Computer Sciences Department

Understanding and Exploiting Network Traffic Redundancy

Archit Gupta
Aditya Akella
Srinivasan Seshan
Scott Shenker
Jia Wang

Technical Report #1592

March 2007

UNIVERSITY OF
WISCONSIN
MADISON

Understanding and Exploiting Network Traffic Redundancy

Archit Gupta*, Aditya Akella*, Srinivasan Seshan**, Scott Shenker† and Jia Wang††
University of Wisconsin-Madison*, CMU**, UC Berkeley†, AT&T Research††

***Abstract*— The Internet carries a vast amount and a wide range of content. Some of this content is more popular, and accessed more frequently, than others. The popularity of content could be quite ephemeral - e.g., a Web flash crowd - or much more permanent - e.g., google.com's banner. A direct consequence of the skew in popularity is that, at any time, a fraction of the information carried over the Internet is redundant.**

We make two contributions in this paper. First, we study the fundamental properties of the redundancy in the information carried over the Internet, with a focus on network edges. We collect traffic traces at two network edge locations – a large university's access link serving roughly 50,000 users, and a tier-1 ISP network link connected to a large data center. We conduct several analyses over this data: What fraction of bytes are redundant? What is the frequency at which strings of bytes repeat across different packets? What is the overlap in the information accessed by distinct groups of end-users?

Second, we leverage our measurement observations in the design of a family mechanisms for eliminating redundancy in network traffic and improving the overall network performance. The mechanisms we proposed can improve the available capacity of single network links as well as balance load across multiple network links.

I. Introduction

Internet end-hosts peruse a broad range of data over the wide-area network, including HTTP content, streaming video and audio, sensor data, VoIP conversations, gaming data, and files shared in a peer-to-peer fashion. Factors such as similarity in social interests and hobbies, geographic proximity, and the disproportionate popularity of some data mean that an end-host's transfer is somewhat likely to be similar in content, if not entirely identical, to other concurrent network transfers. Some of the content may overlap with transfers that occurred at a different time (e.g. people repeatedly accessing google.com), or with transfers involving different hosts altogether (e.g. multiple news sites reporting the same Reuters' news story, or a bit-torrent based file download).

This repetition of content is likely to be very pronounced at the network edges, such as access links of stub networks or the network links of small regional and local ISPs. At such locations, groups of users and servers who share the network infrastructure also share several other artifacts very closely, such as work and living environment, professional interests, interest in local events or affairs, and client bases.

From the point of view of infrastructure providers, the repeated bytes are essentially useless or *redundant*. If, by some mechanism, the infrastructure is able to carry only unique or useful bytes, then the provider may be able to manage her capacity in a more efficient way. In turn, this could improve the network service for end-users and stub networks by increasing available bandwidth.

In this paper, we make two important contributions. First, we analyze packet level data collected at diverse network locations to gain an in-depth understanding of the nature of redundancy and the extent of redundant bytes in network traffic. These measurements are unique in several ways and are targeted at helping us understand how to build, and where to deploy, mechanisms that can exploit the redundancy in network traffic to the fullest extent. Second, we use our measurement insights to develop two mechanisms for eliminating redundancy and improving the overall capacity of various network links.

Some of the questions we hope to answer via measurement include: How redundant is network traffic? What are the likely sources of redundancy? And, what is the overlap in the content accessed by distinct groups of users? A number of past studies and systems have had a similar goal of identifying and eliminating redundant transfers. However, most such studies [13], [14], [4] have focused on specific applications and looked for repetition of content at the object level. Our work takes a more *information-centric* approach. Specifically, to examine the true extent and nature of redundancy, we look for strings of bytes which repeat across a stream of packets.

In this context, the most similar work to ours was done by Spring et al. [11], which also examines redundancy at the level of strings of bytes. However, there are a number of key contributions that are unique to our work. First, our goal is to evaluate a wide range of system designs for eliminating redundancy beyond the single-link techniques explored by Spring et al. These include mechanisms which apply to pairs of links incident on the same router, and those that apply to the network infrastructure of an ISP as a whole. Second, to achieve this goal, our measurement study is broader and deeper than Spring et al.'s. We study the impact of population size, network location, and the popularity of content on the amount of redundancy we observe. We also measure traffic redundancy at diverse points in the network. This allows us to explore redundancy elimination techniques which are tailor-made to offer the maximum benefit for specific network locations. Finally, the Internet application mix has changed significantly since 1999 and our observations quantify the impact of this shift on the redundancy in network traffic.

To understand the nature of redundancy, in the first part of the paper, we conduct an in-depth analysis of several hours worth of packet traces collected from two very different network locations: a large university's access link serving roughly 50,000 users, and a tier-1 ISP network link connected to a large data center. We analyze this data at two levels – "macroscopic" and "microscopic". In our "macroscopic"

analysis, we ignore whether the content of the packets are associated with particular protocols, particular flows or particular groups of users. Thus, the macroscopic analysis helps us gain a high-level understanding of the redundancy in network traffic. We find that amount of redundancy is significant and varies greatly between network locations. We see approximately 12-15% redundancy in our university traces, but 45% redundancy in the data center. While the range of possible content in transfers explains the difference between university and data center redundancy, the sheer volume of redundant information in our data center traces is quite staggering. In addition, we find that eliminating redundancy would shrink the difference between peak and minimum traffic more significantly – by 25% for the university and by 52% for the data center. This suggests that redundancy elimination can bean effective tool for managing network bandwidth, especially under heavy load.

In order to gain a deeper understanding of the characteristics of redundancy, we conduct several "microscopic" analyses. We find that a large number of data "chunks" see just a few repeated transfers. Surprisingly, this contributes to the overall redundancy much more than "highly" popular data chunks that see frequent ($>$10 in a 60s period) reuse. We also find that traffic redundancy increases relatively smoothly with the number of users that are using a particular link. This is a surprising contrast with previous studies [13], which measured only minor increases in the related metric of Web cache hit rates.

In the second part of our paper, we show how our measurement observations can be employed to develop approaches for eliminating redundancy in the network. These mechanisms are quite helpful in improving the overall capacity of the network, especially at the Internet's edges. The first approach we propose - peak reducer - is similar in many ways to the design in Spring et al. We use this as a building block to design two unique new approaches, a link load balancer and a general network redundancy eliminator. The load balancing design relies on the networks having some redundant, and perhaps idle, connectivity. This is common to a number of network topologies, such as multi-campus enterprises and multi-homed data centers. We show how to effectively use redundancy in their data transfers to move load from constrained links to underutilized links. Finally, we show that redundancy elimination can be applied in much more general settings to reduce the overall load on the network. Our general network redundancy eliminator design relies on either the use of network coding-based techniques [1], [6] or on simple extensions to our load-balancing techniques. The preliminary evaluation of these techniques shows that network traffic redundancy can be exploited effectively for applications far beyond the single link techniques studied in the past.

The rest of the paper is structured as follows. Section II describes our traces in greater detail. We present our measurement results in Section III. Section IV describes our redundancy elimination mechanisms. In Section V we survey related work. We summarize the paper in Section VI.

II. Data Sets

To quantify the extent of redundancy in network traffic, and to shed light on the factors which contribute to the redundancy, we collect several full packet traces at two distinct network edge locations. The first is a large university's access link to the commercial Internet, and the second is the link connecting a tier-1 backbone router with a large data center.

Due to differences in the collection infrastructure at the two locations, the two sets of traces differ in some important respects. We describe the traces next and highlight the differences in the traffic they capture. We summarize the key characteristics of our traces in Table I.

A. University Traces

We monitored the access link of a large University located in the US. The University has a 1Gbps full-duplex connection to the commercial Internet and has roughly 50000 users. We used Endace Systems' DAG 4.3GE dual-port gigabit Ethernet network monitoring card and the associated software for capturing packets and writing them to disk. We logged entire packets (including payloads) going in either direction on the access link.

A couple of limitations of our storage infrastructure imposed key restrictions on our data collection: First, our disk array was unable to keep up with the traffic rate at peak utilization. As a result, we were only able to log traffic from one direction at a time. Second, the amount of space available on the disk was quite limited. Hence, each of our traces cover at most a few minutes' worth of traffic observed on the link.

