of Connections	42	21
Mean Length (hr)	58.82	114.99
Median Length (hr)	19.99	43.45
Max Length (hr)	425.98	593.08

Table 1: Connection length statistics for connections that sampled FT roughly every 30 seconds.

5. CONNECTION LENGTH

In this section we discuss the length of the "ping" connections used for the FT measurements described in the last section. We are attempting to assess how often the MBI state gets internally unsynchronized, flushed or in some way makes an established connection unusable. Again, we analyzed all the connections with varying sending rates and the results come out consistent. Therefore, as in the last section, we only present the results of the connection that sends a request approximately every 30 seconds.

Table 1 provides the results of our analysis. We note that the inside client used twice as many connections as the outside client. The median connection length on the outside client is roughly twice as long as on the inside client. In addition, we note that the maximum connection length recorded on the inside client is roughly 21 times longer than the median connection recorded on the inside client, indicating that a client behind the MBI can sustain long connections. Also, the distribution of connection lengths recorded on the inside client does not suggest any sort of connection timeout in the MBI biasing the measurements[4].

We can further assign the blame of unnecessarily shortened connections to the MBI by noting that while there are twice as many connections used inside the MBI the instances of not being able to connect to the server (as outlined in the last section) are roughly equivalent across client. This

[4]In addition, the operational security team at *Site1* verified that there is no intentional timeout configured into the MBI that would effect these tests.

indicates that, in many cases, simply making a new TCP connection (and, hence starting over with fresh state in the MBI) to the server fixed the problem. These results suggest that something within the MBI was flushed or became unsynchronized.

6. BULK DATA TRANSFER

We wrote a simple client and server to test the raw TCP transfer speed through the MBI. The client sends 1 MB of data from memory to the server. The server discards the data upon receipt. In our experiments, we conduct bulk transfer measurements roughly every 10 minutes (where the actual inter-measurement time is dictated by a Poisson process with a mean of 10 minutes).

The last four bytes of data transmitted by the client contain a random number that the server echoes back. The transmission time is defined as the time between when the client application sends the first byte of data to the operating system for transmission until the client application receives the random number echoed by the receiver.

Waiting until the echoed random number is received is key to correctly measuring the end-to-end throughput. Clients, such as *ttcp*, that measure TCP transfer speed by considering the transfer "done" when the last byte of data is sent from the application to the operating system or even waiting on the `close()` call to return (after setting the socket to `LINGER`) may overestimate the end-to-end throughput attained. The problem is that the MBI proxies connections on behalf of the end-host. So, the end host's transmission of data bytes and their acknowledgment have no relationship with the ultimate recipient of the data (the server at *Site2*). Therefore, we introduce an application layer acknowledgment in an effort to measure the time required to actually transmit all the data to the ultimate receiver.

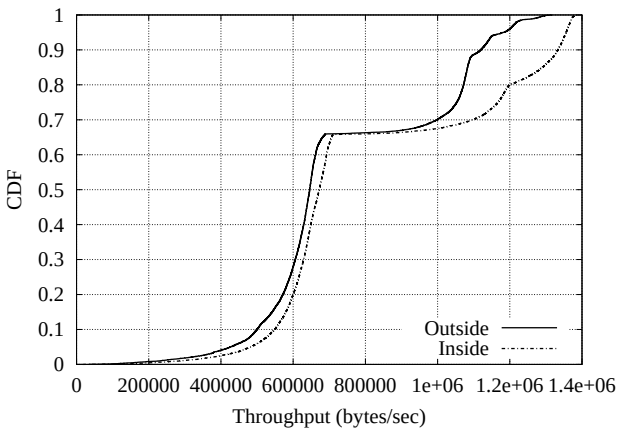


Figure 4: Distribution of throughput from bulk transfer experiments.

Figure 4 shows the distribution of throughput obtained in our bulk transfer experiments. Our dataset contains over 15,000 transfers from each client. The figure offers two immediate results. First, we note that the distribution of throughput attained (by both clients) is bi-modal. In addition, we observe that the inside client achieves higher throughput than the outside client.

