Identifying Performance Bottlenecks in CDNs through TCP-Level Monitoring

Peng Sun, Minlan Yu, Michael J. Freedman, and Jennifer Rexford
Dept. of Computer Science, Princeton University
Princeton, New Jersey, USA
{pengsun, minlanyu, mfreed, jrex}@cs.princeton.edu

ABSTRACT

Content distribution networks (CDNs) need to make decisions, such as server selection and routing, to improve performance for their clients. The performance may be limited by various factors such as packet loss in the network, a small receive buffer at the client, or constrained server CPU and disk resources. Conventional measurement techniques are not effective for distinguishing these performance problems: application-layer logs are too coarse-grained, while network-level traces are too expensive to collect all the time. We argue that passively monitoring the transport-level statistics in the server's network stack is a better approach.

This paper presents a tool for monitoring and analyzing TCP statistics, and an analysis of a CoralCDN node in PlanetLab for six weeks. Our analysis shows that more than 10% of connections are server-limited at least 40% of the time, and many connections are limited by the congestion window despite no packet loss. Still, we see that clients in 377 Autonomous Systems (ASes) experience persistent packet loss. By separating network congestion from other performance problems, our analysis provides a much more accurate view of the performance of the network paths than what is possible with server logs alone.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operation - *Network Monitoring*; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Measurement, Performance

Keywords

TCP, Content Distribution Network, Performance Bottleneck

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
W-MUST'11, August 19, 2011, Toronto, Ontario, Canada.
Copyright 2011 ACM 978-1-4503-0800-7/11/08 ...$10.00.

1. INTRODUCTION

Content distribution networks (CDNs) run replicated servers to deliver content to a large number of clients. To optimize data-transfer performance, CDNs need to pick the right server for each group of clients, and select routing paths that traverse the Internet quickly [15]. To make better decisions, CDNs should first identify the performance bottlenecks. CDNs can then optimize their software or upgrade their machines if the servers are limiting the performance, work with their ISPs if network performance is spotty, or notify clients if a small receive buffer is the bottleneck (e.g., through an embedded message in the Web page, or a modified HTTP header).

To diagnose performance problems, CDNs need to collect and analyze measurement data. However, conventional measurement techniques are either too coarse-grained for diagnosing problems, or too expensive for achieving good coverage.

Application-layer logs [5, 6], while useful for capturing download times and average throughput, are too coarse-grained to uncover *why* clients experience performance problems. Server logs cannot distinguish whether a connection is limited by network congestion, slow server writes, a small receive buffer, or a long round-trip time (RTT). This also makes it difficult to diagnose congestion by correlating performance across connections [13], because many connections are limited by other factors.

Network-layer packet traces [17, 18] are useful for measuring packet loss and inferring TCP connection state (e.g., slow start and receive window size), but they are too expensive to capture all the time. In addition, real-time inference of internal TCP state (e.g., number of bytes in the send buffer, whether a transfer is congestion-window limited, etc.) from in-network traces is challenging, especially if the variant of TCP running on each end point is not known in advance.

Active probing is effective for inferring the properties of network paths [14, 11, 15, 10]. However, these techniques rely on a representative set of nodes to launch probe traffic, and introduce extra load on the network. In addition, active probing does not provide direct visibility into bottlenecks on the server machines.

Instead of, or in addition to these techniques, we believe CDNs should capitalize on *transport*-layer statistics [16] readily available in the server network stack [12]. It is more direct to measure end-to-end performance in the server network stacks than to sniff packets in the network, and collecting transport-layer statistics involves less overhead than

packet traces and active probing. These transport-layer statistics directly reveal whether a connection is constrained by the server (e.g., too little data in the send buffer), the congestion window (e.g., a small window during slow start), network congestion (e.g., packet loss), or the client buffer (e.g., a small receive window). We built a measurement tool to collect and analyze the TCP statistics, and applied our tool to CoralCDN [8] running on PlanetLab. In addition to characterizing performance bottlenecks, we identified congested network paths by correlating packet losses across connections to clients in the same AS. This is more accurate than the analysis based on throughput (e.g., from server logs), since many connections are *not* limited by network conditions [9].

