Jack Zinman
General Attorney

AT&T Services, Inc.
1120 20th St. NW, Suite 1000
Washington, D.C. 20036
Phone 202 457-3053
Fax 202 457-3074

April 25, 2008

VIA ELECTRONIC SUBMISSION
Ms. Marlene H. Dortch
Secretary
Federal Communications Commission
445 12th Street, SW – Lobby Level
Washington, D.C. 20554

Re: *Broadband Industry Practices*, WC Docket No. 07-52

Dear Ms. Dortch:

On April 22, 2008, Vuze, Inc. ("Vuze") submitted a "study" in the above-referenced docket, in which it purports to have used a Vuze "software plug-in" installed by end users on their computers to measure the incidence of "network connections interrupted by reset messages." Vuze also submitted copies of letters that it sent to certain network providers, including AT&T, in which it enclosed a copy of its "study" and requested information on the network management practices of those providers.

In response, AT&T provided the attached letter to Vuze. The letter references a white paper from the Department of Computer Science at the University of Calgary, which "show[s] that reset connections arise from local events such as network outages, attacks, or reconfigurations, as well as from global trends in TCP usage."[1] The white paper concludes that "application-level browser behaviour is the primary contributor to the global trend of abnormal TCP behaviours"[2] and that "[c]orrecting the TCP behaviour of a popular Web browser would likely eliminate most of these TCP RST anomalies."[3] A copy of the white paper is enclosed for the Commission's convenience.

If you have any questions or need additional information, please do not hesitate to contact me. Pursuant to section 1.1206 of the Commission's rules, this letter is being filed electronically with the Commission.

Sincerely,

/s/
Jack Zinman

Enclosures

[1] M. Arlitt and C. Williamson, *An Analysis of TCP Reset Behaviour on the Internet*, University of Calgary (2004) available at http://pages.cpsc.ucalgary.ca/~carey/papers/2005/TCP-Resets.pdf.

[2] *Id.* at 8.

[3] *Id.*

at&t

Charles R. Kalmanek, Jr.
Vice President –
Internet & Network Systems Research
AT&T Labs, Inc.

180 Park Avenue
Room A147
Florham Park, New Jersey 07932

973-360-8720 office
973-360-8871 fax

April 24, 2008

Gilles BianRosa, CEO
Vuze, Inc.
558 Waverley Street
Palo Alto, CA 94301

Re: Letter to Randall Stephenson

Dear Mr. BianRosa:

Thank you for your recent letter to Randall Stephenson regarding network management practices. AT&T shares Vuze's belief that industry collaboration is essential to address the complex network management challenges facing the Internet community. To that end, we encourage Vuze to raise network management issues through the Distributed Computing Industry Association (DCIA) and its consensus-building forums, such as the P4P Working Group, of which Vuze and AT&T are both members. By doing so, Vuze may work cooperatively with other application and content providers, including BitTorrent, Pando, and Joost, and network providers, such as AT&T, Verizon, and Comcast, toward mutually beneficial solutions that enhance the Internet experiences of all of our customers.

In response to your specific question about AT&T's network management practices, AT&T does not use "false reset messages" to manage its network. We agree with Vuze that the use of the Vuze Plug-In to measure network traffic has numerous limitations and deficiencies, and does not demonstrate whether any particular network providers or their customers are using TCP Reset messages for network management purposes. Given that Vuze itself has recognized these problems with the measurements generated by its Plug-In, we believe that Vuze should not have published these misleading measurements, nor filed them with the FCC. Moreover, as Vuze and others have acknowledged, TCP resets are generated for many reasons wholly unrelated to the network management practices of broadband network providers, which explains why resets may appear on networks of companies such as AT&T who do not use TCP resets for network management (see, e.g., *An Analysis of TCP Reset Behaviour on the Internet*, University of Calgary (2004)). In any event, we would appreciate reviewing in the DCIA and P4P forums the methodology that Vuze used to collect network traffic information so that we may better understand the basis for your concerns regarding TCP Reset messages and service provider network management practices.

We look forward to hearing from and working with Vuze in the consensus building forums cited above to discuss our mutual concerns in greater detail.

Sincerely,

Charles R Kalmanek, Jr.