We collected two sets of traces at the University access link. First, we collected several 60s-long traces between 6am on Friday, Dec 15 and 9pm on Saturday Dec 16, 2006. On average, we collected approximately 3 traces per hour for either direction during this period, resulting in a total of 147 traces for the inbound direction, and 147 for the outbound direction. We alternated between inbound and outbound traffic, with a gap of 30s between the traces for the two directions. The total size of these traces is 558GB. Henceforth, for ease of exposition, we shall use term *Univ-In-60s* to refer to the inbound traffic traces, and the term *Univ-out-60s* for the outbound traffic traces.

Second, on Jan 26, 2007, we collected 191GB worth of traffic – at least 6GB in each direction – at the beginning of every hour starting at 10am and ending at 7pm. Again, we alternated between the incoming and outgoing directions. On average, each trace covered 150 seconds worth of traffic in a particular direction. Henceforth, we shall use the terms *Univ-In-long* and *Univ-Out-long* to describe these traces.

Our collection infrastructure incurred very few drops during the trace gathering ($< 0.1\%$).

B. Data Center Traces

We also collected traces at a link connecting a tier-1 backbone router to a large data center which hosts several popular content providers. A majority (75%) of the traffic served by the content providers is HTTP and Gaming traffic.

Trace name	Description	Dates/Times	Span of each trace	Number of traces	Total Volume (GB)
Univ-In-60s	Inbound traffic at university access link	6:00 AM on 12/15/06 to 9:00 PM on 12/16/06	60s worth of traffic	147	253
Univ-Out-60s	Outbound traffic at university access link	6:00 AM on 12/15/06 to 9:00 PM on 12/16/06	60s worth of traffic	147	305
Univ-In-long	Inbound traffic at university access link	10:00 AM on 01/26/07 to 7:00 PM on 12/16/06	150s worth of traffic	10	97
Univ-Out-long	Outbound traffic at university access link	10:00 AM on 01/26/07 to 7:00 PM on 01/26/07	150s worth of traffic	10	94
DC-In-Out	All traffic at data center access link	4:27 PM on 01/22/07 to 5:05 PM on 01/22/07	38 min worth of traffic	1	42

TABLE I
CHARACTERISTICS OF THE DATA TRACES

Actually, the data center was connected to the backbone via several links and traffic from the data center was load-balanced across these links at a flow level. We tapped exactly one of these links.

Unlike the University traces, however, we were able to tap both directions simultaneously for the Data Center traces. Also, unlike the university traces, we were able to collect a contiguous 38-minute long full packet trace. This trace was collected between 4:27PM ET and 5:05PM ET on Monday, Jan 22, 2007. The total volume of traffic collected was 42GB. Hence forth, we shall use the term *DC-In-Out* to refer to this data.

In all, we collect and analyze 791GB worth of packet-level data.

III. DECONSTRUCTING REDUNDANCY

Our goal in this paper is to understand how we can build mechanisms which leverage redundancy in network packets to improve the overall capacity of the network. The effectiveness of such approaches obviously depends on how redundant network traffic really is. Clearly, if only a small fraction of traffic is redundant, then the cost of deploying such mechanisms may not measure up to the observed benefits. Even if network traffic is heavily redundant, in order to build the optimal mechanisms which exploit the redundancy to the fullest, we must understand several lower-level properties of the redundant information, e.g., how do groups of users share redundant bytes among each other.

With these questions in mind, we perform several "macroscopic" analyses over the above traces; these analyses help us quantify the extent of redundancy. We also perform several "microscopic" analysis which help us understand the underlying nature and causes of redundancy. We present the macroscopic analyses of the traces in Section III-B, followed the microscopic analyses in Section III-C.

The unique contributions of our measurement analysis are as follows:

1) The extent of redundancy in network traffic is significant but differs vastly for different network locations: it is around 15% for the University traces, and 45% for the data center traces.
2) Eliminating redundant bytes from network traffic could smooth out variations in link utilization significantly. The difference between peak and minimum traffic can be shrunk by 25% for the university link and 52% for the data center link.
3) Strings of bytes which are repeated a few times (< 3 repetitions) contribute 50-70% of the observed redundancy. On the other hand, highly popular strings of bytes which are repeated > 10 times, make a minor a contribution (15-30%). Thus, natural implementation alternatives in redundancy elimination systems, such as using smart cache replacement policies, may not work well in practice.
4) The amount of redundancy grows steadily with the number of users that are sharing a network link. This indicates that redundancy elimination systems may offer better benefits when they are deployed at network aggregation points such as access routers. Also, the amount of redundant information can be reduced significantly if groups of users can share bytes from each other's transfers.

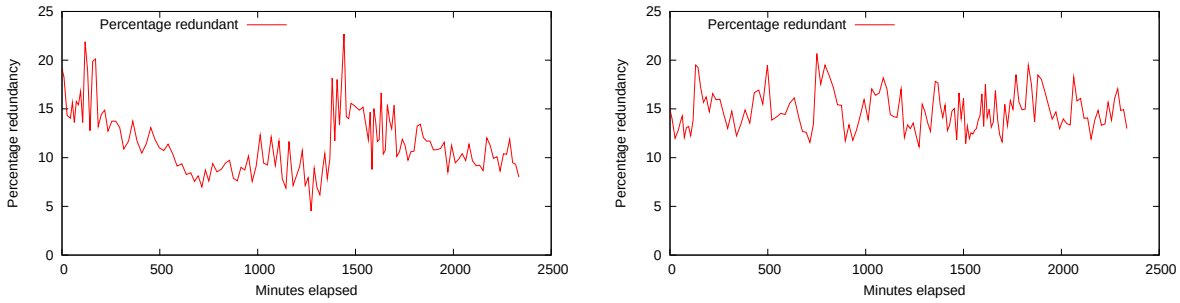
Before describing the results from our analysis in greater detail, we first outline the approach we use to identify redundant strings of bytes from packet payloads (Section III-A).

A. Protocol-independent Redundancy Detection

As mentioned in Section I, we wish to study redundancy at the level of the individual bytes in packets. Spring *et. al* developed an approach which uses random finger-prints to identify strings of bytes which are repeated across packets [11]. The approach treats the content of packets as a stream of bytes and does not distinguish packets on the basis of application or protocol. Hence, it may even find redundancy in packets belonging to completely unrelated applications.

We use this approach in our study of redundancy as well. Next, we provide a detailed overview of this approach.

For each packet, we compute a set of "fingerprints": a fingerprint is the output from applying a hash function to a 64 byte sub-string of the packet's payload. Thus, if a packet has S bytes in its payload, $S \geq 64$, we compute a total of $S - 63$ fingerprints. For packets containing fewer than 64 bytes, we simply compute a single hash over the entire payload.



(a) Incoming (b) Outgoing

Fig. 1. Fraction of redundant bytes at the University access link. The x-axis it the time elapsed (in minutes) since the start of our trace collection at 6:00AM.

We select a small subset of these fingerprints and call them the representative fingerprints for the packet. Essentially, the representative fingerprints are "hooks" into the packet payload which we use to find redundant content. We store a packet's representative fingerprints in a "fingerprint index" and the packet's payload in a "packet store". The index maps the fingerprint one-to-one onto the region in the packet's payload which the fingerprint describes.

For an incoming new packet, we similarly compute its representative fingerprints. For each representative fingerprint, we check if it already exists in the index. Say exactly one representative fingerprint sees a hit in the index. This means that the incoming packet has a 64 byte sub-string that matches with a packet currently in cache. We find the matching packet indexed by the fingerprint and expand the 64 byte matching region in both directions to obtain the maximal region of bytes which are common to the two packets. We insert the new packet into the packet store and change the association in the index so that the matching fingerprint points to the bytes in the new packet's payload.

After finding a maximal match region between an incoming packet and a cached packet, we check to see if other representative fingerprints of the incoming packet see hits in the index as well. When this happens, it means that different regions of an incoming packet have matched with distinct regions of one or more cached packets. As before, we compute the new maximal match regions, and perform the re-association of the matching fingerprints in the index. Finally, we compute the number of bytes in the union of all match regions in the incoming packet.

This approach uses a simple FIFO algorithm for managing the packet store upon overflow: when the packet store is full, the earliest packet in the store is evicted and all fingerprints which map to it are freed.