The bi-modal distribution of throughput likely comes from

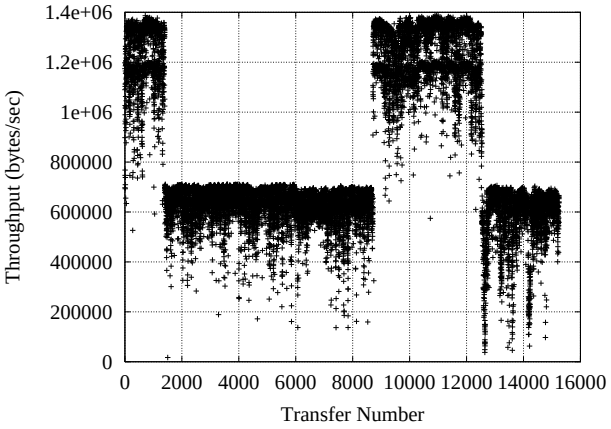


Figure 5: Throughput over time for transfers from the inside client.

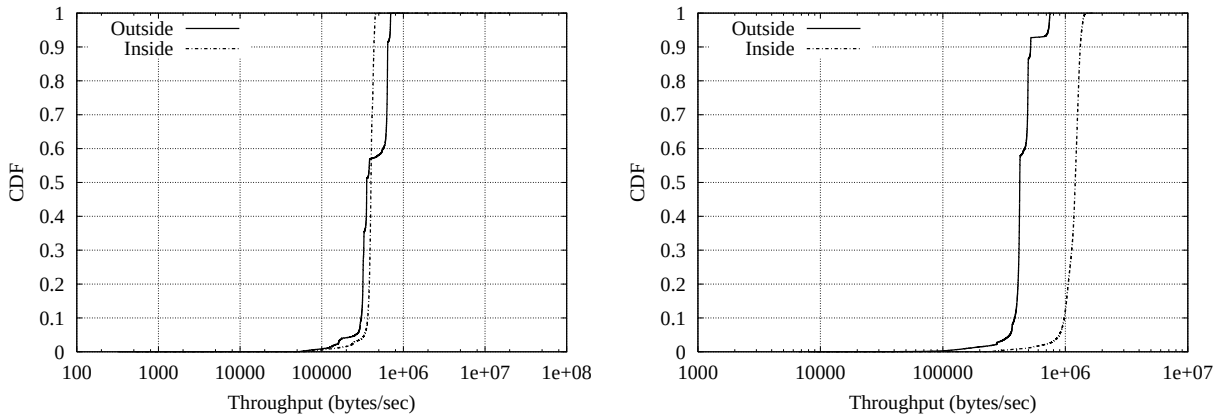
changes in the path between *Site1* and *Site2*. Figure 5 is a scatter plot of the throughput for each transfer conducted from the inside client as a function of the transfer number. The plot clearly shows distinct changes in the maximum throughput attained during various periods of our dataset. The distinct changes in the upper bound on performance could be caused by changes in the route between *Site1* and *Site2* or a change in some rate-limiting policy along the path. Without further measurements (e.g., traceroutes) we cannot say with certainty exactly what caused the change. (A like plot from the outside client shows the same pattern in throughput changes.)

The second item we notice from figure 4 is that the inside client obtains better throughput than the outside client. Looking at roughly the midpoint of each part of the distribution we see a difference in throughput of roughly 3.4% (at the 33^{rd} percentile) and 16.0% (at the 85^{th} percentile). This effect is difficult to explain without rich packet-level tracing at numerous points throughout the MBI. However, we offer a couple of *possibilities*. First, different variants of TCP offer different performance characteristics (e.g., see [5] for a comparison of loss recovery techniques and their impact on performance). Without packet-level traces we cannot quantify the impact of any differences that exist in the end-host TCP and the MBI's TCP implementation, however we believe the difference could explain some of the difference in the throughput measured. In addition, we note that the TCP model [8] offers insight into the throughput attained by concatenated TCP connections.