An Analysis of TCP Reset Behaviour on the Internet

Martin Arlitt Carey Williamson
Department of Computer Science
University of Calgary
Calgary, AB, Canada, T2N 1N4
E-mail: {arlitt,carey}@cpsc.ucalgary.ca

October 30, 2004

Abstract

This paper presents a one-year study of Internet packet traffic from a large campus network, showing that 15-25% of TCP connections have at least one TCP RST (reset). Similar results have also been observed from measurements of other Internet links. The results in this paper show that reset connections arise from local events such as network outages, attacks, or reconfigurations, as well as from global trends in TCP usage. In particular, we identify application-level Web behaviour as the primary contributor to the global trend in reset TCP connections. The most prevalent anomaly is the absence of the normal FIN handshake for connection termination. Instead, connections are often reset by the client. We believe that particular implementations of HTTP/TCP connection management cause this global trend.

1 Introduction

The Transmission Control Protocol (TCP) is a connection-oriented transport-layer protocol that provides reliable byte-stream delivery between two hosts on a network [8]. TCP is the dominant transport-layer protocol on the Internet today, carrying more than 90% of all data traversing backbone links [4, 9, 10].

A normal TCP connection passes through several distinct states, from the connection establishment phase to connection termination. All TCP implementations should follow this behaviour, to ensure reliable data transfer to any other TCP-enabled host on the network.

Recent Internet traffic measurements suggest that many TCP connections are not following the rules. For example, in a recent study of Internet-bound traffic from a large campus network, 20-80% of TCP connections observed are abnormal, in that they experience at least one TCP reset. Similar results were observed in our measurements of backbone links operated by a commercial network service provider. This phenomenon has been observed on other networks as well, including Lawrence Berkeley Labs (LBL) [7].

In this paper we show how local events and global trends are responsible for this behaviour. One of the global trends is the absence of the normal FIN handshake for connection termination. Instead, many connections are reset by the client (i.e., the originator of the connection), and others by the server. While there are many possible reasons for this, including user behaviours, erroneous TCP implementations, and misbehaving network devices, evidence suggests that network applications (e.g., Web browsers) are the real cause.

Our goal in this paper is to identify the different causes of abnormal TCP connections, quantify their frequency of occurrence, and determine whether they are intentional or unintentional.

The remainder of the paper is organized as follows. Section 2 provides some background information on TCP and the terminology used in this paper. Section 3 describes the experimental methodology for our measurements. Section 4 presents the results from our study. Finally, Section 5 concludes the paper.

2 TCP Background

TCP is a connection-oriented, end-to-end reliable-byte-stream transport-layer protocol [8]. It is widely used on the Internet and in the Web. This section defines TCP terminology relevant for understanding this paper. More detail on HTTP TCP behaviour appears in [2].

The fundamental unit of data transfer in TCP is a *byte* (i.e., for sequence numbering, flow control, and error control purposes). However, TCP implementations generally work with a larger logical unit size called a *segment* when transmitting packets across an IP internetwork. The Maximum Segment Size (MSS) is a settable parameter for a TCP transfer. The choice of the MSS typically depends on the Maximum Transmission Unit (MTU) size supported by the underlying network layer. In most instances, each TCP segment is carried in one IP packet; hence we use the terms segment and packet interchangeably throughout the paper.

The task of TCP is to divide the application-layer data into one or more segments, transmit them across the network, and deliver them reliably (and in order) to the receiving TCP. Each segment carries an explicit sequence number, for the purposes of ordering and reliability.

TCP uses a three-way handshake for reliable connection management. The opening handshake uses the SYN flag bit in the TCP packet header. The purpose of this handshake is to ensure that the other endpoint exists and is willing to establish a connection, and that the request is genuine (i.e., not a delayed duplicate, or a replay attack). During this handshake, the two endpoints establish the starting sequence numbers for data transfers in each direction, set connection parameters (e.g., MSS), and negotiate desired options (e.g., timestamps, SACK, or the window scale option for high bandwidth-delay product networks).

The closing handshake uses the FIN flag bit in the TCP packet header. The purpose of this handshake is to ensure that each endpoint has received, delivered, and acknowledged all application-layer data that was sent while the connection was open. Either endpoint can initiate the close of the connection. Once each endpoint has sent a FIN and acknowledged the other endpoint's FIN, the connection is marked as CLOSED. The connection record then remains in a TIME_WAIT state for a few minutes before being recycled for use as another TCP connection state record.

The RST (reset) bit in the TCP packet header is used to signal error conditions detected by TCP. For example, the arrival of a data packet for which no connection is open would generate a TCP reset. Similarly, the arrival of a TCP segment with an inappropriate sequence number, or the arrival of a SYN ACK packet for which no SYN had been initiated, would also trigger a TCP reset.