B. Traffic Redundancy: Macroscopic Analyses

In this section, we perform a few macroscopic analyses which help us gain a superficial understanding of traffic redundancy. Specifically, we focus on quantifying the extent of redundancy and not so much on understanding the causes thereof. We also study how factors such as the link utilization impact the amount of redundancy we observe.

Extent of Redundancy. We focus on the Univ-In-60s, Univ-Out-60s and DC-In-Out traces in this analysis. Where possible, we compare observations from the two sets of traces, but for brevity we present a majority of our results for the University trace. As mentioned previously, we use the technique developed by Spring et al. to identify redundant bytes.

The machines on which we analyzed the data center traces had a limited amount of memory, which constrained the size of the packet store to ≤ 0.4GB. We had a much larger amount of memory on machines where we analyzed the University traces. However, we use a 0.4G packet store to analyze the University traces as well to allow for a direct comparison with the data center results.

Note that the initial 0.4G of each trace will see fewer hits than the rest of the trace and will contribute mainly to warming up the packet store. To ensure this does not distort the amount of redundancy we observe, we report results only for data appearing past the first 0.4G in a trace.

In Figures 1(a) and (b) we show the fraction of redundant bytes over time for Univ-In-60s and Univ-Out-60s traces. We see that on average, 12.51% of the incoming bytes and 15.87% of the outgoing bytes are redundant. On occasion, as many as 22% and 21% of the bytes in the incoming and outgoing direction are redundant. In general, we note that there are several instances where redundancy in the incoming traffic is quite low.

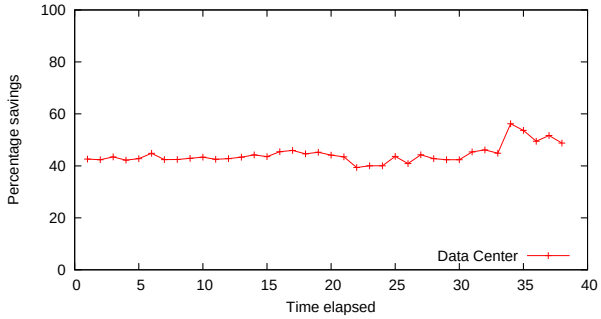
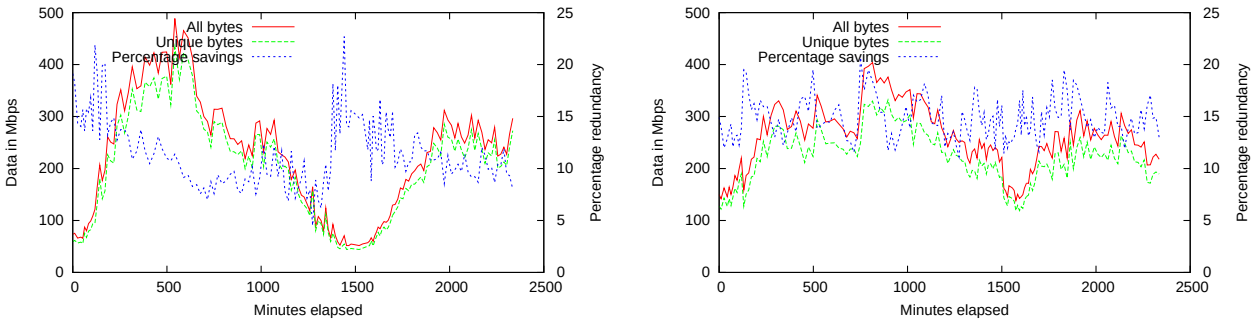


Fig. 2. Fraction of redundant bytes at the Data Center access link. The x-axis it the time elapsed (in minutes) since the start of our trace collection at 4:27PM.



(a) Incoming (b) Outgoing

Fig. 3. Volume, in Mbps, of all bytes, and unique bytes.

In Figure 2, we show results from our analysis of the DC-In-Out traces. We find that the extent of redundancy is much more pronounced for the data center: on average, 45% of the bytes are redundant, which is almost 3X higher than the redundancy factors for the University! In a way, this is expected: the university access link serves a very diverse user base, while the link to the data center caters to a homogeneous group of users most of whom are interested in accessing the few content providers hosted at the data center.

The utilization on the link to the data center was fairly low throughout our trace collection, ranging between 13% and 18%. Although we don't have data to corrobate it, we believe that the fraction of redundancy we observe in the data center traffic will remain roughly the same even at very high link utilization because the number of content providers hosted at the data center is fixed and limited; a greater traffic volume translates directly to more requests for the same content.

Link Utilization. We now study how the amount of redundancy depends on the volume of traffic or the utilization of the network link. If there is a positive correlation – that is, the fraction of redundant bytes is monotonic in link utilization – then we can significantly limit the variation in the link utilization by removing the redundant bytes.

In Figure 3(a) and 3(b), we plot the volume in Mbps of all bytes for the Univ-In-60s and Univ-Out-60s traces (y1-axis). We overlay the fraction of redundancy in the same Figure as well (y2-axis). Contrary to what we expected, we see a slight negative correlation between the fraction of redundancy and the volume of traffic for inbound traffic (Figure 3(a)). However, we there is a definite positive correlation for the outbound traffic (Figure 3(b)).

How do these correlations affect the usefulness of redundancy elimination techniques? To answer this question, in Figures 3(a) and 3(b) we also plot the volume in Mbps of unique bytes. Comparing the total traffic volume with the volume of unique bytes, we note the range of link utilization can be substantially lowered by eliminating redundant bytes: The difference between peak traffic volume to minimum traffic when all bytes are considered is 270Mbps for Univ-Out-60s and 420Mbps for Univ-In-60s; When only unique bytes are considered the differences are 200Mbps ($\sim 25\%$ lower) and 370Mbps ($\sim 12\%$ lower). Because of the positive correlation with load, the reduction is much more impressive for the outbound direction than the inbound direction.

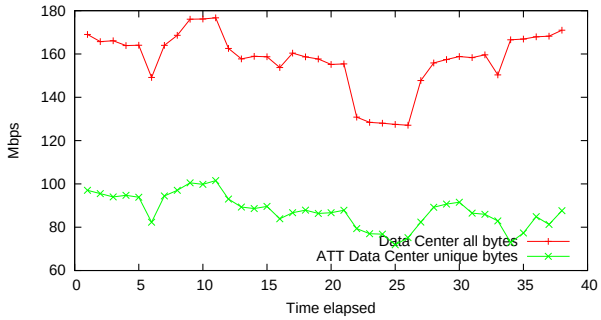


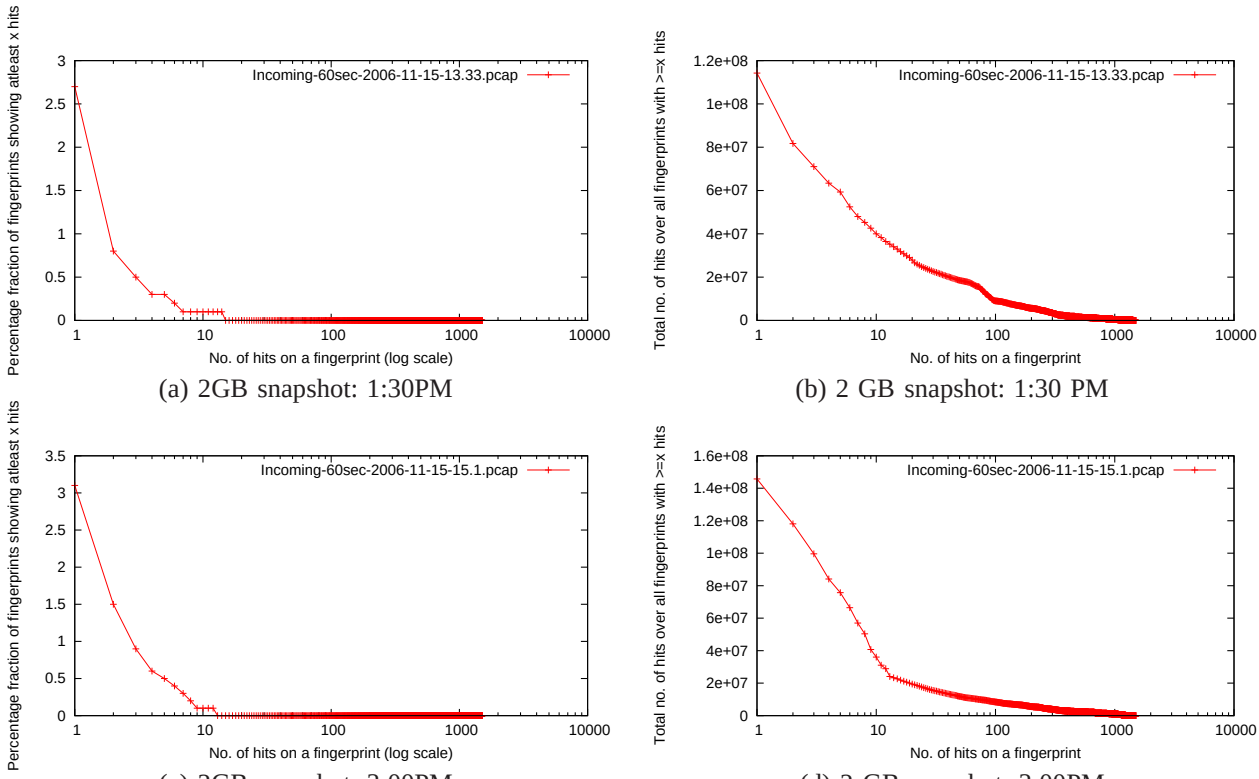
Fig. 4. Volume, in Mbps, of all bytes, and unique bytes for the Data Center traces.