The remainder of the paper is structured as follows. We first discuss the design and implementation of our in-stack monitoring tool in Section 2. Then, Section 3 presents the results from applying our tool to CoralCDN. These specific results do not necessarily apply to other CDNs, since PlanetLab servers are resource constrained. Still, we believe our techniques are broadly useful for other CDNs and cloud services in understanding and improving user-perceived performance. In Section 4 we demonstrate how to identify congested network paths by correlating packet losses across connections to clients in the same AS. We conclude the paper in Section 5.

2. MEASUREMENT FRAMEWORK

In this section, we first describe how we collect in-stack information. Then we show how to classify the performance problems for each connection, and use two example traces to illustrate different classes of performance bottlenecks. Our tool further correlates across the network performance problems affecting clients in the same AS, as discussed in Section 4.

2.1 Measuring TCP Statistics

Our measurement tool polls the network stack to collect connection-level statistics via web100 tool [2]. Web100 is a kernel patch that extracts the already-existing TCP statistics, and offers an API to access their values in user space (An upgraded version, Web10G, is now available for the Linux mainline [3]). The TCP-level statistics we collect are shown in Table 1. They directly tell the performance problems of each connection, with less overhead than packet traces and more details than CDN logs.

The data include two types of statistics: (i) instantaneous snapshots (e.g., *Cwin*, the current size of the congestion window) and (ii) cumulative counters (e.g., *RwinLimitTime*, the total time the connection has been receive-window limited). Figure 1 shows two connections from our measurements of CoralCDN, and they illustrate how the TCP statistics evolve over the life of the connection. Furthermore, our tool is lightweight. It periodically polls those statistics variables every 50ms, and generates less than 200MB of data per server per day in our measurement of CoralCDN.

2.2 TCP Performance Classifier

With the TCP-level statistics, we then characterize the performance limitations for each connection. There are four categories of performance limitations: the network path, the server network stack, the clients, and the CDN server applications. Since we have direct access to the TCP stack, we can easily distinguish these performance limitations using the TCP-level statistics as summarized in Table 2.

Statistics	Definition
Cwin	Current congestion window
Rwin	Current receive window
BytesInFlight	# of bytes sent but not ACKed
BytesInSndBuf	# of bytes written but not ACKed
SmoothedRTT	Smoothed RTT computed by TCP
BytesWritten	Cumulative # of bytes written by app
BytesSent	Cumulative # of bytes sent
PktsRetrans	Cumulative # of pkts retransmitted
RwinLimitTime	Cumulative time that a connection is limited by receive window
CwinLimitTime	Cumulative time that a connection is limited by congestion window

