Redundancy in Network Traffic: Findings and Implications

Ashok Anand[1] *, Chitra Muthukrishnan*, Aditya Akella* and Ramachandran Ramjee†
*University of Wisconsin-Madison, †Microsoft Research India
{ashok,chitra,akella}@cs.wisc.edu, ramjee@microsoft.com

ABSTRACT

A large amount of popular content is transferred repeatedly across network links in the Internet. In recent years, *protocol-independent redundancy elimination*, which can remove duplicate strings from within arbitrary network flows, has emerged as a powerful technique to improve the efficiency of network links in the face of repeated data. Many vendors offer such redundancy elimination middleboxes to improve the effective bandwidth of enterprise, data center and ISP links alike.

In this paper, we conduct a large scale trace-driven study of protocol independent redundancy elimination mechanisms, driven by several terabytes of packet payload traces collected at *12* distinct network locations, including the access link of a large US-based university and of 11 enterprise networks of different sizes. Based on extensive analysis, we present a number of findings on the benefits and fundamental design issues in redundancy elimination systems. Two of our key findings are (1) A new redundancy elimination algorithm based on Winnowing that outperforms the widely-used Rabin fingerprint-based algorithm by 5-10% on most traces and by as much as 35% in some traces. (2) A surprising finding that 75-90% of middlebox's bandwidth savings in our enterprise traces is due to redundant byte-strings from within each client's traffic, implying that pushing redundancy elimination capability to the end hosts, i.e. *an end-to-end redundancy elimination solution*, could obtain most of the middlebox's bandwidth savings.

Categories and Subject Descriptors: C.2.m [Computer Communication Networks]: Miscellaneous

General Terms: Algorithms, Measurement.

Keywords: Traffic Redundancy, Traffic Engineering.

1. INTRODUCTION

Network traffic exhibits large amount of redundancy when different users on the Internet access same or similar content. Many diverse systems have explored how to eliminate this redundant content from network links and improve network efficiency. Several of these systems operate at the application- and object-levels. For example, Web proxy caches [27] and the more recent P2P caches [6] store frequently accessed objects and serve repeated requests from cache. Dictionary-based algorithms such as GZIP [30] remove duplicate bytes from within objects. Numerous studies have explored the effectiveness of such application-layer and object-level systems and have developed algorithms for optimizing their designs (see, for example, [28, 29, 17, 30, 15]).

In recent years, a new class of *protocol-independent redundancy elimination* algorithms have been developed that can identify and remove strings of bytes that are repeated across *network packets*. First pioneered by Spring et al. [26], and later developed into "WAN Optimization" middleboxes by multiple vendors [7, 8, 3, 1, 5, 4], these approaches operate in a transparent fashion below the application layer and suppress any repeated strings of bytes that appear on a network link. Since these approaches subsume multiple object-level and application-specific techniques they are both more effective at removing redundant bytes and also more flexible to use.

Protocol-independent redundancy elimination is becoming increasingly popular [2, 9]. Redundancy elimination middleboxes are being widely deployed to improve the effective bandwidth of network access links of enterprises and data centers alike, and for improving link loads in small ISP networks. Driven by the high effectiveness of these systems, recent efforts have also considered how to make these systems first-class network entities. For instance, Anand et al. proposed that protocol-independent redundancy elimination be deployed on a wider scale across multiple network routers enabling an IP-layer redundancy elimination service [12]. This expands the benefits of redundancy elimination to multiple end-to-end applications. It could also enable new routing protocols [12]. Li et al. similarly considered how to modify Web applications by introducing data markers that help in-network redundancy elimination mechanisms to identify and remove redundancy more effectively [19]. Ditto [16] similarly proposes to use application-independent caching at nodes in city-wide wireless mesh networks to improve the throughput of data transfers.

As redundancy elimination techniques become more widely deployed and more tightly integrated into network infrastructure and protocols, it becomes crucial to understand the benefits, trade-offs and design issues in these systems. Unfortunately, there is very little insight into even the basic issues underlying these systems today: What is the optimal level of performance one can expect from protocol-independent packet-level techniques? Do currently popular techniques [26] perform close to optimal or do better-performing algorithms exist? When is network-based redundancy elimination—which is the currently popular model of deployment and usage—most effective and under what situations do end-to-end approaches offer better cost-performance trade-offs? What fundamental traffic

[1]A part of this work was done while the author was doing an internship at Microsoft Research, India.

redundancy patterns drive the design and bound the effectiveness of redundancy elimination systems? Understanding these issues is central not only for improving the design and ensuring cost-effective usage of current redundancy elimination techniques, but also in guiding future redundancy elimination-based network architectures such as those proposed in [12, 19, 16].

In this paper, we conduct a large scale trace-driven study of protocol independent redundancy elimination techniques to shed light on some of the above fundamental issues. Our large-scale study is driven by packet payload traces collected at *12* distinct network locations, including the access link of a large US-based university and of 11 enterprise networks of different sizes (up to 100+ IPs). These traces, described in §3 and Table 1, span diverse user populations and cover multiple days and terabytes worth of traffic, giving us a comprehensive view into traffic redundancy and the effectiveness of redundancy elimination.