The difference between the peak and minimum traffic volumes for the data center, shown in Figure 4, is 54Mbps. It reduces by 52% to 26Mbps when only unique bytes are considered. Thus the benefits are more dramatic in the data center setting.

Thus, eliminating redundant bytes from network traffic can reduce the variations in link utilization by a significant amount. In Section IV, we leverage these observations to motivate the design of a "peak reducer" for reducing the load on constrained network links.

Redundancy vs Packet-store size. In practical redundancy elimination devices, the size of the packet store contributes crucially to the overall cost and performance of the devices. Larger packet stores may help identify and eliminate more redundancy than small stores; however, it may be very expensive to provision larger stores on the devices. Thus, unless the improvement in the fraction of redundancy identified is significant, large packet stores may not be a viable option.

In the above analyses, we used a fairly small packet store of size 0.4GB. To understand if larger packet stores can capture a much greater amount of redundancy, we analyze the Univ-In-Long and Univ-Out-Long traces using packet stores of sizes $S = 0.5$GB, 1GB and 2GB. Just like the previous analyses,



(a) 2GB snapshot: 1:30PM

(b) 2 GB snapshot: 1:30 PM

(c) 2GB snapshot: 3:00PM

(d) 2 GB snapshot: 3:00PM

Fig. 5. Popularity of content in two traces: In Figures (a) and (c), we show a CDF of the number of fingerprints which see a certain minimum number of hits. In (b) and (d), we show the total number of hits contributed by all fingerprints which see a certain minimum number of hits. Figures (a) and (b) are for a Univ-In-60s trace collected at 1:30PM, while (c) and (d) are for a Univ-In-60s trace collected at 3:00PM.

we compute the redundancy fraction for the data past the first S bytes in the trace. Although we do not show the results here, when we set $S = 2$GB, the fraction of redundancy was 3-4 percentage points higher for Univ-In-Long traces, and 4-6 percentage points higher for Univ-Out-Long traces compared to the 0.4G packet store. These improvements are significant relative to the average redundancy with $S = 0.4$G (this fraction is 15% for the Univ-In-Long traces and 18% for the Univ-Out-Long traces). Thus, while our 0.4G packet store is able to capture a majority of the redundant bytes in our traces, using larger packet stores may help capture a significant amount of additional redundancy.

C. Microscopic Analyses

While the previous section shed light on the prevalence of redundant information in network traffic, in this section, we explore several lower-level aspects of the redundancy. Our goal here is to understand in detail the nature of redundancy, i.e. key properties which are common to the redundant bytes.

Redundancy Pattern. The first question we ask is the following: is the redundancy in network traffic primarily due to a few pieces of content repeated multiple times or multiple pieces of content repeated a few times each? If the former is true, then, a small packet store which caches only the most frequently accessed chunks of packet data would be sufficient for identifying a significant fraction of all redundancy in network traffic. However, if the latter is true we may have to store many more chunks of data in a much larger packet store.

To answer this question we must count how often a "chunk of data" appears in different packets. This is difficult to track because a given data chunk may not repeat as a whole across packets; instead, different parts of a data chunk may repeat in different sets of packets. To simplify things we focus on small fixed-size chunks instead; specifically, we count how often 64B substrings repeat in a trace. Since the total number of 64B substrings is almost as large as the total number of bytes in packet payloads, tracking the frequency counts for all substrings is computationally difficult. Thus, we perform this analysis on small snapshots (1GB and 2GB long) of the traces we collected.

In Figures 5(a) and (c), we show a CDF of the number of 64B chunks which see a certain minimum number of matches for 2GB-long snapshots of Univ-In-60s trace collected at 1:30PM and 3:00PM. The redundancy fraction in these traces was 15% and 14%, respectively, were redundant in these traces. The results for 1GB snapshots we qualitatively similar and we omit them for brevity.

First, we note that $\sim 97\%$ of the chunks do not see even a single match. Among the chunks that do see a match, a large fraction see very few matches: $\sim 80\%$ see under 3 matches, fewer than 3% see more than 10 matches, and a handful of chunks see as many as 1000 matches.

To put these numbers in better perspective, in Figures 5(b)

and (d), we show the total number of matches contributed by chunks which see a certain minimum number of matches. Figures (a) and (c) showed that few chunks see more than 10 matches, and a negligible fraction see more than 50 or 100 matches; Complementing these results, Figures (b) and (d) show that the aggregate number of matches contributed by such chunks is also not very high. For example, chunks which see at least 10 matches contribute to 30% of all hits for the 1:30PM trace and 14% of all hits for the 3:00PM trace. For chunks which see at least 100 matches, the contribution is even lower: 8% for the 1:30PM trace and 5% for the 3:00PM traces.

This suggests that a major fraction of the redundancy we observe arises from multiple chunks of data which see a handful of hits each. We also analyzed two Univ-Out-60s traces and two DC-In-Out traces in a similar fashion, and observed qualitatively similar results. We omit results for brevity.

As we mentioned in Section III-A, Spring et al store packets in a FIFO order, evicting the oldest packet upon overflow. So is FIFO ordering a good idea in practice? Our initial thinking was that employing "smarter" cache management techniques such as LRU (evicting the least recently used) or Frequency Counts (evicting a packet which saw the fewest hits) may yield better overall redundancy. However, the above results suggest the "smarter" cache management approaches are unlikely to offer any improvement over FIFO.[1]

Properties of Users' Sharing. In Section I, we posited that one of the key reasons for the prevalence of redundant information is an overlap in user interests. In practice, the extent to which this contributes to the overall redundant traffic, and to how one can exploit redundant information, depends on how groups of users share interests and information. Consider two situations: one where users belonging to different departments in a university have no overlap in interests, and another where there is a significant overlap. Within any window of time, we would expect to see a greater fraction of redundancy bytes in the latter case. Also, if groups of users in the latter case are able to share redundant bytes with each other, then this can save a significant amount of network bandwidth.

This notion of sharing of bytes has a counterpart in cooperative Web proxy caches [4], [13]. The benefits of such systems were studied extensively by Wolman, et al. in [13], [14]. However, there is a subtle but important difference between cooperating proxies and sharing redundant bytes across groups of users. We explain the difference later in Section IV.

We now explore how the sharing of bytes among groups of users impacts the observed redundancy. We focusing on the Univ-In-60s traffic traces. Since we do not track individual users in our traces, we approximate the sharing among users by instead studying sharing of information among distinct internal IPs.

We study the dependence of the fraction of redundancy on the number of users sharing information in four Univ-In-60s traces collected at distinct times-of-day. We use the largest possible size for the packet store within the constraints on our setup, namely 2GB. Some of the 60s traces were $>$ 2GB in size while others were smaller and therefore the traces experience cache warming effects to varying degrees. To ensure that this does not create a bias in our analysis, we analyze only a 2GB snapshot from the beginning of each trace because it is guaranteed to fit entirely within the packet store.