Table 1: Key TCP Statistics in our Tool

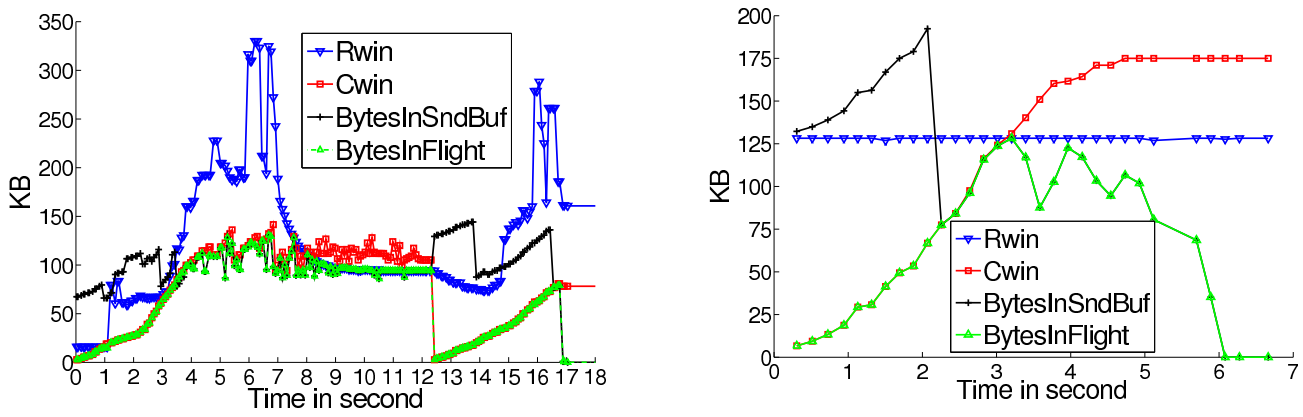
(i) Network Path: The network congestion can limit the performance of the connection by a small congestion window. Our tool detects congestion when (a) the connection is limited by congestion window and (b) a packet loss has occurred (i.e., the number of retransmissions has increased). When the connection is no longer limited by the congestion window or the congestion window size returns to its size before the loss, we mark the end of the impact of loss. In Figure 1(a), a packet loss happens at around 12 seconds, causing the decrease of the congestion window. The small congestion window recovers for the next five seconds. During this period, the key performance limitation is the congestion window size after packet loss.

(ii) Server Network Stack: The congestion window can limit performance even when no losses occur, particularly at the beginning of a connection (i.e., during slow start). Our tool detects this situation when (a) the congestion window limits performance, (b) no loss has occurred, and (c) the impact of a previous loss has ended. The period from 0s to 8s in Figure 1(a) and the period from 0s to 2s in Figure 1(b) show examples. During the time, the small congestion window size is the dominant performance limitation.

(iii) Clients: A small receive window size can also limit the performance. We detect this situation when the receive window is the bottleneck of the connection. The connection in Figure 1(a) is receive-window limited from 8s to 12s; the receive-window size drops greatly at 8s, limiting the data the server can send.

(iv) CDN Server Applications: The CDN server may limit the performance because a slow CPU or scarce disk resources constrains the amount of data written into the TCP send buffer. We classify a connection as application limited when the server has less data in the send buffer than the congestion and receive window sizes (i.e., the connection is neither congestion-window limited nor receive-window limited). Starting from about 3s in Figure 1(b), the application does not have enough data to send. The *BytesInSndBuf* (marker +) and *BytesInFlight* (marker $\triangle$, overlapping) are smaller than the congestion and receive window sizes.

In each 50ms interval, a connection may have one or more performance limitations. Therefore, we calculate the *fraction* of time the connection is limited by the network path,



(a) Network-path Limited

(b) Application Limited

Figure 1: Traces of Network-stack Statistics for Two Example TCP Connections (Some Lines are overlapping)

Limitations	Classification Method
Network Path	ΔCwinLimitTime > 0 and ΔPktsRetrans > 0
Server Network Stack	ΔCwinLimitTime > 0 and ΔPktsRetrans $= 0$ and Cwin $\geq$ Cwin_before_loss
Clients	ΔRwinLimitTime > 0
Server App.	BytesInSndBuf $<$ min{Cwin, Rwin}

Table 2: Performance Limitation Classifier

the server network stack, the clients, and the server applications. We also accumulate the number of lost packets.

3. STUDYING CORALCDN WITH OUR TOOL

In this section, we use our measurement tool to identify the performance bottlenecks for CoralCDN and discuss how CoralCDN can use these results to make better decisions on server and path selection.

3.1 TCP-level Monitoring for CoralCDN

CoralCDN is a popular open content distribution network, serving over one million clients for over 20 thousand domains per day [7]. CoralCDN consists of Coral DNS servers and Coral HTTP proxies. The clients accessing a website served by CoralCDN are transparently directed by CoralCDN's DNS server to a nearby Coral proxy.