Our study consists of three parts. In the first part, presented in §4, we focus on the redundancy elimination algorithms. We compare the popular algorithm due to Spring et al. with a new algorithm based on Winnowing [25], as well as with a hypothetical algorithm that can identify the optimal amount of redundancy. In the second part, presented in §5, we study the macroscopic benefits of redundancy elimination, focusing on whether and how redundancy elimination improves the average and peak utilizations for different networks' links. We also examine the impact on temporal variations in traffic volumes and compare redundancy elimination against protocol-specific approaches such as HTTP object compression and caching. Finally, in §6, we take an in-depth microscopic look into network traffic redundancy. We study a variety of issues which have direct implications on the design of redundancy elimination systems. These include various properties of redundant content ranging from origin of redundant strings, prevalence and importance of partial packet matches, and temporal and frequency distribution of the redundant strings. In all cases, we study traces from the full set of 12 network locations to ensure that our empirical insights are broadly applicable.

We present a full list of the findings and implications of our study in §7. Two of our key findings are *(1) A new redundancy elimination algorithm that provides more uniform selection of chunks for indexing outperforms the widely used fingerprint selection algorithm proposed in [26] by 5-10% on most traces and by as much as 35% in some traces*. (2) A surprising finding that 75-90% of middlebox's bandwidth savings in our enterprise traces is due to redundant chunk matches from within each client's traffic. This implies that pushing redundancy elimination capability to end hosts, i.e. *an end-to-end redundancy elimination solution, could obtain most of the middlebox's bandwidth savings, diminishing the need for deployment of expensive middleboxes in enterprises* and sidestepping the attendant problems such as encryption. Some of our other observations include: (1) While average bandwidth savings of redundancy elimination can be as high as 60%, it may not result in similar savings in peak usage, and more generally, the burstiness of traffic after redundancy elimination is not commensurately reduced. (2) Redundant segment matches follow a Zipf-like distribution with the implication that small caches can capture bulk of the bandwidth savings, while it takes increasing amount of cache size and indexing effort in order to obtain incremental gains.

2. RELATED WORK

Several papers have looked at various aspects of redundancy in network traffic. Some of the approaches have focused on object-level redundancy, while the more recent ones examine packet-level redundancy. Next, we compare these papers with our work.

Object-level approaches. Object-level caching and its effect on wide-area performance has been extensively studied. Several studies have examined that user accesses of Web objects are Zipfian in nature [14]. Other studies have considered the impact of these access patterns on caching and the effectiveness of caching. For instance, Wolman et al. considered the sharing of Web documents among users at the University of Washington [29]. They showed that users are more likely to request objects that are shared across departments than objects that are only shared within a department. In follow-up work, Wolman et al. [28] 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. Redundancy elimination during file downloads has also received a large amount of attention [21, 22]. The basic idea behind these approaches is to divide files into content-based chunks and download only those chunks that are not already present locally. In contrast with these object-level approaches, our work takes a more information centric view by focusing on the repetition in content within packets. We study the popularity of strings contained in packets, how the packet level content is shared by different pools of users, and whether similar evidence of diminishing returns exists when we consider greater amounts of caching and sharing of packet level content.

Packet-level approaches. As mentioned in §1, Spring et al. developed the first protocol independent approach for identifying redundant bytes in network traffic [26]. We describe this approach and other candidate approaches in more detail in §4. Applying this approach to traces collected at an enterprise network, Spring et al. found that, on average, 20% of the bytes were redundant in the inbound direction, and 50% were redundant in the outbound direction [26]. They also showed that protocol-independent techniques are more effective than object level caching. In our work, we consider a much broader data set including both directions of a University access link and 11 enterprise networks of various sizes. Our work also goes beyond just identifying the amount of redundancy identified by a specific packet-level algorithm and examines several fundamental issues. These include: what is the amount of redundancy that a packet-level approach can identify in the best case? How close to optimal do practical algorithms get? What benefits do packet level techniques offer in managing link loads? What trade-offs do they impose? What are the fundamental characteristics of the duplicate strings? How do these characteristics impact the design of practical redundancy elimination mechanisms?

More recently, Anand et al. [12] explored the benefits of deploying Spring et al.'s mechanism on all Internet routers. Such a deployment would enable redundancy elimination as a primitive service that is accessible to all end-to-end flows. They showed that such a service can improve the performance of end-to-end flows, improve link loads everywhere and also enable new routing and traffic engineering mechanisms. Our work informs the design of such wide-spread redundancy elimination services. In particular, our work shows where middleboxes are beneficial, how to design those caches and how much cache to provision to obtain reasonable redundancy elimination. Our observations regarding the sources of redundancy and the temporal and spatial variations could be leveraged when designing new "redundancy-aware" protocols such as the ones outlined in [12].