The results from our analysis of the four traces are shown in Figure 6. For each trace, we produce the fraction of redundancy at S client IPs by computing the average amount of redundancy within the traffic destined for 5 random subsets of S IPs each.

Before interpreting these results, we draw the reader's attention to Figure 7 which plots the byte hit rate as a function of client population size for a large proxy cache at the University of Washington (this is reproduced from [13]). There is a clear evidence of diminishing returns in the object hit rate. Our analysis of the Univ-In-60s traces shows a steadier increase in the fraction of redundancy: In Figure 7, the improvement in the object hit rates between 2000 clients (43%) and 10000 clients (52%) is small relative to the hit rate for 2000 clients.

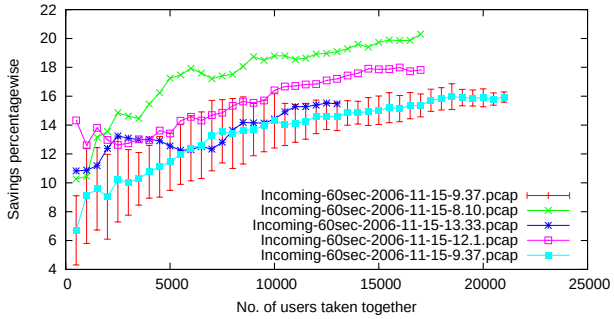


Fig. 6. Fraction of redundancy vs number of IPs for four Univ-In-60s traces. Error bars are show for just one of the four traces. The utilization of the links at these four time instances are: 180Mbps (0810 hrs), 260Mbps (0937 hrs), 353Mbps (1200 hrs) and 422Mbps (1333 hrs).

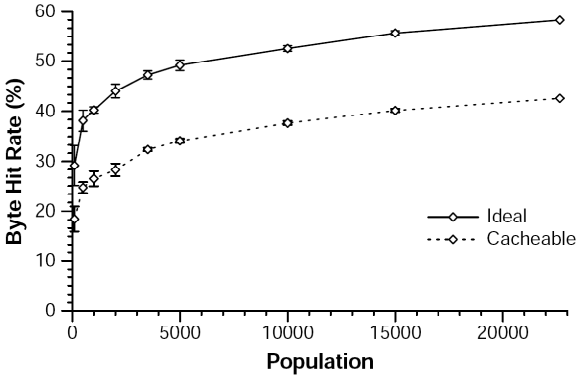


Fig. 7. Proxy cache byte hit rate as a function of client population in the UW study (circa 1999). This figure is a reproduction of Figure 1 in [13].

[1]In fact, we implemented LRU and found that it offers no benefit over FIFO.

The relative improvement is more impressive in Figure 6 – 10% at 200 clients to 15% at 10000.

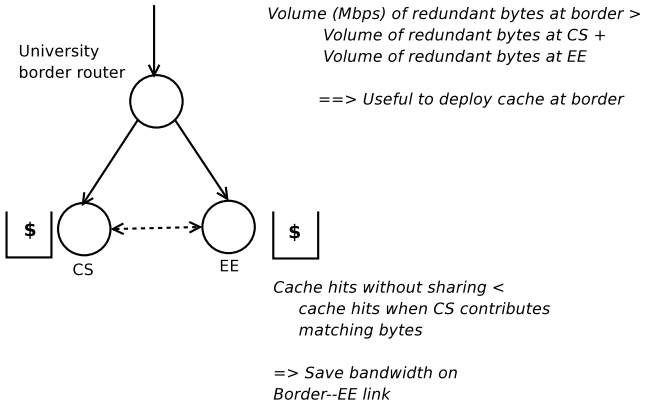


Fig. 8. Interpreting the user-sharing results.

Wolman, et al. suggest that sharing objects among a large collection of cooperative caches deployed across an entire organization is not very beneficial compared to using independent caches for small groups of users (e.g. per department). In contrast, our results suggests that redundancy elimination techniques may be more beneficial if they are deployed at locations where traffic aggregates from a large number of users, e.g. the border router of an enterprise. This is shown pictorially in Figure 8. Our results also suggest that cooperation among redundancy elimination boxes (shown with $ signs in Figure 8) that serve small groups of users can substantially improve redundancy identification within a network. In turn, this can save bandwidth on internal network links (Figure 8). In Section IV, we use these observations to guide the design of a network link load-balancer which facilitates sharing of redundant bytes across packet stores shared by different groups of users.

The previous groups arbitrary sets of users together. To understand if a more informed grouping users according to their network location network affects our observations, we grouped the IPs into /24 address blocks. Upon analyzing the dependence between the number of /24s sharing data with each other, and the amount of redundancy identified, we saw the same trends as Figure 6. We omit these results from the paper.

Redundancy in popular applications. In Table II, we show the redundancy that we observe in popular network applications. For each application, we show what fraction of all bytes belong in the application, and the fraction of redundancy in the application's data. Our observations here are different from Spring et al. in a few key ways. For example, Spring et al. [11] found HTTP traffic and SMTP traffic to be modestly redundant ($\sim 30\%$ and $\sim 20\%$, respectively, when averaged across inbound and outbound). In contrast, we see that the redundancy in SMTP traffic is much higher, while the redundancy in HTTP traffic is lower. Also, Spring et al. observed a much larger fraction of port 80 traffic (62% averaged across inbound and outbound) of all bytes belong to port 80 traffic, while we an average of 45%. These differences show evidence of the shift in the application mix and usage since 1999. We also note that 5% of all bytes, on our average belong to HTTPS and hence are part of encrypted payloads. As expected, HTTPS shows minimal redundancy.

IV. Exploiting Redundancy

In the previous section, we studied the extent and the nature of redundant bytes in network traffic. We found that a significant fraction of bytes are redundant and that sharing of redundant bytes among groups of users can yield significant benefits. Next, we show how to leverage these observations to design mechanisms for improving network performance. Specifically, we present three different techniques which can be used to manage scarce network capacity at edges of the Internet. Here, we make liberal use of the term "edge" to include: (1) stub networks, (2) data centers (3) ISP access infrastructures, (4) the network infrastructure of small, regional tier-3 or tier-4 ISPs, and (5) boundaries between neighboring ISPs, or peering locations, specifically those involving small regional ISPs.

The Internet's edge is known to be capacity-limited. It is widely believed that bandwidth constraints at last mile home access links contribute to the packet drops and queuing delays experienced by most Internet transfers. Enterprises and stub networks lack the economic means and incentives to over-provision their access connections. Large data-centers buy plenty of excess bandwidth from their ISPs to accommodate surges in traffic; but the excess bandwidth is also quite expensive.

Even if we set last mile constraints aside, recent measurement studies have shown that limited bandwidth provisioning in smaller ISPs, such as regional and local ISPs, DSL and Cable Internet providers, can also cause serious congestion on end-to-end transfers [2], [5]. These studies also show that network peering points have very little spare capacity; in particular, at peering links where one or both ISPs are small (e.g. tier-3 or tier-4), the available capacity is often under 10Mbps.

In what follows, we present three preliminary techniques which leverage traffic redundancy to save bandwidth and enable cost-effective management of traffic at the aforementioned edge locations.

Our first technique – peak reducer – is similar in many ways to the design in Spring et al. and can help reduce traffic variations on individual network links. Our second technique – load balancer – can use redundancy elimination to move traffic from constrained links to under-utilized links. Our third technique uses redundant traffic to either identify network coding opportunities [1], [6] or to perform load balancing across multiple end-to-end paths in a network. We present preliminary evaluation results that show the effectiveness of these techniques.

Port Number	Protocol	Univ-In-60s		Univ-Out-60s	
		% of all bytes	*% redundancy*	*% of all bytes*	*% redundancy*
20	ftp-data	0.04	16.93	1.1	7.5
25	smtp	0.02	22.69	0.08	70.63
53	dns	0.22	21.39	0.14	47.99
80	possibly http	58.10	12.49	31.69	20.37
443	https	0.60	2.00	3.59	2.08
554	rtsp	3.38	1.99	1.34	24.40

TABLE II
REDUNDANCY IN KEY APPLICATIONS

A. Redundancy Eliminator/Peak Reducer

The first approach we propose is a simple "Redundancy Eliminator" or "Peak Reducer" which works across individual network links. This is based on a similar idea that was sketched in [11]. We will use the redundancy eliminator as a basic building block in the remaining two techniques.