We deployed our measurement tool on a Clemson PlanetLab node running a CoralCDN proxy. The Coral proxy configures specific ports to serve the clients, and they are exclusively used by CoralCDN. We use the port number to distinguish the traffic to CoralCDN from other connections. Since February 8, 2011, we have been collecting TCP statistics for all the connections belonging to CoralCDN. For the rest of the paper, we present the results from the data collected over one week of February 19–25, 2011; our analysis on other weeks show similar results. During the week, 209K connections accessed the CoralCDN server.

Note that, since CoralCDN is a proxy, it temporarily caches the web pages requested by clients. When a cache miss occurs, the connection will be held to wait for the proxy to perform a lookup in its distributed hash table (DHT) and fetch data from either another Coral proxy or the origin server. The waiting time is about 1 to 2 seconds, which greatly degrades performance. With our tool, we detect a cache miss when the send buffer has no data for several seconds at the beginning. Among the 209K connections in the week, 34.3% encountered a cache miss. Because the time to fetch data from the origin server dominates the performance, we exclude these connections from our analysis. The following analysis focuses on the remaining 137K connections, which are from 2008 ASes.

3.2 Characterizing Performance Bottlenecks

We first study the frequency of different performance limitations in order to identify the factors that influence data-transfer performance the most. For each connection, at each interval, we classify the performance limitations based on Table 2. We then calculate the fraction of time that a connection is limited by each performance bottleneck. Table 3 shows the percentage of connections that have different percentage of performance limitations, leading the following conclusions:

(1) In a significant fraction of performance problems, the CoralCDN proxy is limited by the application or servers, not network conditions. A large fraction of connections are limited by the *server application*. More than 10% of connections are application-limited for more than 40% of their lifetime. The limitations can come from the web service softwares, or the slow CPU and scarce disk resources of the PlanetLab node.

(2) The performance of many connections is limited by the congestion window despite having no packet loss. Nearly 20% of connections spend at least 40% of their lifetime constrained by a small congestion window, despite no packet loss. Many of these connections are short-lived, spending most of their time in TCP's slow-start phase. In CoralCDN, more than 80% of the connections last less than 1 second. For the many small data transfers, performance is largely a function of how TCP increases the congestion window and the RTT between the client and

	% of Connections with Problems for > X% of Duration					
Limitations	**>0**	**>20%**	**>40%**	**>60%**	**>80%**	**>99%**
Server Application	19.76%	15.32%	10.75%	4.98%	2.86%	0.004%
Server Network Stack	52.47%	28.59%	18.72%	10.21%	5.30%	3.99%
Network Path	6.20%	5.30%	3.94%	2.77%	1.68%	0.27%
Clients	5.77%	2.09%	1.27%	0.60%	0.19%	0.03%

Table 3: Percentage of Connections that have Different TCP Performance Problems

the server. As such, directing clients to the *closest* Coral proxy (in terms of RTT) is critically important for improving performance. In addition, the network stack can use a larger initial congestion window, as some commercial content providers do [4].

(3) Some connections have significant packet losses. The network path problem degrades around 6% of the connections. As shown in Table 4, 1.26% of the connections experienced a packet loss rate higher than 20%. Information about packet loss can drive the CDN's decisions for server selection and path selection, in the hope of delivering content to clients over less congested paths. We explore this topic in greater detail in Section 4.

(4) A small fraction of clients are severely limited by the receive window. Around 0.6% of the connections (i.e., 0.8K connections) are severely affected by the receive-window size at the clients. Most of these clients only have a receive window of less than 30KB. The Web service providers could notify these clients of the problem, through a banner or embedded message in the web page, or a hint to the browser in the HTTP response header. For these clients, reconfiguring the receive-buffer size could improve performance dramatically.

4. CORRELATING NETWORK PROBLEMS ACROSS CLIENTS

In this section, we explore how to use our measurements to identify network paths with performance problems. We first illustrate the importance of identifying connections limited by network congestion rather than other factors, such as the server application or the client receive window. Next we show that correlating packet loss across connections can identify Autonomous Systems (ASes) affected by network performance problems. Then, we show that network performance problems persist over time, allowing measurements from one time period to guide a CDN's future decisions about server and path selection.