The TCP RST provides the means for a TCP endpoint to indicate that something seriously wrong has happened within the network, and that a new TCP connection needs to be established, if reliable communication is to continue. In normal operation, TCP resets are a relatively rare event, analogous to MAC-layer collisions on a CSMA/CD Ethernet LAN (i.e., the exception, rather than the rule).

Our recent measurements indicate that a disproportionately large number of TCP connections experience TCP resets. Our measurement experiments seek the cause(s) of this anomalous behaviour.

3 Experimental Methodology

There are two main parts to our study: passive measurements from a campus network, and active measurements with different client and server platforms. This section describes the methodology for each part of the study.

3.1 Passive Measurements

The primary data set in our paper comes from passive network traffic measurements of the campus network at the University of Calgary. In cooperation with staff from University of Calgary Information Technologies (UCIT), the administrators of the campus network, we installed a network monitoring machine at the main campus router.

The monitoring machine is a dual-processor Dell (two 1.4 GHz Pentium III processors) with 2 GB RAM and 140 GB of disk. The campus network backbone is currently connected to the commercial Internet via a 100 Mb/s full-duplex Ethernet link. The traffic on that network link is forwarded (via port mirroring) to our monitor via a 1 Gb/s half-duplex Ethernet link.

Table 1: Application Software and Operating Systems Tested

Software	Versions
Client Web Browser	Mozilla 5.0, Internet Explorer 6.0
Local HTTP Server	Apache 1.3.27
Remote HTTP Server	Apache 2.0.52, Microsoft IIS 6.0, Zeus 4.4, SunONE 6.0
Client Operating System	RedHat Linux 9.0, Windows 2000

We use *tcpdump*[1] for our network measurements. It allows collection of TCP/IP packet headers, and (optionally) packet payloads as well. For the primary measurements in this paper, we configured the software to collect all TCP SYNs, FINs, and RSTs traversing the University's commercial Internet backbone link. These packets are recorded to a file, with a new file created for each 1 hour of network traffic. The recorded files are then moved off the monitoring machine to an analysis machine. The 24 files for each day are concatenated together (in timestamp order), and processed using *Bro*[2]. *Bro* was originally designed for network intrusion detection, and thus is a very powerful tool for analyzing network traffic. We have written a number of scripts in *Bro's* scripting language which we use to specify our analyses.

The measurement results in this paper focus on two traces. The first covers the year-long period spanning October 1, 2003 through September 30, 2004. This trace contains 26,839,809,058 packets, comprising 7,893,035,860 TCP connections. Later in the paper we examine a day-long trace from August 31, 2004. The second trace records all packets sent via the commercial Internet link between non-university clients and the campus Web server. There are 361,420 connections and 14,393,799 packets in this trace. This trace is used to examine causes of the global trend. The passive measurement results appear in Section 4.1.

3.2 Active Measurements

The passive measurements identified a large number of reset TCP connections on the campus network. Further investigation (described in Section 4.1) identified that many of these reset connections were for Web transactions. Because of this anomaly, we conducted isolated tests with several different Web browsers, Web servers, and operating systems in an attempt to discern the sources of the TCP resets.

These tests used active measurement techniques to initiate HTTP/TCP connections from a client to a server. The *tcpdump* software was run simultaneously on the network to record packet traces of the resulting activity.

Table 1 summarizes the software configurations used in our manual testing of Web clients and Web servers. According to a Netcraft survey [5] in October 2004, the four most prevalent Web servers on the Internet are Apache (67.92%), Microsoft Internet Information Server (IIS) (21.09%), SunONE (3.04%), and Zeus (1.35%). All four of these are tested in our study.

On the client side, we initiate HTTP connections in two different ways: from a browser (e.g., Mozilla, Internet Explorer), and from the command line using telnet to port 80 on the server. The use of these two techniques allows us to assess application-layer HTTP/TCP behaviour separately from the TCP protocol stack embedded in the operating system kernel. We consider one version of Linux (RedHat 9.0), and one version of Windows (Windows 2000). The active measurement results appear in Section 4.2.

