The following paper was originally published in the
Proceedings of the USENIX Annual Technical Conference (NO 98)
New Orleans, Louisiana, June 1998

Increasing Effective Link Bandwidth by Suppressing Replicated Data

Jonathan Santos and David Wetherall
Massachusetts Institute of Technology

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

Increasing Effective Link Bandwidth by Suppressing Replicated Data *

Jonathan Santos †
David Wetherall ‡
Software Devices and Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
http://www.sds.lcs.mit.edu/

Abstract

In the Internet today, transfer rates are often limited by the bandwidth of a bottleneck link rather than the computing power available at the ends of the links. To address this problem, we have utilized inexpensive commodity hardware to design a novel link layer caching and compression scheme that reduces bandwidth consumption. Our scheme is motivated by the prevalence of repeated transfers of the same information, as may occur due to HTTP, FTP, and DNS traffic. Unlike existing link compression schemes, it is able to detect and use the long-range correlation of repeated transfers. It also complements application-level systems that reduce bandwidth usage, e.g., Web caches, by providing additional protection at a lower level, as well as an alternative in situations where application-level cache deployment is not practical or economic.

We make three contributions in this paper. First, to motivate our scheme we show by packet trace analysis that there is significant replication of data at the packet level, mainly due to Web traffic. Second, we present an innovative link compression protocol well-suited to traffic with such long-range correlation. Third, we demonstrate by experimentation that the availability of inexpensive memory and general-purpose processors in PCs makes our protocol practical and useful at rates exceeding T3 (45 Mbps).

*This work was supported by DARPA, monitored by the Office of Naval Research under contract No. N66001-96-C-8522.

†Email: jrsantos@mit.edu
‡Email: djw@lcs.mit.edu

1 Introduction

In the Internet today, transfer rates are often limited by the bandwidth of a bottleneck link rather than the computing power available at the ends of the links. For example, access links (modem, ISDN, T1, T3) restrict bandwidth due to cost, while wireless links restrict bandwidth due to properties of the media. A traditional solution to this problem is the use of data compression, either at the link or application level. Existing compression schemes, however, tend to miss the redundancy of multiple instances of the same information being transferred between different clients and servers. This is problematic because such transfers have become prevalent with the growth of information services such as the Web.

Danzig's 1993 study of Internet traffic [3] noted that half of the FTP transfers could be eliminated with a caching architecture that suppressed multiple transfers of the same information across the same link. Since that time, protocols and traffic patterns have changed with the growth of the Web – it is now HTTP, not FTP, that is dominant. However, the level of redundancy is still perceived to be high, despite the application-level caching mechanisms that have emerged to curtail it.

In this paper, we revisit the problem of improving effective link bandwidth in the context of traffic with replicated data, as may occur due to TCP retransmissions, application-level multicast, DNS queries, repeated Web and FTP transfers, and so on. We have designed an innovative link compression scheme that uses a network-based cache to detect and remove redundancy at the packet level. Our

scheme takes advantage of the availability of inexpensive memory and general-purpose processors to provide an economical means of purchasing additional bandwidth. That is, given the one-time costs of $5000 per PC and the monthly costs of $2500 per T1 (1.5 Mbps), it is cheaper to purchase the two PCs used by the scheme than the bandwidth they are expected to save.

Our scheme has several interesting properties:

- It is independent of the format of packet data contents and so provides benefits even when application objects have been previously compressed, e.g., for Web images already in JPEG or GIF format.
- It utilizes a source of correlation that is not available at individual clients and servers and is not found by existing link compression schemes.
- It provides the bandwidth reduction benefits of caching in a transparent manner, e.g., there is no risk of stale information or loss of endpoint control.
- It constructs names at the link level using fingerprints and so does not depend on higher level protocol names or details. For example, the same information identified by different URLs will be compressed by our scheme, but not by Web caches.

Our scheme overlaps application-level caching systems – most notably Web caches – in that both reduce the impact of repeated transfers of the same information. However, our scheme is intended to complement Web caches rather than to compete with them, since it addresses a slightly different goal and works at a different level. For example, Web caches do not take advantage of replication across multiple caching systems, protocols and application objects.