4.1 More Accurate View of Network Paths

CDNs rely on accurate estimation of network conditions (e.g., throughput, delay, and packet loss) for load balancing and path selection. CDNs could easily determine the average throughput of each connection from coarse-grained server logs, and identify ASes affected by low throughput from the CDN server to the clients. Based on these results, the CDN could change its server-selection and routing policies to circumvent the network performance problems.

However, drawing conclusions from throughput measurements alone would be misleading, since some connections are limited by non-network factors, such as the limited CPU/disk resources at the server, a small congestion window during slow start, or a small receive window in the client. In contrast, our tool can directly see the network conditions at transport layer, and separate network congestion (i.e., packet loss) from other factors to produce more accurate estimates of path performance. To carry out an AS-level analysis, we map each client's IP address to an AS, based on the ASes originating IP prefixes into BGP as seen in RouteViews [1]. Then we correlate across the connections in the same AS to analyze the path performance.

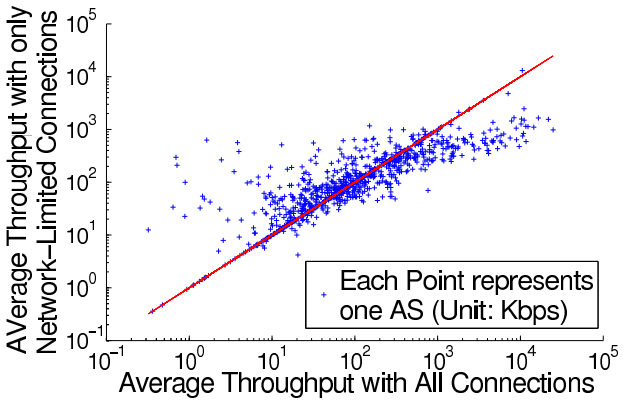


Figure 2: Effect of Focusing on Network-limited Connections on Throughput

To illustrate the risks of focusing on average throughput alone, we compare the average throughput for *all* connections to the performance of network-limited connections. The scatter plot in Figure 2 plots a single point for each AS, where the x-axis is the average throughput for all connections and the y-axis only considers the connections that are network-limited. The solid line shows where the two throughputs are equal. In the left part of the curve, many ASes have very low average throughput but relatively few of these connections are limited by the network—in fact, the average throughput of network-limited connections is much higher. Focusing on low average throughput alone may wrongly label some ASes as experiencing bad network performance when they are actually bottlenecked by other factors.

4.2 Correlating Packet Loss by AS

With more fine-grained performance data, we can analyze network performance directly based on the packet-loss statistics for each AS. Figure 3 plots the fraction of ASes with a large fraction of connections experiencing a high packet loss rate (e.g., 5%, 10%, or 20% loss). For 7.1% of the ASes (143 ASes), more than half of the connections have loss rates in excess of 10%. CoralCDN could conceivably adjust

% of Connections with Packet Loss Rate > X%									
>0	>1%	>5%	>8%	>10%	>15%	>20%	>30%	>40%	>50%
6.29%	5.44%	3.80%	3.06%	2.67%	1.91%	1.26%	0.60%	0.33%	0.19%

Table 4: Percentage of Connections that have Different Levels of Packet Loss

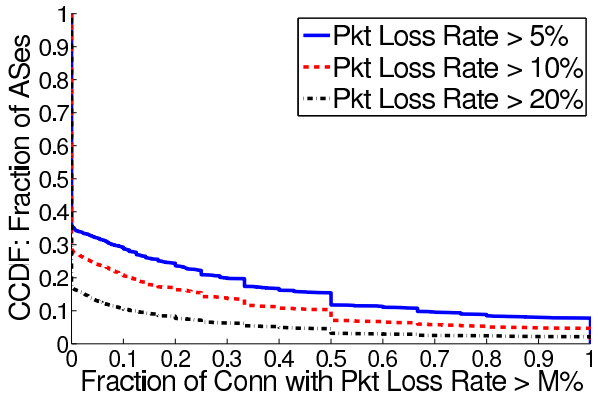


Figure 3: Y Fraction of ASes that have at least X Fraction of Conn. with Pkt Loss Rate >M%