WAN Optimization. Bandwidth requirements of network entities such as small ISPs and enterprise networks have seen steep increases in recent years. However, augmenting WAN capacity to meet the growing demand can be an expensive proposition. To meet these requirements, small ISPs and enterprises are increasingly turning toward WAN optimization middleboxes which can simultaneously improve the effective capacity of network links and lower link usage costs. Different vendors like Riverbed, Cisco, Juniper etc. are involved in this increasingly competitive market (see [7, 8, 3, 1, 5, 4] for descriptions of the products, and a longer list of products at [10]). The core techniques used by these optimizers are similar to those in packet-level redundancy elimination such as the ones we describe in §4. In addition, some products employ domain-specific data compression by effectively representing known data patterns, protocol specific optimizations like protocol spoofing by bundling multiple requests of chatty applications to one, etc. While these products are presumed to offer substantial benefits at the locations where they are deployed, very little is known in the open literature about the quantitative extent of benefits, the underlying tradeoffs involved in using these approaches, and the challenges and design considerations in implementing WAN optimization. Our measurement observations shed light on these important issues.

3. DATA SETS

Our empirical study is based on full packet traces collected at several distinct network edge locations. One is from a large university's access link to the commercial Internet, while the others are from access links of enterprises of various sizes. The key characteristics of our traces are shown in Table 1.

Enterprise Traces. We monitored access links at 11 corporate enterprise locations and collected several days worth of traffic going into and out of these sites. We classify the enterprises as small, medium or large based on the number of internal host IP addresses seen (less than 50, 50-100, and 100+, respectively) in a typical 24 hour trace at each of these sites. While this classification is somewhat arbitrary, we use this division to study if there are redundancy properties that are dependent on the size of an enterprise. Note that the total amount of traffic in each trace is also approximately correlated to the number of host IP addresses, though there is a large amount of variation from day to day. Typical incoming traffic numbers for small enterprises were about 0.3-3GB/day, for medium enterprises were about 2-12GB/day and for large enterprises about 7-50GB/day. The access link capacities at these sites varied from a few Mbps to several tens of Mbps. Note that even the largest enterprise site in our trace is one or more orders of magnitude smaller than the University site in terms of number of IPs, traffic or access link capacity. Finally, the total volume of enterprise network traffic collected and analyzed is about 3TB.

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 50,000 users. We logged entire packets (including payloads) going in either direction on the access link. Due to some limitations of our collection infrastructure (our disk array was unable to keep up with traffic rate at peak utilization), we were only able to log traffic from one direction at a time. Thus, we alternatively logged a few minutes of traffic in each direction.

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 3 traces per hour for either direction, resulting in a total of 147 traces for each 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, 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, during Jan 23-25, 2007, we collected 1.1TB worth of traffic during different hours between 10am and 7pm. Again, we alternated between the incoming and outgoing directions; each trace spanned ≈ 600 seconds on average. Henceforth, we shall use the terms *Univ-In-long* and *Univ-Out-long* to describe these traces.

Other uses of the traces. We also focus on the subset of the University traces involving traffic to and from a certain high volume /24 prefix owned by the University. Several of the most popular Web servers in the University are located on this /24. We use these University trace-subsets as potential logs for evaluating how redundancy suppression may help data centers.

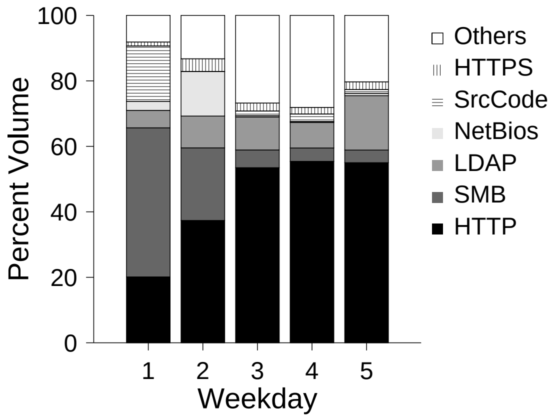


Figure 1: Protocol distribution over five days at the incoming link of a large enterprise (starting with HTTP at the bottom)

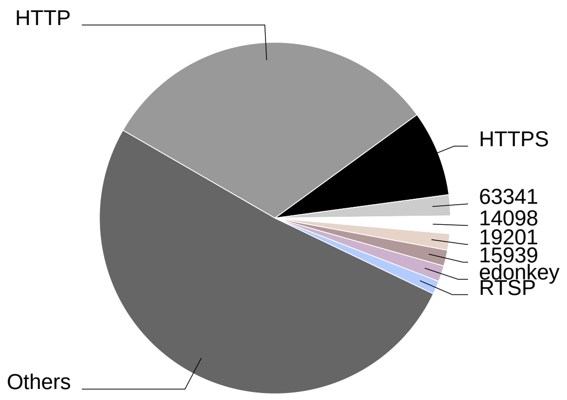


Figure 2: Protocol distribution at University outgoing link

In order to provide a flavor of these traces, we show the trace composition by protocols across five week days for a large enterprise trace in Figure 1. These protocols were identified using TCP port numbers. This figure highlights the significant differences in network access link traffic since the 1999 enterprise traces that were analyzed by Spring et al. in [26]. First, while [26] observed a dominant fraction of traffic comprised of HTTP (64% of incoming traffic), we see HTTP traffic is significant but not dominant today, with large variations seen from day to day (20-55%). Second, while [26] hardly observed any file transfer traffic (FTP 1.9%), the traffic over our enterprise access links comprise a significant amount (25-70%) of file transfer traffic (SMB, NetBios, Source