For our comparison the TCP model for throughput, T, distills down to:

$$T \propto \frac{1}{R\sqrt{p}} \tag{1}$$

where R is the round-trip time and p is the loss rate. The rest of the parameters in the model (e.g., the MSS) are static across connections in our experimental setup. From equation 1 it follows that reductions in either the RTT or the loss rate increase the throughput. For concatenated TCP connections we can use the model for each connection between the client and the server with the ultimate throughput dictated by the minimum of the throughputs calculated.



(a) File retrieval. (b) File transmission.

Figure 6: Distribution of FTP throughput.

From the setup of the MBI we know that there are several local hops between the inside client and the firewall that intercepts the end-to-end TCP connection and initiates a new connection to the server. And, from the FT analysis presented in § 4 we know that the median RTT is roughly 25 msec. Therefore, eliminating several local hops from the RTT could easily reduce the RTT of the TCP connection that ultimately connects the MBI to the server at *Site2* by 5–10% – thus yielding a like increase in throughput. This *could* account for much of the difference in throughput observed. However, future work in this area (using packet trace information) to confirm our sketch would be useful.

7. FTP

Next we look at a set of transfers made using FTP [10] between *Site1* and *Site2*. The MBI transparently intercepts FTP sessions initiated inside the MBI and silently proxies all connections. We used a modified BSD FTP client in our experiments and the standard FreeBSD FTP daemon on the server. The client was instrumented to dump timestamps of all events (request transmission, response arrival, etc.) during the session. In addition, we added a "sleep" command to the FTP client that sleeps for a random amount of time chosen using a Poisson process with a mean given by the user. We initiated FTP sessions approximately every 5 minutes (based on a Poisson process). In addition, between each FTP command we slept for approximately 2 seconds. Each FTP session consists of 35 commands and 4 file transfers (of 100 KB each). The client used both data connections opened actively and passively (using the "`PORT`" and "`PASV`" FTP commands) and both transmitted and retrieved files. The commands issued on the FTP control connection are as follows:

1. The "`USER`" and "`PASS`" commands to login to the FTP server.
2. The following 6 commands are issued (separated by approximately 2 seconds) prior to each file transfer: `TYPE A, CWD /, PWD, STAT, TYPE I, MDTM`. The next command sets up the data connection (either a "`PORT`" or "`PASV`" command), followed by either a "`STOR`" or "`RETR`" command to initiate the file transfer. This step is repeated for each file transfer (i.e., 4 times).
3. The "`QUIT`" command to terminate the session.

Figure 6 shows the throughput distribution for the transmission and reception of files via FTP inside and outside the MBI. The "`PASV`" command is used to setup the data connection[5] We make the following observations from the plots:

- The file retrievals perform comparably regardless of the location of the client[6].
- The throughput when transmitting files is higher when inside the MBI by nearly a factor of three at the median point. This is largely explained by the way FTP works and the MBI proxying the TCP connections. Since the proxy terminates the connection to the client and starts a new connection to the server the transfer becomes a LAN transfer from the client's perspective. Further, FTP does not include an application level acknowledgment (as the tests in the last section did). Therefore, as soon as the FTP client sends the last byte of data it considers the transmission finished even though all of the data has not yet arrived at the receiver.
- The throughputs obtained by the two sets of file retrievals and the file transmission from the outside client are comparable underscoring the fact that these transmissions are experiencing dynamics based on traversing the Internet while the file transmission from the inside client is only experiencing local network dynamics.
- Finally, we note that in the FTP tests the throughput obtained did not reach the upper bound of the lower

[5]The results for using "`PORT`" to setup the data connection are omitted due to space considerations, but are consistent with the "`PASV`" results presented in this paper.
[6]Unfortunately, our data is not rich enough to determine the cause of the knee in the plot around $y = 0.55$.

mode of the available bandwidth shown in the last section. Therefore, the distribution does not show the bi-model characteristic that the bulk transfer results illustrated. This is explained by the file size difference for the two transfers. In the FTP tests we used a 100 KB transfer, as opposed to the the bulk throughput tests that used a 1 MB transfer. The 100 KB tests did not afford enough time or data to open TCP's congestion window to fully utilize the available bandwidth.