In this paper, we present: a trace-driven traffic analysis that motivates our scheme; the design of our system; and an experimental characterization of a prototype implementation. Our traffic analysis in Section 2 uses several traces of at least one million packets each that we recorded between our site (the MIT Laboratory for Computer Science, including the Web consortium) and the rest of the Internet. In Section 3, we describe the system architecture and compression protocol, along with a prototype implementation running under Linux. In Section 4, we evaluate the performance of this prototype. We then contrast our system with related work and conclude in Sections 5 and 6, respectively.

2 Analysis of Replicated Traffic

To understand the potential of a system for suppressing replicated data transfers at the packet level, we began our design by analyzing network traffic. We define a packet to be *replicated* when the contents of its payload match exactly the contents of a previously observed payload. Since packet headers are expected to be constantly changing and a function of the source and destination hosts rather than the data being transported, we do not consider them in our search for replicated data.

Note that it is not clear that overlapping Web transfers will translate into replication that satisfies our definition and that may be detected and removed at the packet level. First, data sent multiple times must be parceled into packet payloads in the same manner, despite potentially different protocol headers, path maximum transmission units (MTUs), and protocol implementations. Second, the timescale of replication (which may be hours for Web documents) must be observable with a limited amount of storage. We therefore characterize the replication as defined above by answering the following questions:

- How much data is replicated?
- What kind of data is most likely to be replicated?
- What is the temporal distribution of replicated data?

2.1 Obtaining the Packet Traces

As input to our analysis, we collected a series of full packet traces of all traffic exchanged between our site and the rest of the Internet. New traces (rather than publicly available archives) were necessary because we require the entire packet contents in order to detect repeated data. The choice of our site was expedient, but it makes an interesting test case because it is a diverse environment hosting many

	All Inbound		Inbound HTTP		All Outbound		Outbound HTTP	
Set	Total Vol. (MB)	% Repl.	Total Vol. (MB)	% Repl.	Total Vol. (MB)	% Repl.	Total Vol. (MB)	% Repl.
A	277	12	26	19	554	18	267	24
B	189	2	13	8	563	21	384	28
C	105	2	3	10	294	21	239	24
D	237	11	22	7	606	19	420	25
E	217	4	28	8	594	23	427	29
Total	1025	7	91	11	2610	20	1736	26

Table 1: Total volume and replicated percentage (by volume) of inbound and outbound traffic

clients and servers. It includes the Web Consortium, MIT Laboratory for Computer Science and the MIT AI Laboratory.

Each trace was captured using `tcpdump` as a passive monitor listening to Ethernet traffic traveling on the segment between the Lab and the Internet. Five sets of 1-2 million packets each were gathered at different times of day, corresponding to approximately 2.6 GB of raw data in total. No packet capture loss was detected.

2.2 Analysis Procedure

We statically analyzed each trace by searching through the packets sequentially for replicated data. To expose the application data, we progressively stripped protocol headers up to the TCP/UDP level. For example, TCP payloads were identified by removing first the Ethernet, then IP and finally TCP headers. Our analysis therefore slightly underestimates the amount of replicated data due to changing headers at higher protocol layers that could not easily be taken into account; one example of traffic that falls into this category is DNS responses.

2.3 Replication by Traffic Type

Our initial analyses classified replication by traffic direction (incoming and outgoing) and type (TCP, UDP, other IP, and other Ethernet). It quickly became evident that most replication occurred in outgoing TCP data on ports 80 and 8001, i.e., Web traffic responding to queries from other sites. To highlight this, we separately classified TCP port 80 and 8001 traffic as HTTP traffic.

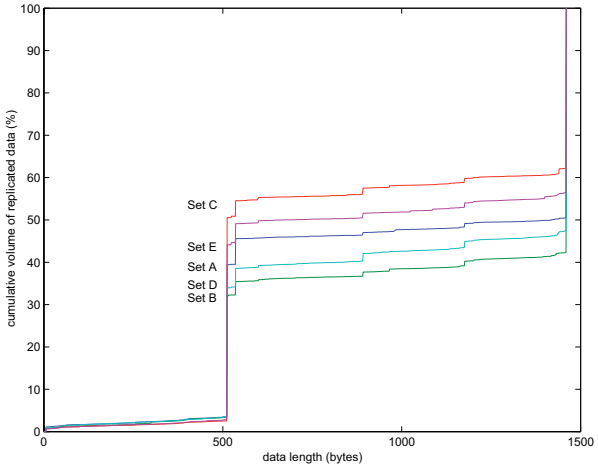

Figure 1: Cumulative volume of replicated data by packet length

Table 1 summarize the amount of replicated data that was found in each packet trace, for inbound and outbound traffic, respectively. The left-hand columns show the results for all types of traffic in each trace, while the right-hand columns summarize the replication in only the HTTP traffic for each trace.