Trace name	Description	Dates/Times	Span of each trace	Number of traces	Total Volume (GB)
Small Enterprise (3 sites, 0-50 IP)	Inbound/Outbound	1PM on 07/28/08 to 7PM on 08/08/08	24 hours	8	100
Medium Enterprise (5 sites, 50-100 IP)	Inbound/Outbound	1PM on 07/28/08 to 7PM on 08/08/08	24 hours	8	400
Large Enterprise (2 sites, 100+ IP)	Inbound/Outbound	1PM on 07/28/08 to 7PM on 08/08/08	24 hours	8	500
Large Research Lab (1 site, 100+ IP)	Inbound/Outbound	10AM on 06/16/08 to 10AM on 07/03/08	24 hours	17	2000
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 to 7:00 PM, between 01/23/07 and 01/25/07	$\approx$600s worth of traffic	27	550
Univ-Out-long	Outbound traffic at university access link	10:00 AM to 7:00 PM, between 01/23/07 and 01/25/07	$\approx$600s worth of traffic	27	550

Table 1: Characteristics of the data traces gathered from 12 sites

Code Server, etc.). These file transfer traffic were directed to other branch offices around the globe. This is likely due to a significant shift in enterprise management approach in the last few years where increasingly servers are centralized in a few locations/data centers in order to save administrative expenses. Finally, Figure 2 shows protocol composition at a university outgoing link, and the traffic characteristics here are quite different from the enterprise - HTTP is significant at 36% and a large portion of traffic, likely peer-to-peer, is classified as others (note that the edonkey control traffic itself is significant and identified separately).

4. ALGORITHMS FOR REDUNDANCY ELIMINATION

Broadly speaking, redundancy in network packets can be eliminated in two ways: 1) detection and removal of redundant strings across different packets, also called *redundancy suppression* and 2) redundancy elimination within a packet using *data compression*. We first discuss algorithms for performing redundancy suppression, and then briefly discuss compression.

4.1 Redundancy Suppression

To date, middlebox-based techniques for redundancy suppression [12, 26] rely on the approach proposed by Manber [20], in the context of identifying similar files in a file system. We first describe this approach, referred to as MODP, and then describe an alternative approach called MAXP. We then describe a technique for estimating the performance of an optimal redundancy suppression algorithm and present performance comparisons of MODP and MAXP with respect to the optimal.

Before we describe the MODP and MAXP algorithms, we first describe the overall approach behind redundancy suppression for network traffic, first proposed in [26]. Given a cache/dictionary of past packets, redundancy suppression techniques need to identify contiguous strings of bytes in the current packet that are also present in the cache. This is accomplished by identifying a set of representative "fingerprints" for each packet and then comparing these fingerprints with a "fingerprints store" that holds the fingerprints of all the past packets in the cache. The fingerprints serve as "random hooks" into portions of the packet payload and are used to find redundant content. For each fingerprint of the packet that is matched against the store, the matching packet is retrieved and the matching region is expanded byte-by-byte in both directions to obtain the maximal region of redundant bytes. Once all matches are identified, the matched segments are replaced with fixed-size pointers into the cache, thereby suppressing redundancy. Finally, the cache and fingerprint store are updated with the new packet. If the packet cache is full, the earliest packet in the store is evicted and all its fingerprints are freed. The key difference between the MODP and MAXP algorithms is simply in how the representative fingerprints are computed, which we describe next.

4.1.1 MODP

In this algorithm, Rabin fingerprints [23] of sliding windows of w contiguous bytes of each packet payload are computed. Parameter w represents the minimum match size of interest. Smaller w would help identify more matches at a potential cost of missing larger matches. Typical values for w range from $12 - 64$ bytes.

For a packet with S bytes of payload, $S \geq w$, a total of $S - w$ fingerprints are generated. Since $S >> w$, the number of such fingerprints is approximately the same as the number of bytes in the packet.

Since it is impractical to store all these fingerprints, a fraction $1/p$ of fingerprints are chosen whose value is **0 mod p** (p can be chosen as a power of two for ease of computation). In this way, fingerprints are chosen independent of their position and is thus robust to reordering and insertions/deletions. In cases where the MODP selection criteria does not choose even a single fingerprint from a given packet, we explicitly enforce that at least one fingerprint is chosen per packet.