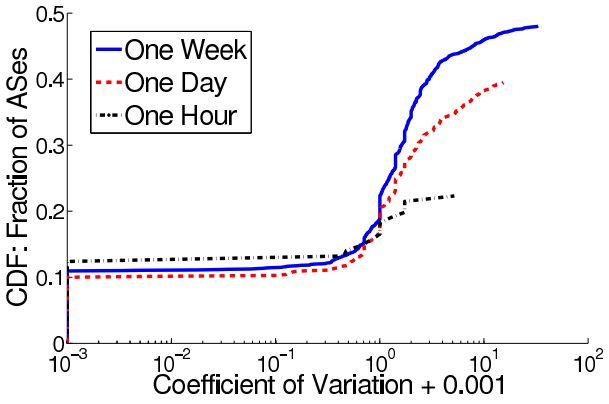


Figure 4: CV of Pkt Loss Rate for ASes

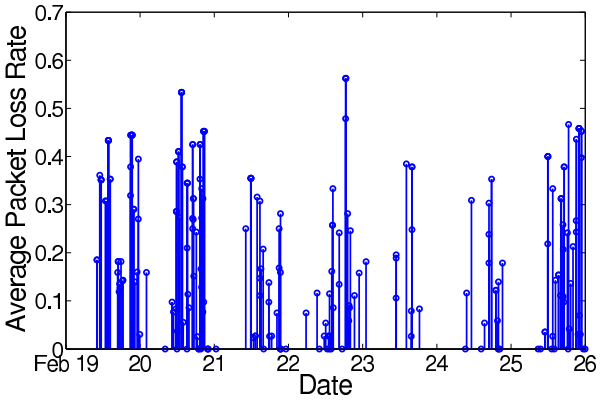


Figure 5: Example AS with Persistent Loss Problem

the server or path selection to improve the network paths to these ASes in order to reduce their loss rates.

4.3 Stability of Packet Loss Over Time

CDNs cannot make decisions based on network performance unless the measurements have predictive value. We calculate the *coefficient of variation (CV)* for each AS to characterize the persistency of packet losses in three steps: (i) We aggregate all the active connections during an *aggregation interval*, and calculate the average packet loss rate among these connections for the aggregation interval. In our study, we use 60 seconds as the aggregation interval[1], and exclude those intervals during which no packets are sent. (ii) With the average packet loss rate for each aggregation interval for each AS, we then explore the variance of packet loss rates over a period of time called an *analysis window*. A longer analysis window indicates a coarser grained packet loss behavior (e.g., during a week, a few clients access CoralCDN each day and experience packet losses every access), while a shorter analysis window shows users' behaviors at the micro level (e.g., in an hour a few clients continuously access CoralCDN and all have packet losses). (iii) In the analysis window, we calculate the *coefficient of variation (CV)*, which is defined as the ratio of the standard deviation to the mean of the series of average packet loss rates. A distribution with $CV < 1$ is considered low-variance. In our scenario, $CV < 1$ means that the average packet loss rate for the AS is stable, and the loss condition is considered persistent.

Figure 4 shows the fraction of ASes that experience persistent loss problems on different time scales[2]. For the analysis window of one week, 18.8% of ASes (377 ASes) have persistent packet loss ($CV < 1$). For a randomly picked day (Feb 21 2011), the ratio of ASes with persistent loss problem is 122 out of 739, while the ratio for one hour analysis (Feb 21 2011 18:00–19:00) is 19 out of 121. Our tool can provide with good prediction of the packet-loss statistics for these ASes. For example, our tool identified AS 33774 which repeatedly experienced packet loss for a total of 436 connections during the whole week (Figure 5). Based on the information, CoralCDN can infer a bad network path between the edge server and the AS.