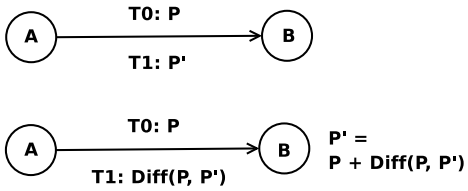


Fig. 9. Redundancy Eliminator/Peak Reducer.

Referring to Figure 9, we assume that neighboring routers A and B use the algorithm in Section III-A for identifying redundant content. Both routers store the contents of the packet they saw in the recent past, and they use packet stores of the same size. As mentioned earlier, packets in the store are indexed by a small number of finger-prints. When an incoming packet has a matching fingerprint, we grow the matched bytes leftward and rightward to obtain the maximal match region. Redundancy elimination works at router A for packets traversing the A–B direction.

Say a fraction of bytes in packet P', which arrives at time T1 at A, match with those in packet P which arrived at time T0, $T0 < T1$. When A encounters P', it computes the maximal match region with packet P. A then transmits a stripped version of P', say P". P" has the same header as P', a shim header, and the bytes which are not common to P and P'. The shim header contains (1) the fingerprint which matched in the two packets (4 bytes) and (2) the byte range summarizing the match region corresponding to the fingerprint (~ 4 bytes per match region). A also marks a bit in the P" IP header (e.g. in the TOS field) to indicate to B that new P" needs reconstruction.

When B receives P", it parses the shim header, looks up the packet P using the finger-print (note that in B's packet store, the fingerprint will point to the packet P), retrieves the bytes from P as indicated by the byte range and inserts them into P", to create P.

In our original description of Spring et al's algorithm in Section III-A, we noted that the approach could identify multiple match regions within the same packet. One might consider encoding only the first one or two matching regions and ignore other matching regions to keep the size of the shim header small. We evaluated this approach using Univ-In-60s traces and found that it misses out on a large fraction of redundant bytes (we saw a reduction in the redundancy fraction by 15% when we encoded the first three match regions). Thus, practical implementations must either use a large fixed-size shim, or a variable length shim which encodes all match regions.

By filtering out redundant bytes from packets traversing the A-B link, the peak reducer lowers the average link utilization. As we show in Figures 3 and 4 of Section III-B, the redundancy elimination approach can also significantly reduce the variations in the volume of traffic network links. Thus, when networks use redundancy eliminators on their links, they will have less of a need to over-provision link capacity.

B. Redundancy-based Load Balancer

The above approach applies the benefits of redundancy elimination to managing load on individual network links. This is useful to manage the capacity on an enterprise access link, a congested link in an ISP's access tree, or at a peering link.

We now propose a "Load Balancer" which extends the benefits across several links of a network, specifically to links which share a ingress end point. This approach leverages the observations in Section III-C that the amount of redundancy that we can identify increases steadily with the number of users sharing information.

We illustrate the approach using the example in Figure 10.

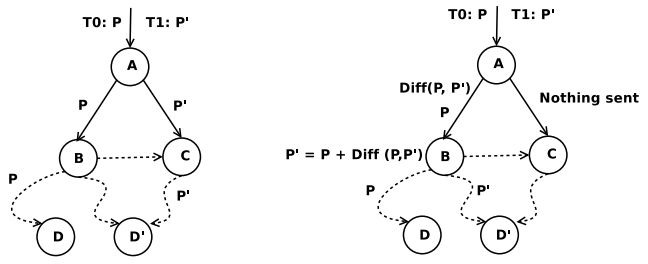


Fig. 10. Redundancy-based load balancing.

Say router A receives a packet P' at time T1. According to A's routing table, the next hop for P' is C; by default, P'

would have traversed A-C-D' to reach its eventual destination D'.

We propose a simple modification to traditional route look-up at A and argue that this modification can save the bandwidth on links A-C and A-B. We assume that all routers—A, B and C—use a single packet store and they all employ the redundancy eliminator approach to identify and encode redundant information.

Our modification requires that A look not only at the destination address in P', but also at the payload in P' to compute the next hop. To do this, A annotates each packet in its store with the interface on which it was forwarded.

Thus, referring to Figure 10, when A receives P', the annotation in A's packet store will indicate that some of the bytes in packet P' have already traversed the A–B link at an earlier time, say as part of packet P. Instead of sending P' along A–C according to its traditional routing table, A invokes its redundancy eliminator to send the unique bytes in P' (i.e. diff (P', P)) to B, along with the matching fingerprint and the bytes range for the redundant region. B can then reconstruct P' from its cache. On the other hand, if router A finds that the bytes in P' are all unique, then it sends P' to C, just as its routing table dictates.

After the reconstruction, B looks up the destination address for P' in its routing table and forwards it along to the destination. This may take P' along the B-C-D' path or along an entirely different route; this depends on B's routing table.

Discussion of the approach. This approach is a variant of traditional load balancing, but with two key differences. First, it functions at a packet level while traditional load balancing approaches work at a flow level. We acknowledge that this will create reordering in packets received by end-hosts.[2]. Second, unlike load balancing, it can lower the aggregate network traffic volume at the same time as balancing the load on links. Thus, redundancy-based load balancing increases the overall capacity of the network.

We make a few key assumptions above in the above simplistic description of load balancing: First, we assume that the amount of spare bandwidth on the path between B and D' is at least as high as the spare bandwidth on the path between C and D'. If this is not true, then the load balancing approach may overload network links which are already congested. Thus, for the approach to be effective, routers in the network must have reasonably accurate information on the status of various network links.

Second, we assumed that both B and C are located in the "general direction" of D'. If this is not the case, then load balancing can cause packets to linger in a network for a long time. To prevent this from happening, the load balancer can keep a short list of 2-3 "good" next hops per destination (i.e. routers which take packets closer to the destination), and only use these hops to forward packets.

The load balancing approach is similar in spirit to sharing among proxy caches, but there is an important difference. For redundancy-based load balancing to work, we either need: (1) a direct link between the packet stores which are sharing bytes, or (2) a path between them that bypasses the network links which are traversed by packets arriving at the packet stores. We note this is common to a number of network topologies, such as multi-campus enterprises and multi-homed data centers. In contrast, the requirements are less stringent for proxy caches: as long as proxy caches are closer to the groups of users they serve than to most general Internet Web servers, they can offer significant savings in Web access latency and wide-area bandwidth.

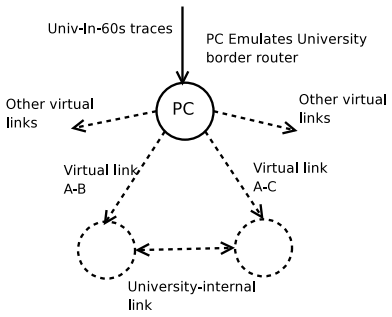


Fig. 11. Emulation of Redundancy-based load balancing.

Preliminary Evaluation. We now present preliminary evidence of the benefits of our load balancing approach. Specifically, we show how sharing redundancy within a network can improve the fraction of redundant bytes which can be eliminated from network links.

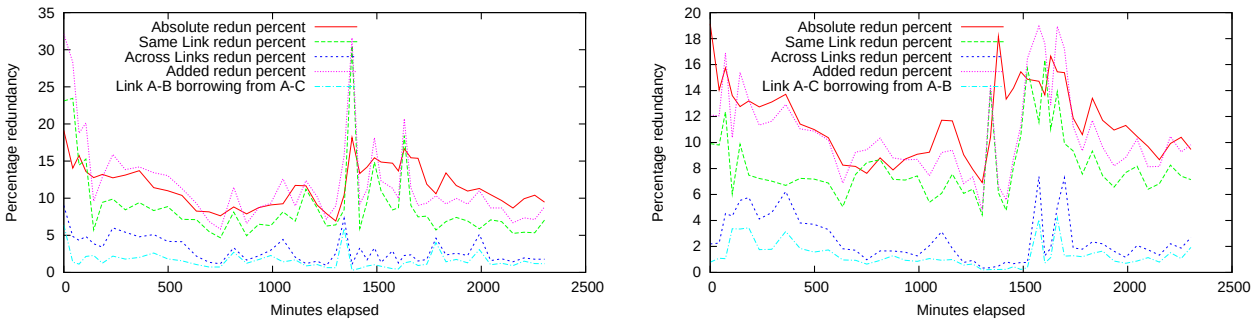
To perform this study, we use a simple emulation running on a stand-alone PC (See Figure 11). We modified Spring et. al's approach to use an annotated packet store, and to forward packets using both the destination address as well as the redundancy in the packet payload, as described above. We run the load balancer on our stand alone PC.