Parameter p controls the memory overhead of the fingerprint store with typical values of p ranging from 32 to 128. For example, in [12], using 16 fingerprints per 1500 byte packet (or $p \approx 90$) results in indexing memory overhead of 50% the cache size.

4.1.2 MAXP

One shortcoming with the MODP approach is that the fingerprints are chosen based on a *global* property, i.e., fingerprints have to take certain pre-determined values to be chosen. While this would result in the desired fraction of fingerprints being chosen across a large packet store, on a per-packet basis, the number of fingerprints chosen can be significantly different and not enough sometimes.

In order to guarantee that adequate number of fingerprints are chosen uniformly from each packet, a *local* technique such as winnowing [25] is essential. Winnowing, similar to the work by Manber [20], was also introduced in the context of identifying similar documents and can be easily adapted to identifying redundancy in network traffic. The key idea behind winnowing is to choose those fingerprints that are **local-maxima (or minima) over each region**

of p bytes, thus ensuring that one fingerprint is selected over every segment of a packet.

Our MAXP fingerprint selection algorithm is based on the local-maxima based chunking algorithm designed for remote differential compression of files [13]. This algorithm is similar to winnowing but has the advantages of imposing a lower bound on chunk length and lower computational overhead since the local-maxima is computed using the bytes directly as digits (rather than computing a hash first before the minima computation in winnowing). The details of the algorithm can be found in [13].

While the authors of winnowing show that, in the case of web files, the MODP approach can result in no hashes being picked for approximately $30K$ of non-whitespace characters, it is not immediately clear whether similar deficiencies of MODP will be visible in our setting, where the network traffic comprises a mix of protocols.

4.1.3 Optimal

For a given minimum match size w, the optimal algorithm for redundancy elimination would require that every fingerprint be stored for potential future matches. Since the number of fingerprints is of the order of number of bytes in the packet store, the memory overhead of indexing every fingerprint is simply impractical.

We devise an alternate approach based on bloom filters in order to estimate the upper bound of the optimal algorithm. Instead of indexing every fingerprint, we store the fingerprints in an appropriately sized bloom filter. We then identify fingerprint matches when the bloom filter contains the fingerprint. Given all the matched regions in the packet, we can identify the optimal set of matches that maximizes redundancy elimination.

While bloom filters are susceptible to false positives, we choose 8 hash functions and a bloom filter size (in bits) that is 16 times the number of bytes in the packet store so that the false positive rate is under 0.1% [17]. One drawback of using a bloom filter is that even though we know a match exists with high probability, we do not know the location of the match. This is problematic in the case of overlapping matches of two or more w byte strings, since we are unsure whether the overlapping matches correspond to the same location in the packet store (resulting in maximum redundancy elimination) or not. We optimistically assume that overlapping matches in the bloom filter would also overlap in the packet store and thus the resultant computation provides an upper bound in the redundancy suppression performance of the optimal algorithm.

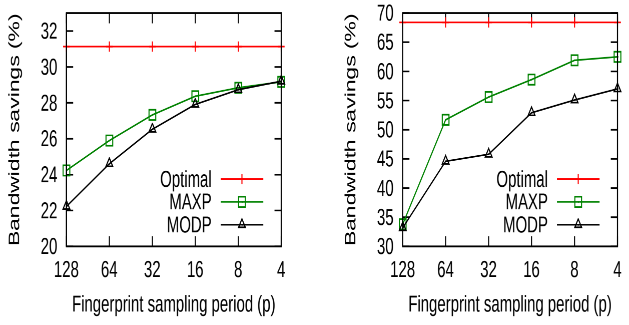
Finally, if we need to model packet eviction from a full cache, a counting bloom filter can be used instead of a simple bloom filter.

4.1.4 Comparison

In this section, we present a comparison of the bandwidth savings of the MODP and the MAXP algorithms with respect to the upper bound of the Optimal algorithm. Given the high computational requirements for obtaining the results presented in this section, we use only small representative subsets (averaged over several GBs/hours worth of traffic) of the overall traces. While we present extensive trace evaluations of MODP and MAXP in the next section, here our goal is to evaluate their performance characteristics relative to the optimal.

We define bandwidth savings as the ratio of number of bytes saved due to redundancy elimination as compared to the original network traffic. While computing savings, we take into account all overheads including packet header overheads and the shim headers [12] necessary to encode the pointers in the MODP/MAXP algorithms.

We use a minimum match window size, w, of 32 bytes (the impact of w is discussed in §6) and vary the sampling period p from



(a) Trace 1 (b) Trace 2

Figure 3: Comparison of MODP, MAXP, and Optimal

128 to 4. Results from two enterprise traffic traces are shown in Figure 3(a) and (b), respectively.