4 Results

4.1 Passive Measurements

Figure 1 provides a graphical representation of the TCP connection activity between the University of Calgary and the commercial Internet for the period spanning October 1, 2003 through September 30, 2004. The top

[1] `http://www.tcpdump.org/`

[2] http://www.icir.org/vern/bro-info.html

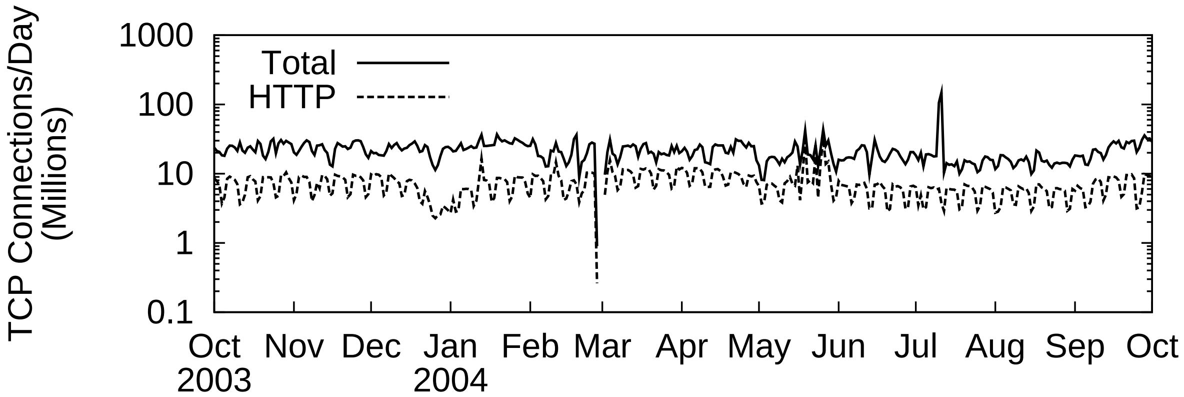

Figure 1: Total Daily TCP and HTTP Connection Activity (U. Calgary)

Table 2: Summary of Selected TCP Connection States Reported by *Bro*

State	Description
SF	Normal SYN handshake for initiation and FIN handshake for completion
REJ	Connection rejected. Initial SYN elicited a RST in reply.
RSTO	Connection reset by originator.
RSTR	Connection reset by responder.
S0	Initial SYN seen, but no reply.
S1	Connection established (SYNs exchanged), but nothing further seen.

line represents the total number of TCP connections each day, while the bottom line represents HTTP TCP connections (to or from well-known TCP port 80).

Several observations are evident from Figure 1. First, the number of TCP connections sampled in our study is on the order of tens of millions per day, large enough to provide reasonable estimates of the TCP connection states on which we focus. Second, a significant fraction of the TCP connections, approximately 35%, are for HTTP (Web) traffic. Third, the HTTP proportion of the total TCP traffic is relatively consistent across the one-year period studied. There are noticeable weekly patterns, and longer-term trends consistent with the academic calendar. These behaviours are very pronounced in the HTTP data, since most Web traffic is human-initiated. The patterns are less pronounced in the TCP data, since many TCP connections are machine-generated.

Figure 1 shows that there are three obvious anomalies in this data set. The first occurred February 27, 2004. A brief power outage corrupted the file system on our monitor, which had to be manually repaired. The second occurred in mid-May, and the third on July 10, 2004. Both of these involved machine-generated (local) events. We provide more details on these two events later in the paper.

The next step in our analysis was to classify each observed TCP connection based on the state reported by *Bro*. This state indicates whether the connection was "normal" or "abnormal". Among the abnormal states, *Bro* classifies each connection according to the packet patterns observed.

Table 2 provides a description of the most relevant TCP connection states. A "normal" TCP connection has a proper SYN handshake and FIN handshake (SF). Abnormal connections can take many possible states. The most common of these abnormal states are REJ, RSTO, and RSTR.

Figure 2 shows the results from classifying all of the TCP connections from our measurement data. The graph shows the daily percentages observed for each of the five most prevalent TCP connection states. The name of each state appears to the right of its associated band in the graph.

Figure 2 shows that the "normal" SF case (middle of the graph) is the most common state observed in the overall TCP traffic. However, the SF percentage is a lot lower than expected: it ranges between 20% and 60%, with an overall average of about 40%.