We feed several Univ-In-60s traces into the PC. Thus, the PC emulates the University's border router. We obtained the forwarding table from the university border router and use the table to map the packets in Univ-In-60s traces onto "virtual links" incident on the stand alone PC. The virtual links represent university-internal links which carry incoming traffic to internal destinations. We call them "virtual" because any packet we send on them during our emulation is simply used for tallying the redundant bytes and then dropped. We used a 2GB packet store on the PC.

Based on the border router's routing table, we had to use $\sim$ 10 such virtual links. Of these, two links—which we call link A-B and link A-C—carried the greatest amount of incoming traffic during the course of our measurements. In reality, routers B and C are connected by an internal network link as well.

The central goal of our emulation experiment is to quantify the additional byte savings from applying the load balancing approach at the border router. We present the results of our

[2]However, several recent mechanisms have been proposed for dealing with out of order packet arrivals at end-hosts [16].



(a) Link A-B (b) Link A-C

Fig. 12. Analysis of the benefits of redundancy-based load balancing using a simple emulation set-up.

emulation in Figure 12. Our analysis uses the Univ-In-60s traces.

Figures 12(a) and (b) show the fraction of redundant bytes traversing the A-B and A-C links, respectively, when load balancing is not employed. This is labeled "same link" redundancy. The fraction varies between 5% and 25% on link A-B, and 5% and 15% on link A-C. In a few cases we see that the fraction of redundancy on either link is higher than the overall redundancy in the Inbound traffic at the university (this is labeled "absolute" redundancy). However, in a majority of instances, the links A-B and A-C see lesser redundancy than the overall amount.

We also show the additional redundant bytes on link A-B and A-C when load balancing is employed across all internal links. We plot this as a fraction of all bytes traversing the link and label it "across links" redundancy. This emulates an *ideal* situation where links A-B and A-C are able to share redundant bytes with all other internal links incident on the border router, as well as with each other. We see that, on average, the magnitude of the additional redundancy is roughly a third of the redundancy on the link. Surprisingly, there are instances where the added redundancy fraction is comparable to the redundancy within the link. By summing up the additional redundancy and the redundancy in traffic within a link, we can observe the benefit that load balancing brings to traffic on a single link; this is shown by the line labeled "Added" redundancy which closely tracks, and some times overshoots, the overall redundancy fraction at the border router. These results show evidence of the value of redundancy-based load balancing in managing the load on groups of network links.

Finally, Figures 12(a) we show the additional redundant bytes on link A-B when load balancing is applied only across A-B and A-C (As we mentioned earlier, an internal link directly connects nodes B and C in the University network). A similar curve is shown for link A-C in Figure 12(b). Again, there are several instances where both links benefit significantly by sharing redundant bytes with each other using the load balancing approach.

Possible deployment domains. In a practical deployment, the load balancer will likely be deployed at "aggregation points", such as enterprise gateway routers and ISP peering routers. Other "internal routers", such as department gateways in a university should simply use the rudimentary peak reducer/redundancy eliminator. Also, the load balancing functionality could be turned off by default, and switched on as soon as the load on network links crosses a certain threshold.

Next, we enumerate several operational scenarios where redundancy-based load balancing could prove very useful.

Multi-homed data centers which load-balance out-bound traffic on their ISP connections can employ this approach to optimize the usage of their multiple Internet connections. Of course, the ISPs must be willing to cooperate with the data center (e.g. in reconstructing packets).

Multi-campus enterprises which buy expensive VPN services from large ISPs to connect multiple branches together can also use this approach to save bandwidth costs on their VPN tunnels. For example, suppose an enterprise uses three VPN tunnels to connect its head office in New York city (equivalent to router A in Figure 10), with branch offices in Yorktown, NY (router B) and White-Plains, NY (router C), and to connect its branch offices directly. When an important memo from the NYC office needs to be distributed to all employees, our load balancing approach will ensure that the tunnels carry a minimal amount of traffic.

Regional ISPs which serve limited geographic regions, e.g. tier-3, tier-4 and DSL providers, can also find this approach useful. If the ISP has a backbone of its own, then this approach is directly applicable: for example, out of the 10 tier-3 ISPs study in the Rocketfuel project [10], one ISP has a small, 3-PoP, fully-connected backbone which looks very similar to the triangle configuration in Figure 10 (AS 3701). The inter-link latencies are 5-10ms.

Even if the ISP leases connectivity at multiple locations from a single large ISP, the approach is still applicable (this situation is analogous to the multi-campus enterprise example). Of course, the larger ISP must agree to perform the load balancing.

Finally, *single-campus enterprises* who may not have the means to over-provision their internal network links can use this approach to for intra-enterprise traffic management. If the enterprise's IP network is a tree rooted at its Internet gateway, then the approach is useless. However, we expect that most

small enterprises would add a few additional internal links for redundancy reasons as well as to maintain internal connectivity in the event that the Internet gateway goes down. The university network we study in this paper has the property that a few "cross links" connect different departments internally.

C. Exploiting Network-wide Redundancy

We now discuss two other techniques to exploit redundant information traversing unrelated network links. These approaches take a more holistic view of the redundancy in the information carried by the entire network. These approaches are useful to further reduce the load on backbone links of small region ISP networks.

We develop these techniques using the toy example in Figure 13(a). This figure shows the backbone network of a small ISP, consisting of four routers - A, B, C and D - located at different PoPs, and two internal routers E and F. Two similar copies of packet P enter the network at A, destined for C and D respectively. Two copies of another packet Q enter at B, destined again for C and D. The routes taken by the packets in the default case, where routers forward on the basis of destination address alone, are shown in Figure 13(a). Note that the total number of packets traversing all links in this example in 8.

Network Coding. The first technique we proposed is based on network coding [1], a topic of intense recent interest in both the networking and theory communities. Several techniques based on network coding have been proposed to improve the throughput of multicast transmissions [17], [3]. Recently, Katti et al have shown how to extend the benefits of network coding to unicast traffic in the wireless domain [6]. They argue that the broadcast nature of the wireless medium, where a single packet transmission can be heard by multiple receivers at once, can create several unique opportunities for coding. In what follows, we use an example to argue informally that *redundant content, which may in fact span multiple related or unrelated unicast or multicast streams, can offer similar coding opportunities as well*.

We illustrate network coding in Figure 13(b). Say routers A and B employ a modified load balancer which simply marks a few bit in the header of a repeated packet indicating that a similar packet was forwarded on a different network link. The modified load balancer simply forwards the marked packet along the default forwarding routes. Thus, in Figure 13(b), router A marks a few bits in the header of the copy of P traversing the A–E link, indicating that a similar copy was forwarded on another network link (A–C) toward C. Similarly, B marks the copy of Q traversing the B–E link indicating that another copy was forwarded toward D. When E receives the marked packets, it immediately identifies an opportunity to performing "coding" on the information contained in the packets: E combines the packets and forwards a single copy of $P \oplus Q$ to F. F duplicates the combined packet and sends a copy each to C and D. C can in turn retrieve Q from P and $P \oplus Q$. The total number of packets traversing the network in this example is 7, one less than the default case.

Next, we present a variant of load-balancing which is a simpler to implement. Also, as we will show, the approach can offer better performance than the coding approach for the example we consider (We have constructed other examples where coding is identical to or marginally better than the approach below. We omit these for brevity).