Next we observe the time required for responses to commands sent on the control connection to arrive at the client. Figure 7 shows the distribution of the feedback time. The results are similar to the ping tests outlined in § 4, with both clients showing largely the same delay distribution. We note one glaring difference between the inside and outside clients at the low end of the distribution, where the inside client shows lower delay than the outside client. This anomaly is caused by the FTP implementation in the MBI, which does not understand the "`MDTM`" command (used to determine the modification time of a given file). Therefore, the inside client receives an error from the MBI for these commands – which experiences only local network delays, while the outside client receives the correct response from the server – incurring the Internet path delays.

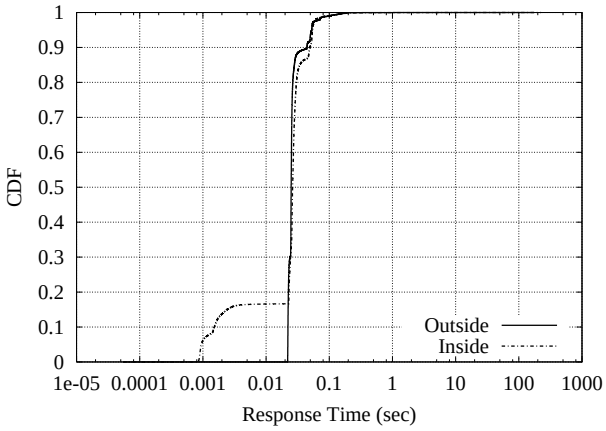


Figure 7: Distribution of time required for commands on FTP control connection.

The success rate of both the inside and outside FTP transfers is over 99.99% for over 121,000 file transfers. The most prevalent problem encountered was with the data connection not being available. This problem was encountered in roughly the same number of cases for the inside and outside client and so does not appear to be a problem with the MBI. Finally, we note that each client issued just over one million commands on the control connection. Every command issued completed successfully on the outside client. Two commands failed on the inside client (not counting the "`MDTM`" failures discussed above).

8. CONCLUSIONS AND FUTURE WORK

From the results presented in this paper we find that the measured MBI offers a mixed bag of performance costs and benefits. For instance, we note that setting up a short transaction takes roughly 5 times longer when traversing the MBI. However, once a TCP connection is setup, the added delay required to traverse the MBI is small (1 to 2 msec). Additionally, transmitting files from inside the MBI to a server across the network is faster than transfering the data from outside the MBI. Therefore, our conclusion is that the impact of the MBI on performance is application dependent. Finally, we note that the MBI generally increases the instances of failures in the network across all of our experiments. However, the application success rate is over 99.9% in all of our experiments. So, even though the MBI increases the failure rate by several times in some cases the overall success rate is high.

We see two major areas for future work in the area of measuring middleboxes: (*i*) measuring a larger number of production environments to assess whether the results from *Site1*'s network are representative and (*ii*) capturing packets at each step through an MBI and reconstructing the events to determine the root causes of the performance costs and benefits, as well as the failures noted in our results.

Acknowledgments

This paper has benefited from the contributions of a number of people. Engineers at *Site1* and *Site2* aided in the setup and design of the experiments presented in this paper. In addition, Vern Paxson provided encouragement and useful conversation throughout the work. My thanks to all!

9. REFERENCES

[1] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS)*, May 1995.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP Performance Over Wireless Networks. In *ACM MobiCom*, Nov. 1995.

[3] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations, June 2001. RFC 3135.

[4] K. B. Egevang and P. Francis. The IP Network Address Translator (NAT), May 1994. RFC 1631.

[5] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, 26(3), July 1996.

[6] M. Handley, V. Paxson, and C. Kreibich. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *Proceedings of USENIX Security Symposium*, 2001.

[7] S. Karandikar, S. Kalyanaraman, P. Bagal, and B. Packer. TCP Rate Control. *ACM Computer Communication Review*, 30(1):45–58, Jan. 2000.

[8] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, Sept. 1998.

[9] J. Postel. Transmission Control Protocol, Sept. 1981. RFC 793.

[10] J. Postel and J. Reynolds. File Tranfer Protocol (FTP), Oct. 1985. RFC 959.

[11] D. Zimmerman. The Finger User Information Protocol, Dec. 1991. RFC 1288.