The S0 (SYN only) and REJ (reject) states combined are the next most common. These are the bottom two

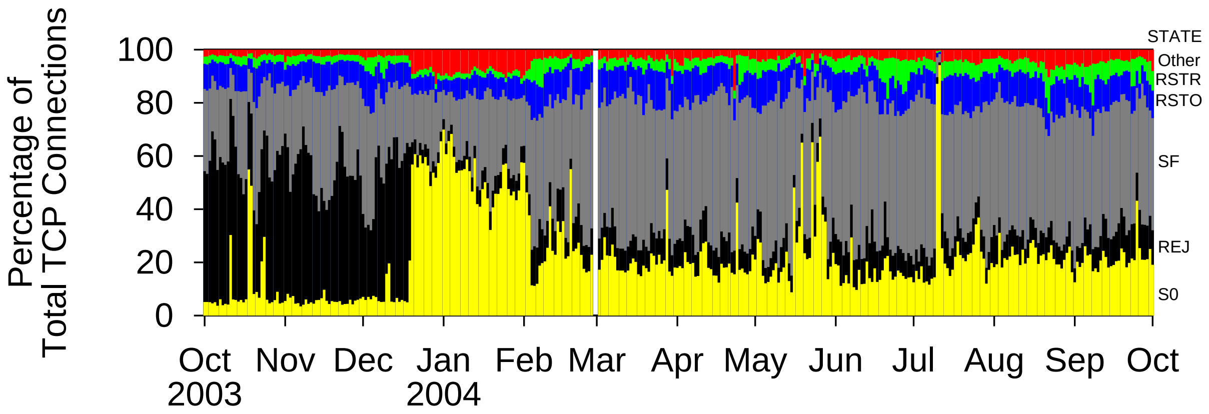

Figure 2: TCP Connection Status (All TCP Connections)

bands in Figure 2. These values are inflated due to several local events. Between October 1 and December 19, 2003, approximately 40% of connections were in the REJ state. A local service moved to a new server (with a new IP address), but numerous remote clients were either configured with the IP address rather than the fully qualified domain name, or had a stale entry in their DNS caches. As a result, these remote client machines continuously attempted to establish a TCP connection to a now closed port on the old server. On December 19, 2003 a router on the campus network was configured to drop these SYN packets before they reached the old server. As a result, Figure 2 shows a dramatic drop in the REJ connections, and a corresponding increase in the S0 connections (as our monitor still saw the incoming SYN packets).

In mid May and again on July 10, there were significant spikes in the percentage of TCP connections in the S0 state. The spikes in mid May were caused by a single client trying to reach a non-existent Web server. The rate at which these connection requests were issued indicates that the underlying HTTP requests were not human-generated. The large spike in July was an attempted scan of a single machine.

Even if these local events are ignored, there are still many connections with S0 or REJ states. We attribute much of this to hosts scanning other hosts or networks, and expect that this is a global trend. There are likely other local events hidden within Figure 2, since the percentage of S0 connections is currently much higher than it was a year ago. However, we defer a more detailed analysis of these states for future work.

The next two most common TCP states are RSTO and RSTR. These correspond to connections that actually transferred data between the client and server (in one or both directions), but that were then reset by either the client (RSTO) or the server (RSTR), rather than terminated with a FIN handshake. Figure 2 shows that approximately 15% of TCP connections fall into one of these two categories. These percentages are relatively stable throughout the trace, suggesting this is likely a global trend.

Figure 3 provides a further breakdown of the data, by focusing on the TCP states observed for HTTP TCP connections (a subset of the total TCP activity). The normal SF state is the most prevalent, accounting for approximately 70% of all HTTP connections throughout the year. The percentages are quite stable, with the exception of the anomaly in May mentioned earlier. Figure 3 shows that the local anomaly persisted much longer than was evident in Figure 2. With the exception of this anomaly, the proportion of REJ and S0 states has dropped significantly, suggesting that most of the failed connection attempts in the overall TCP traffic are not HTTP-related.

The main observation in Figure 3 is that the proportion of TCP RST states has increased significantly, particularly for resets by the originating client. Across the entire duration of the trace, the RSTO state accounted for about 22% of the HTTP TCP traffic, while the RSTR state averaged around 3%. These behaviours occur consistently, on a daily basis, suggesting that they are a stable part of the Internet traffic.