Load-balancing revisited. Consider Figure 13(c), where we show how redundancy-based load balancing at routers A and B can further improve the load on the network. In this case a single copy of P and Q first traverse the A–C and B–D links respectively. When it is time to send another copy of P to D, router A sends a small token to C, just like in the peak reducer example, indicating C to retrieve a copy of P from its packet store, and forward it along to the eventual destination. A similar sequence of actions are executed independently for Q. It may well be the case that according to C's routing table, the reconstructed copy of P must be forwarded to D via F, and then onward to the final destination. In this case, the total number of packets traversing all network links is 6, one less than the coding example, and 2 less than the default routing approach.

Coding vs. Load-Balancing. while the network coding approach is theoretically appealing, it faces serious implementation challenges. For example, router E may have to buffer "code-able" packets such as P for a small amount of time, until a suitable counterpart is found. In the above example, Q is suitable for P because a replica of Q has already been forwarded to the network location (i.e. router D) where P eventually needs to go. Packets must carry additional information for routers to consider them suitable for coding. Also, only the redundant bytes in the packets can be coded together. The unique bytes must be transmitted separately, and this may require the packet to be fragmented.

In contrast with the coding approach, the load-balancing approach is much simpler to implement. The routers in the load-balancing approach use purely local information to make forwarding and redundancy elimination decisions. For example, the load balancing approach treats P and Q independently, while the coding approach naturally needs to consider them together. Also, load-balancing does not cause packet fragmentation. Thus, it imposes fewer performance constraints on routers. We believe that because of its local nature and ease of implementation, the load balancing approach is also more likely than the coding approach to identify and exploit the redundancy in network traffic.

V. Related Work

Several papers have looked at various aspects of redundancy in network information: for example, repeated requests for Web objects, repetitions and commonalities in packets header, as well as repetition at the byte level in network traffic. We compare these papers with our work next.

Web Caching. Wolman et al. studied the sharing of Web documents among users at the University of Washington [14]. They showed that users are more likely to request objects that

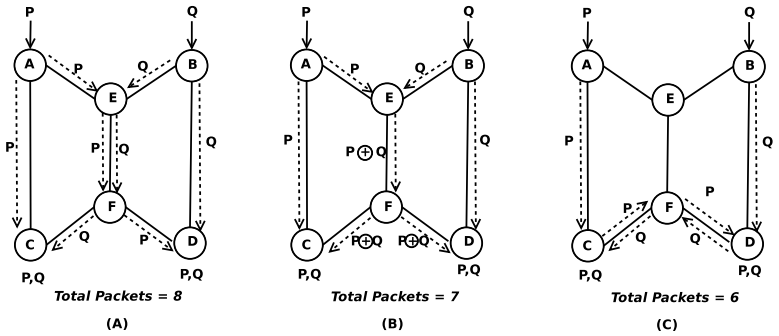


Fig. 13. General network redundancy elimination.

are shared across departments than objects that are only shared within a department. In follow-up work, Wolman et al studied the benefits of cooperative Web caching of Web objects using traces of Web requests collected both at the University of Washington and at Microsoft Corporation [13]. They showed that while sharing of Web objects across departments or divisions in organization can improve Web object hit rates, there is a strong evidence of diminishing returns when the population of clients sharing the cache crosses a certain limit. In contrast with these papers, our work takes a more information centric view by focusing on the byte level repetition in traffic and its relationship with the departmental division of clients and size of the client pool. Thus, our study answers a more general question: How much information are Internet users accessing in common?

Entropy of Headers. Several researchers have studied the entropy of packet headers, or of the payloads of special types of traffic (e.g. worm payloads). For example, Xu et al. [15] and Lakhina et al. [7] independently developed anomaly detection techniques that look for unusual changes in the entropy of IP headers in a stream of traffic. Their tools are meant for performing offline analysis of traffic. Lall et al [8] developed data streaming algorithms for estimating the entropy of IP headers in an online fashion for use in anomaly detection. Singh et al [9] use Rabin's fingerprint algorithm on packet payloads to look for common exploit sequences, and flag the outbreak and spread of malicious content. A subtle difference between these papers and our work is that while these studies focus on identifying uncommon or anomalous behavior in traffic, our goal is to study common usage and access patterns, and understand how to exploit the resulting traffic redundancy to improve network efficiency.

Other studies on traffic redundancy. In the past, a few research studies have developed techniques for estimating the redundancy in network traffic. For example, Sung et al [12] developed data streaming algorithms for approximating the entropy of packet payloads. As mentioned in Section I, Spring et al. developed a protocol independent approach for identifying redundant bytes in network traffic [11]. We have highlighted the key differences between our approach and Spring et al [11] in Section I. Below, we highlight other key differences.

Spring et al applied the algorithm described in Section III-A to analyze the traffic collected on a corporate research center's Internet link. They found that, on average, 20% of the bytes were redundant in the inbound direction, and 50% were redundant in the outbound direction [11]. We study two network locations - a university access link, and the network connection of a data center - both of which are different from the setting analyzed by Spring et. al. While we find a great deal of redundancy in the data center traffic, the redundancy fraction is much lower for the university setting. Several of the data analyses we perform - e.g. the popularity of chunks of data and the properties of sharing of bytes across network links - are quite unique and have not been explored before.

VI. Summary

Internet hosts participate in a wide variety of transfers. Often, there is a substantial repetition in the content of different transfers due to factors such as similarities in users' interests, geographic proximity, application protocol headers etc.

Our goal was to understand the extent and nature of the repetition in network data and explore how this understanding may be leveraged to build mechanisms that eliminate repeated bytes from network traffic. We analyzed several hours' worth of packet level traces collected at two distinct network locations. Our analyses showed that a significant fraction of network traffic is redundant, but there are key differences in the extent of redundancy observed at different network locations. A deeper evaluation of the redundant information revealed two important properties of repeated bytes. First, a large number of data chunks repeat only a few times each, yet they contribute to a major portion of the overall redundancy. Second, traffic redundancy increases relatively smoothly with the number of users that are sharing a particular link.

Using these observations as the foundation, we developed three techniques that can improve the overall network performance by removing redundant information from network traffic. The first technique, peak reducer, applies to individual network links. The second, load-balancer, can manage the load on multiple links which share a common end-point. The third,

general redundancy eliminator, can be applied in more general settings to improve the aggregate network capacity.

REFERENCES

[1] R. Ahlswede, N. Cai, S. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[2] A. Akella, S. Seshan, and A. Shaikh. An Empirical Evaluation of Wide-Area Internet Bottlenecks. In *Internet Measurement Conference*, Miami, FL, Nov. 2003.

[3] P. Chou, Y. Wu, and K. Jain. Practical network coding. In *51st Allerton Conf. Communication, Control and Computing*, 2003.

[4] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. In *ACM SIGCOMM*, pages 254–265, 1998.

[5] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang. Locating internet bottlenecks: algorithms, measurements, and implications. In *ACM SIGCOMM '04*, pages 41–54, 2004.

[6] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. XORs in the air: practical wireless network coding. In *ACM SIGCOMM*, 2006.

[7] A. Lakhina, M. Crovella, and C. Diot. Mining Anomalies Using Traffic Feature Distributions. In *ACM SIGCOMM*, Aug. 2005.

[8] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. In *ACM SIGMETRICS*, pages 145–156, 2006.

[9] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting, 2004.

[10] N. Spring, R. Mahajan, and T. Anderson. Quantifying the causes of path inflation. In *ACM SIGCOMM*, 2003.

[11] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 87–95, 2000.

[12] M. Sung, A. Kumar, L. E. Li, J. Wang, and J. J. Xu. Scalable and Efficient Data Streaming Algorithms for Detecting Common Content in Internet Traffic. In *International Conference on Data Engineering Workshops (ICDEW)*, 2006.

[13] A. Wolman et al. On the scale and performance of cooperative Web proxy caching. In *ACM Symposium on Operating Systems Principles*, 1999.

[14] A. Wolman et al. Organization-based Analysis of Web-Object Sharing and Caching. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, 1999.

[15] K. Xu, Z.-L. Zhang, and S. Bhattacharya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *ACM SIGCOMM*, Aug. 2005.

[16] M. Zhang, B. Karp, S. Floyd, and L. Peterson. Rr-tcp: A reordering-robust tcp with dsack. In *ICNP*, 2003.

[17] Y. Zhu, B. Li, and J. Guo. Multicast with network coding in application-layer overlay networks. volume 22, pages 107–120, Jan. 2004.