First, notice that the performance of MODP and MAXP improves as we decrease the sampling period, because we are indexing more and more fingerprints. We see that their performance approaches the upper bound of the optimal at $p = 4$. Second, we see that MAXP outperforms MODP by up to 10% (2% absolute) in Figure 3(a) while we see a much wider performance gap of up to 20% (10% absolute) between MAXP and MODP in Figure 3(b). In both traces, the selection of fingerprints by MODP is more clustered while that of MAXP is much more uniform; in the case of MODP, the next fingerprint is selected within 10 bytes of the last selected fingerprint's location in more than 30% of the cases for $p = 32$. In contrast, MAXP gives much more uniform selection; the gap is less than 10 bytes in less than 2% of the cases. The clustering impacts MODP more in the second trace than in the first trace; that is, the number of redundant string matches and average match size is similar in trace 1 between MAXP and MODP but quite different in trace 2. This highlights the trace driven nature of MODP's performance and the advantage of uniform fingerprint sampling approach of MAXP. As we shall see in the next section, there are traces where MAXP outperforms MODP by up to 35%.

Finally, we choose $p = 32$ as the default sampling period in the rest of the paper. While $p = 4$ delivers the best performance, the indexing memory overhead at $p = 4$ is approximately 10X the cache size and is not a desirable operating point. A choice of $p = 32$ delivers performance within 10% of $p = 4$ (10-20% of optimal upper bound) and has a memory overhead roughly comparable to the cache size.

4.2 Compression

Clearly, irrespective of whether redundancy suppression is being implemented or not, each packet can be compressed by a compression algorithm such as the deflate algorithm used in gzip. The deflate algorithm replaces repeated strings with pointers and further uses huffman coding to efficiently encode commonly occurring symbols. While the authors in [26] compared redundancy suppression with deflate, they did not consider a) the benefits of aggregating packets within a small time window, say 10ms, before applying compression[1] and b) the benefits of applying compression after redundancy suppression and whether there are any complementary gains. We evaluate these aspects in the next section.

[1]It is well-known that deflate does not compress very small packets well as it needs to build a dictionary before the benefits of compression kick in. However, note that neither MODP nor MAXP benefits from aggregation since the minimum match size $w << S$, the packet size.

Site	#IP	Trace size(GB)	GZIP	GZIP+10ms	MODP	MAXP	MAXP@4xCACHE	MAXP+GZIP+10ms	MAXP-outgoing
Small Enterprise									
1	44	5.8	19	25	35	37	39	41	61
2	31	21.7	7	10	40	54	59	61	54
3	18	0.2	15	21	37	38	38	41	41
Medium Enterprise									
1	54	7.7	6	8	15	16	17	19	51
2	72	7.1	13	18	33	35	39	41	43
3	79	10.6	12	16	25	27	30	34	44
4	79	14.3	9	12	18	19	21	24	25
5	61	4.4	9	13	27	28	31	32	44
Large Enterprise									
1	122	17.5	6	8	15	16	20	19	54
2	142	3	8	12	23	24	26	27	44
3	160	31	10	13	22	23	31	27	34

Table 2: Bandwidth savings across different enterprise sites in percentage

Traffic Type	#IP	Trace size(GB)	GZIP	GZIP+10ms	MODP	MAXP	MAXP@4xCACHE	MAXP+GZIP+10ms
Incoming	9360	22	4	5	9	9	12	10
Outgoing	-	22	3	4	11	12	15	14
Outgoing /24	29	2.3	2	3	33	41	48	43

Table 3: Bandwidth savings across University traffic in percentage

5. MACROSCOPIC VIEW: CORE BENEFITS

In this section, we explore the core benefits of deploying redundancy elimination on either end of the WAN access link to the enterprise/university. Our analysis across diverse settings gives a comprehensive view of when (and to what extent) redundancy elimination techniques help.

We first evaluate the average bandwidth savings due to redundancy elimination using the various algorithms described in §4, individually, and in combination. We then examine the temporal variability of the savings. Finally, we examine redundancy characteristics for different protocols and then focus on HTTP in order to answer the following question: how does protocol-independent redundancy elimination compare with protocol-specific techniques such as compression of HTTP objects and the use of web caches?

5.1 Bandwidth Savings

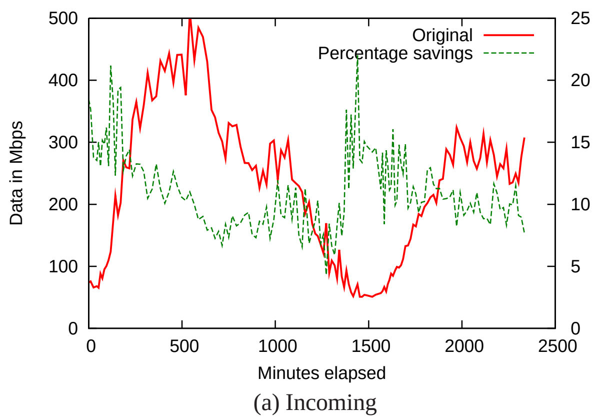
Tables 2 and 3 present the average bandwidth savings using the different redundancy elimination algorithms described in §4 for the enterprise and university traces, respectively.