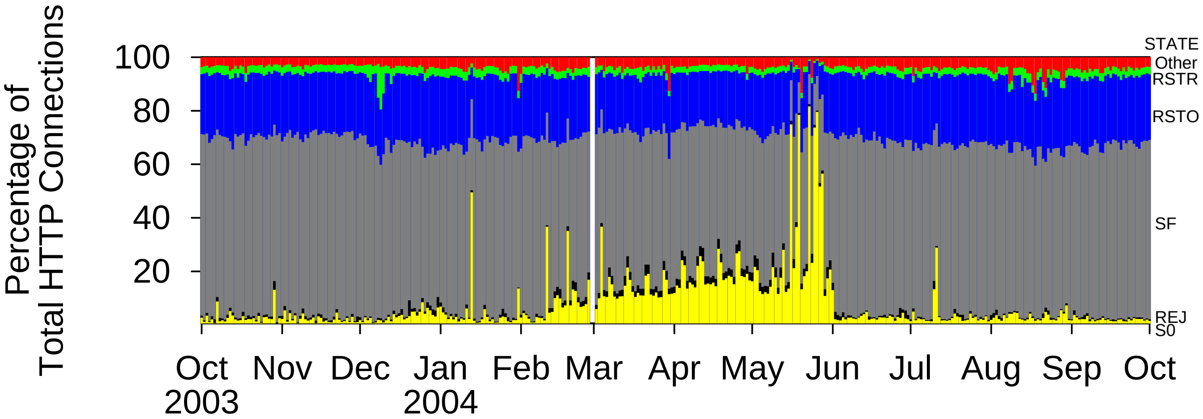

Figure 3: TCP Connection Status (HTTP TCP Connections)

4.2 Active Measurements

In an attempt to pinpoint the sources of the TCP reset behaviours, we manually conducted isolated experiments with selected Web client and Web browser software. Using *tcpdump* packet traces, we then classified the connection state that resulted from each test scenario.

The first set of tests involved remote Web servers hosting commercial Web sites. In particular, we chose the Apache, Microsoft, Zeus, and Sun Web sites, since these sites all run a recent version of their corresponding Web server software. Each of these sites supports persistent connections. We expect that each server has a management procedure to close idle TCP connections. The most common management approaches are likely timeout-based (e.g., close a connection if it has been idle for more than N seconds) and threshold-based (e.g., if more than X persistent connections are open, then close the connection that has been idle the longest).

To study the TCP connection behaviour, we telnet to port 80 on each server and issue an HTTP request, and keep the connection open. From *tcpdump*, we can see how the server closes the connection. Note that we are only interested in the TCP behaviour when the connection is closed, not in determining the management policy (e.g., timeout, threshold) in use by the server. (The aggressiveness of connection closing likely depends on Web workload and TCP resource usage, both at the server and the client.) We also initiate requests from common Web browser platforms to see if there are any differences in TCP behaviour.

Table 3 summarizes the results from these active measurement tests. The first column indicates the technique used to initiate client connections, along with additional relevant information about the human client behaviour. All tests used HTTP/1.1, with persistent connections.

The results in Table 3 illustrate two main points. First, among the four different Web servers tested, three used the proper FIN handshake to terminate an idle persistent connection, while one (Microsoft IIS) used a TCP RST. This type of server behaviour is likely responsible for many of the responder RSTs (RSTR) observed in the campus TCP traffic. This feature of IIS is used to reduce the number of connections that enter the TIME_WAIT state on the server [6].

Second, one of the client browsers tested (Microsoft Internet Explorer on Windows 2000) used TCP RST to terminate connections. The Internet Explorer (IE) browser on Windows 2000 consistently used TCP RSTs to close persistent connections, regardless of the server with which it was communicating. Since telnet on Windows 2000 also used the proper FIN handshake to close TCP connections, we expect that the version of IE 6.0 we are using on Windows 2000 is shutting down the connection in a non-graceful manner [3].

Given the widespread use of Internet Explorer, and its apparent non-standard use of TCP for some versions (we previously observed similar behaviour with IE 5.5 on Windows 2000), we hypothesize that it accounts for the bulk of the RSTO states in our traffic measurements.

A second set of tests was conducted with a local Web client and a local Web server, to demonstrate that the foregoing behaviours are repeatable. By using a local Web server (Apache 1.3.27), we can control the server