These results support our intuition that there are significant amounts of replicated data present in the traces. Further, most of the traffic, as well as a greater percentage of replication, exists in the outbound traffic. Therefore, for the remainder of this paper, we will focus on the outbound traffic over the link.

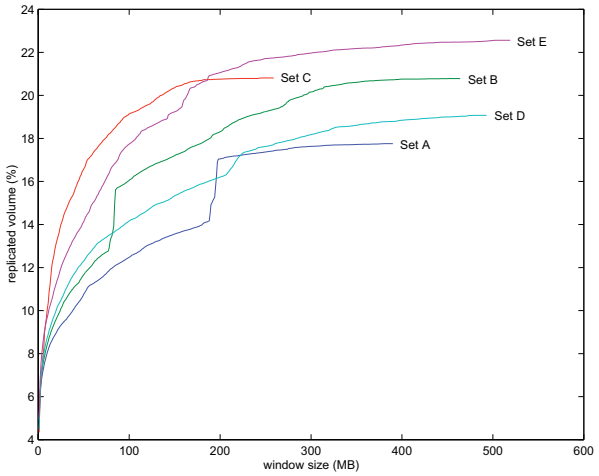

Figure 2: Percent of outbound traffic versus window size

2.4 Replication by Packet Size

A further criterion that is important to our scheme is packet size. Replication in large packets will result in a more effective system than replication in small packets when fixed-length packet overheads and packet processing costs are taken into account.

To assess this effect, we classified the replicated data according to the length of the data payload. Figure 1 depicts the cumulative volume of replicated data according to packet length. The sharp increases round 500 and 1500 bytes correspond to the default TCP segment size is 536 bytes and the maximum Ethernet payload 1.5 Kb. It is apparent that 97% of the volume of replicated data occurs in packets with a length greater than 500 bytes. This suggests that small per packet space costs required for compression will not result in a significant system overhead.

2.5 Distribution of Replication

Finally, the timescale of replication events determines the size of the packet cache needed to observe and remove such redundancy. To quantify this effect, we determined the interval, in bytes of data, from each match to the previous copy of the match. These intervals were then grouped to compute the percentage of the replicated traffic that could be identified as a function of window size.

Figure 2 shows this result for all outbound traffic. The positive result that we infer is that the majority of replicated data can be observed with a cache of 200 MB, i.e., reasonable results can be expected if we cache the data in the amount of RAM that is presently available in PCs.

3 Design and Implementation

We now describe the design and implementation of a compression architecture that suppresses replicated data based on the analysis from Section 2. The overall goal of our scheme is simply to transmit repeated data as a short dictionary token, using caches of recently seen data at both ends of the link to maintain the dictionary and encode and decode these tokens.

The correct operation of this scheme as a distributed system is complicated by the fact that messages may be lost by the channel. Our design must resolve the following issues:

- How are dictionary tokens generated?
- How are dictionaries at either end of the link maintained in a (nearly) synchronized state?
- How are (inevitable) differences in dictionary state handled?

Our approach is based on the insight that the fingerprint of a data segment is an inexpensive name for the data itself, both in terms of space and time. We are aware of the use of fingerprints for identification and version control in various systems, e.g., Java RMI/OS, but to the best of our knowledge this is the first time that fingerprints have been applied for this purpose at the network layer.

We selected the MD5 hash [12] for our implementation because it is 128 bits and may be calculated in one rapid traversal of the data; on a PentiumII (233MHz) the computational rate of fingerprinting exceeds 200 Mbps. Further, given that the hash is large enough and collisions rare enough, it is effectively a unique name for the data. For example, though our architecture handles collisions, none were detected in our trace data analysis.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.