For future work, we will correlate across AS-level paths from BGP data to distinguish the AS hops that have network problems to guide better server and path selection decisions. We will also identify those clients whose performance problems come from the "last hop" (i.e., the local access

[1]We pick 60 seconds because there are only a few active connections with a smaller aggregation interval (e.g., only one to three connections within 10 seconds), and larger intervals show similar results.

[2]We add 0.001 to all CV value to include the condition of $CV = 0$ in the log figure. When no loss happens in the AS, its CV is $+\infty$

networks), since changes in server and path selection cannot significantly improve performance for these clients.

5. CONCLUSION

Our work uses the TCP-level statistics in the server network stack to help CDNs detect and diagnose performance problems. We deployed our measurement tool in CoralCDN, and have successfully identified the major performance limitation for the connections using CoralCDN service. By only looking at packet loss rates, we estimate the network conditions with higher accuracy. Our tool can also benefit the users if the CDNs provide feedback based on our measurement results (e.g., through embedded messages in the web page). The feedback can tell whether the user's performance is limited by the service, the network, or the client network stack.

In future work, we plan to explore better ways to pinpoint network performance bottlenecks, as well as techniques for diagnosing performance problems for virtual machines running on shared computing platforms in the cloud. We are also exploring how to combine the TCP-level statistics with application logs in order to pinpoint the performance bugs in online services.

6. REFERENCES

[1] Route Views Project. `http://www.routeviews.org/`.

[2] Web100 Project. `http://www.web100.org/`.

[3] Web10G Project. `http://web10g.org/`.

[4] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An Argument for Increasing TCP's Initial Congestion Window. *ACM SIGCOMM Computer Communication Review*, 40, 2010.

[5] R. Fonseca, M. J. Freedman, and G. Porter. Experiences with Tracing Causality in Networked Services. In *USENIX INM/WREN*, San Jose, CA USA, April 2010.

[6] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-Trace: A Pervasive Network Tracing Framework. In *USENIX NSDI*, Cambridge, MA USA, April 2007.

[7] M. J. Freedman. Experiences with CoralCDN: A Five-Year Operational View. In *USENIX NSDI*, San Jose, CA USA, April 2010.

[8] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing Content Publication with Coral. In *USENIX NSDI*, San Francisco, CA USA, March 2004.

[9] A. Gerber, J. Pang, O. Spatscheck, and S. Venkataraman. Speed Testing without Speed Tests: Estimating Achievable Download Speed from Passive Measurements. In *ACM IMC*, Melbourne, Australia, November 2010.

[10] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. Moving Beyond End-to-End Path Information to Optimize CDN Performance. In *ACM IMC*, Chicago, IL USA, November 2009.

[11] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *USENIX OSDI*, Seattle, WA USA, November 2006.

[12] M. Mathis, J. Heffner, and R. Raghunarayan. RFC 4898: TCP Extended Statistics MIB. `http://www.ietf.org/rfc/rfc4898.txt`, May 2007.

[13] V. N. Padmanabhan, L. Qiu, and H. J. Wang. Server-based inference of Internet link lossiness. In *IEEE INFOCOM*, San Francisco, CA USA, March 2003.

[14] R. S. Prasad, M. Murray, C. Dovrolis, and k. claffy. Bandwidth estimation: Metrics, measurement techniques, and tools. In *IEEE Network*, November/December 2003.

[15] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting Behind Akamai. In *ACM SIGCOMM*, Pisa, Italy, September 2006.

[16] M. Yu, A. Greenberg, D. Maltz, J. Rexford, L. Yuan, S. Kandula, and C. Kim. Profiling Network Performance for Multi-tier Data Center Applications. In *USENIX NSDI*, Boston, MA USA, March 2011.

[17] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services. In *USENIX OSDI*, San Francisco, CA USA, December 2004.

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