[3] `http://msdn.microsoft.com/library/en-us/winsock/winsock/shutdown_2.asp`

Table 3: Summary of Active Measurement Test Scenarios

Client	Server	State	Comments (times are approximate)
telnet to port 80, issue “HEAD / HTTP/1.1 Host: <server>” and wait (RedHat Linux 9.0)	Apache	SF	Server initiates FIN handshake after 5-second timeout.
	IIS	RSTR	Server closes with TCP RST after 1 minute.
	Zeus	SF	Server initiates FIN handshake after 10 seconds.
	SunONE	SF	Server closes with FIN handshake after 1 minute.
telnet to port 80, issue “HEAD / HTTP/1.1 Host: <server>” and wait (Windows 2000)	Apache	SF	Server initiates FIN handshake after 5-second timeout.
	IIS	RSTR	Server closes with TCP RST after 1 minute.
	Zeus	SF	Server initiates FIN handshake after 10 seconds.
	SunONE	SF	Server initiates FIN handshake after 2 minutes.
telnet to port 80, issue “HEAD / HTTP/1.1 Host: <server>” and wait (Windows XP)	Apache	SF	Server initiates FIN handshake after 5-second timeout.
	IIS	RSTR	Server closes with TCP RST after 1 minutes.
	Zeus	SF	Server initiates FIN handshake after 10 seconds.
	SunONE	SF	Server initiates FIN handshake after 1 minute.
Mozilla 5.0 browser (RedHat Linux 9.0)	Apache	SF	Server initiates FIN handshake after 5-second timeout.
	IIS	RSTR	Server closes with TCP RST after 1 minute.
	Zeus	SF	Server initiates FIN handshake after 10 seconds.
	SunONE	SF	Server initiates FIN handshake after 2 minutes.
Microsoft Internet Explorer 6.0 (Windows 2000)	Apache	RSTO	Browser closes persistent connection with TCP RST.
	IIS	RSTO	Browser closes persistent connection with TCP RST.
	Zeus	RSTO	Browser closes persistent connection with TCP RST.
	SunONE	RSTO	Browser closes persistent connection with TCP RST.

configuration, including the server mechanisms (e.g., timeout setting) for closing persistent connections.

The results from the local tests were consistent with those observed with the remote Web servers. In addition, the local tests showed that IE closes idle connections after 60 seconds, if resource demands do not require reclaiming connections sooner. In all tests, Mozilla closes TCP connections properly (SF), as did telnet on Linux and Windows.

In the tests reported here, the browsers do not have a large number of persistent connections open at one time. In actual usage, browsers may need to close persistent connections more aggressively (e.g., if a client visits a new site different from what they are currently viewing). We have observed cases where IE does in fact close connections more aggressively. Again, it used TCP RSTs rather than FINs. IE also used RSTs to close connections that were still open when the user exited the browser application. Mozilla used FINs to handle this case.

4.3 Browser-Level Analysis

To close our measurement study, we present in Table 4 a more detailed breakdown by browser type of the TCP connection states observed for clients visiting our campus web server. We use a trace involving a single server so that we can focus on the behaviour of the browsers.

We used *Bro* to analyze the relationship between the browser used (indicated by the HTTP `User-Agent` request header) and the TCP connection state.

Table 4 shows the ten most common user-agents in the trace. These agents account for 81.5% of all connections in the trace. The most common browser seen was IE 6.0 on Windows XP. This type of browser established 39.8% of all the TCP connections in the trace. Among this subset of connections, 36% ended with the proper FIN handshake, while 60% were reset by the client.

Table 4 shows that all of the IE browsers had roughly similar behaviour. The single Gecko browser on the list, as well as the “Crawlers” group typically used the FIN handshake to terminate their connections. This suggests that there is not a problem with the server.

Table 4: Browser-Level Analysis of TCP Behaviour

User-Agent	OS	Connections (%)	SF (%)	RSTO (%)	RSTR (%)
IE 6.0	Windows XP	39.8	36.3	60.4	1.2
IE 6.0	Windows 2000	14.3	46.4	51.3	1.1
IE 6.0	Windows 98/ME	9.1	30.6	65.0	3.3
crawlers		8.2	98.4	1.4	0.0
IE 5.x	MacOS	2.3	5.2	93.3	0.5
IE 5.5	Windows 2000	1.8	54.0	43.4	1.3
IE 5.0	Windows 98/ME	1.7	47.8	43.6	6.4
IE 5.5	Windows 98/ME	1.6	38.9	55.5	4.6
IE 5.0	Windows 2000	1.4	53.9	39.4	2.7
Gecko 2004	Windows XP	1.3	85.7	13.3	0.5

The percentage of reset connections seen in this trace is higher than what we have seen in the overall network trace. We believe that this is primarily due to the aggressive closing of persistent connections by the server. The particular server that we monitored actively closed connections if they were idle for a mere two seconds. On connections using IE as the browser, the client typically responded with a RST, rather than completing the FIN handshake.

The aggressive closing of persistent connections is also responsible for some of the RSTR connections. The shorter the idle timeout, the greater the probability that another request from the client is in transit when the server sends its FIN packet to initiate the active closing of the connection. If a request arrives after the server has sent a FIN packet, the server issues a RST, resulting in a RSTR state for that connection.