Let us first focus on the enterprise results in Table 2. We classify enterprise sites as small, medium, and large based on the number of host IP addresses seen within the enterprise. While the total trace size is mostly proportional to the number of IP addresses, we can see several outliers such as small enterprise site 2 with over 20GB of traffic while large enterprise site 2 with only 3GB of traffic. Except for the last column which presents savings on outgoing traffic, the rest of the results are for traffic incoming to the enterprise, since this is the dominant portion of traffic ($\approx$ 80-90% of trace size). We use $w = 32$, $p = 32$ and a default cache size of $250MB$ (plus approximately $250MB$ for indexing overhead), which corresponds to roughly 2 to 10 minutes of traffic at peak utilization for the different sites. We later show that even such a small cache size is sufficient to achieve reasonable benefits.

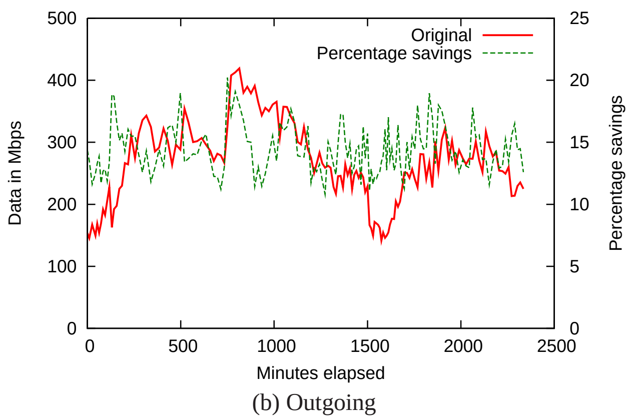
We make several observations from the table. First, while performing GZIP compression per packet provides some gains, the gains are only half or less of the gains from the MODP or MAXP algorithms. Second, while aggregating packets helps improve GZIP gains, an aggregation latency of $10ms$ still does not provide substantial gains. Third, MAXP outperforms MODP by 5-10% on most of the traces (1-2% absolute bandwidth gains) and, in some traces such as small enterprise 2, performs as much as 35% better than MODP (14% absolute). This highlights the importance of the uniform sampling approach of MAXP since one could very well hit a large stream of bytes where the MODP algorithm misses out on significant redundancy elimination opportunities. Fourth, increasing the cache size by 4X provides incremental gains of 0-35% (0-8% absolute), with large enterprises benefiting the most. Fifth, applying GZIP with 10ms aggregation after MAXP provides gains of 8-26% (3-7% absolute) over MAXP alone, delivering, in most cases, comparable savings as using MAXP alone at 4X the cache size. Thus, compression can be effective in complementing the gains obtained via redundancy suppression. Sixth, the average savings are generally higher for the small/medium enterprises as compared to large enterprises. Finally, the savings on outgoing links (last column) are generally higher than the savings on incoming links.

Examining the results for the university traces in Table 3, the broad observations made earlier for the enterprise traces hold good. Here, we only highlight a few salient points. Note that the incoming and outgoing traffic are roughly similar, in size, unlike the skewed incoming dominant case for the enterprise. Thus, we present results for incoming and outgoing separately. Overall, the savings are in the 10-15% range, continuing the trend seen earlier of larger sites resulting in lower average savings. Focusing specifically on the outgoing traffic from high volume /24 prefix that hosts popular web servers, we see that the trace demonstrates significant savings of over 40%. Finally, note that MAXP significantly outperforms MODP by 22% in this trace, again illustrating the advantage of uniform sampling of the MAXP algorithm.

In summary, results from this section demonstrate that redundancy elimination can bring down the average utilization of access links substantially. *The benefits range from 10% to 60% in average bandwidth savings, with smaller sites generally achieving higher savings. The MAXP algorithm outperforms the MODP algorithm, sometimes substantially by up to 35%, and applying packet level gzip compression with 10ms aggregation after MAXP provides further gains of up to 26%.*



(a) Incoming



(b) Outgoing

Figure 4: Volume, in Mbps, of original and compressed data

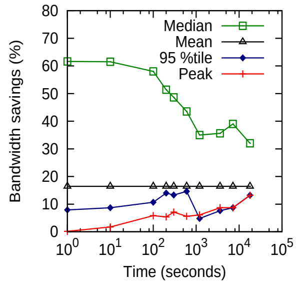
5.2 Temporal variations

While average savings provide a good measure of the overall effectiveness of redundancy elimination, it can hide significant temporal effects. In this section, we study the temporal variability of savings due to redundancy elimination.