One possible reason that a browser may reset a connection (other than to abort a transfer when a problem has occurred) is to reduce the number of TIME_WAIT states on the server [6]. However, it is in the client's best interest to ensure that a connection does not get reused too quickly (the role of TIME_WAIT). Specifically, when problems occur with TCP connections, the user's Web browsing experience is degraded. A better approach for reducing TIME_WAIT overhead on Web servers is for clients to be more proactive, judiciously closing server connections using the proper FIN handshake. The TIME_WAIT state is then maintained at the client, rather than the server.

There is another side effect of resetting TCP connections. We have observed situations where a client has received a complete response on a connection that it reset, and then seen the client immediately request the same file again on a different connection. This not only degrades the user's Web experience, but leads to redundant transfers across the Internet.

The overhead of managing TCP connections in a TIME_WAIT state can be significant [1, 3]. Thus it is understandable why some servers choose to RST connections that have been idle for a long period of time. However, we believe that this should only be done if absolutely necessary (e.g., if the server is actually busy and needs to reduce its overhead), and not as a general practise.

5 Conclusions

This paper describes the different causes of abnormal TCP connections. Our measurement results show that application-level browser behaviour is the primary contributor to the global trend of abnormal TCP behaviours. Specifically, irregularities in the implementation and management of persistent HTTP connections are the cause of this problem. Correcting the TCP behaviour of a popular Web browser would likely eliminate most of these TCP RST anomalies.

Our short-term future work involves creating more sophisticated *Bro* scripts to analyze the HTTP transactions more rigourously. Our longer-term plans include a longitudinal study of TCP behaviour, digging deeper into anomalies and trends that arise.

Acknowledgments

Financial support for this research was provided by iCORE (Informatics Circle of Research Excellence) in the Province of Alberta and by the Natural Sciences and Engineering Research Council (NSERC) of Canada. The authors are grateful to Dan Clark, David Jager, and Tom Seto from University of Calgary Information Technologies (UCIT) for providing access to the campus Internet traffic, to Rob Simmonds and Nayden Markatchev for their assistance with deploying and operating the network monitor, to Vern Paxson for his assistance with *Bro*, and to Venkat Padmanadhan for providing information about Internet Explorer and IIS.

References

[1] M. Aron and P. Druschel, "TCP Implementation Enhancements for Improving Webserver Performance", *Proceedings of IEEE Infocom*, March 1999.

[2] P. Barford and M. Crovella, "Critical Path Analysis of TCP Transactions", *Proceedings of ACM SIGCOMM*, September 2000.

[3] T. Faber, J. Touch, and W. Yue, "The TIME-WAIT state in TCP and Its Effects on Busy Servers", *Proceedings of IEEE Infocom*, March 1999.

[4] C. Fraleigh et al., "Packet-Level Traffic Measurements from the Sprint IP Backbone", *IEEE Network*, 2003.

[5] Netcraft Web Server Survey,
http://news.netcraft.com/archives/web_server_survey.html

[6] V. Padmanabhan, personal communication (email), September 22, 2003.

[7] V. Paxson, personal communication (email), May 11, 2003.

[8] W. Stevens. *TCP/IP Illustrated, Volume 1*, Addison-Wesley, New York, 1994.

[9] K. Thompson, G. Miller, and R. Wilder, "Wide-area Internet Traffic Patterns and Characteristics", *IEEE Network*, Vol. 11, No. 6, pp. 10-23, November/December 1997.

[10] C. Williamson, "Internet Traffic Measurement", *IEEE Internet Computing*, Vol. 5, No. 6, pp. 70-74, November/December 2001.

Federal Communications Commission

The FCC Acknowledges Receipt of Comments From …

AT&T Inc.

…and Thank You for Your Comments

Your Confirmation Number is: '2008425835453 '

Date Received:	**Apr 25 2008**
Docket:	**07-52**
Number of Files Transmitted:	**1**

DISCLOSURE

This confirmation verifies that ECFS has received and accepted your filing. However, your filing will be rejected by ECFS if it contains macros, passwords, redlining, read-only formatting, a virus or automated links to source documents that is not included with your filing. Filers are encouraged to retrieve and view their filing within 24 hours of receipt of this confirmation. For any problems contact the Help Desk at 202-418-0193.

Initiate a Submission | Search ECFS | Return to ECFS Home Page

FCC Home Page | Search | Commissioners | Bureaus/Offices | Finding Info

updated 12/11/03