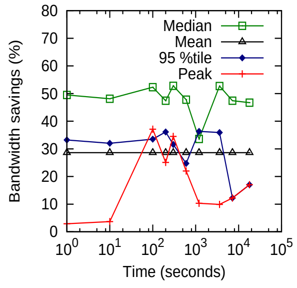
In Figure 4(a) and 4(b), we plot the volume in Mbps of all bytes for the Univ-In-60s and Univ-Out-60s traces (y1-axis). We overlay the fraction of redundancy in the same figure as well (y2-axis). We see a slight negative correlation between the fraction of redundancy and the volume of traffic for inbound traffic (Figure 4(a)). In contrast, there is a slight positive correlation for the outbound traffic (Figure 4(b)). We also observed a slight positive correlation between link utilization and redundancy fraction in the outgoing /24 trace (not shown). The correlation between traffic load and redundancy can play a vital role in terms of bandwidth savings at peak traffic periods. In order to quantitatively study the impact of temporal variability with respect to bandwidth savings, we define two metrics:

- *Peak and 95th-percentile savings*: Since links are sometimes charged and/or provisioned based on peak or 95th-percentile traffic load [18, 24], we compare the peak and 95th percentile savings with the average savings available in the trace. We compute these measures over a range of time buckets starting from 1 second to 5 hours, and study how they vary both with respect to these time buckets as well as compared to the average savings.
- *Burstiness*: We use the burstiness metric as computed using wavelet based multiresolution analysis (MRA) [11] to study how redundancy elimination impacts traffic burstiness at various timescales. MRA-based energy plots depict the variance or burstiness of traffic at different timescales and, in general, one would expect compression to help reduce burstiness in traffic.

In Figure 5(a) and (b), we plot the mean, median, 95th percentile and peak savings over buckets of different timescales (in log scale) from a 24 hour trace for a large-sized and medium-sized enterprise, respectively. Examining the figure for the large enterprise, the results are not very encouraging — the median savings is significantly higher than the mean savings while the peak savings is generally lower than the mean savings, for almost the entire range. This implies that redundancy elimination is negatively correlated with load, i.e., at peak hours there is less redundancy than in lean hours. The 95th-percentile savings measure is somewhat better than the peak measure and approaches the mean savings for time units of 100-600 seconds. The results for the medium-sized enterprise,



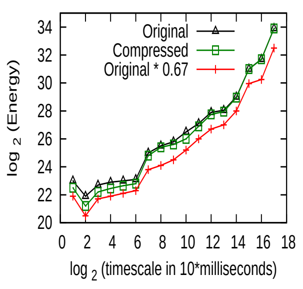
(a) Large enterprise



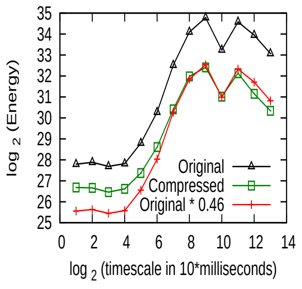
(b) Medium enterprise

Figure 5: Savings in median, 95-percentile and peak usage

while also broadly similar to the large enterprise case, is better with 95th-percentile savings outperforming mean savings over a large range of time values. In case of University, we used Univ-In/Out-60s traces for this evaluation as indicative of few 60 second samples. We observed that peak savings and 95th-percentile savings were better than mean savings for 60 second samples. For Univ-outbound trace, peak savings and 95th percentile savings were 16% and 14.5% as compared to mean of 12%, supporting earlier observation of slight positive correlation of redundancy savings with utilization. For incoming trace, the differences were not significant from mean savings of 10%. In conclusion, examining the average savings over an interval does not depict a true measure of the temporal variations in savings and we find that redundancy elimination may be negatively correlated with load, resulting in lower peak savings as compared to average savings.



(a) Enterprise



(b) Univ-outgoing/24

Figure 6: Burstiness of original and compressed traffic

We now examine the burstiness metric derived from wavelet-based multiresolution analysis using an energy plot, which depicts base-2 log of the energy (variance) against the base-2 log of the time scale. Figures 6(a) and (b) depict the energy plot for the original traffic, the compressed traffic and a hypothetical trace where the original traffic is compressed *uniformly* using the average compression savings value for a large enterprise and the university outgoing /24 traces, respectively. Note that the time scale ranges from 10 milliseconds to 1 hour (1 minute) for the enterprise (university) traces. The differences between the two curves are quite obvious — we notice that the compressed traffic does not reduce the burstiness of original traffic significantly for most of the range of the time scales depicted in enterprise case (the two curves practically overlap compared to the uniform compression curve) while burstiness is reduced significantly in the university trace, especially in the 1 minute timescale. This lack of reduction in burstiness in the enterprise trace is not surprising, given our observation that the median (peak) savings are generally higher (lower) than the mean savings across different timescales in the traces. On the other hand, as mentioned earlier, the positive correlation between link utilization and redundancy in the outgoing /24 trace, helps reduce its overall burstiness.

One caveat with this analysis is that we simply compute burstiness of the original and compressed traces, assuming that the arrival process does not change. However, redundancy elimination may impact TCP's congestion control behavior which can change the arrival process and result in different burstiness values, especially at the smaller timescales. We plan to study this issue in more detail by replaying the traces using TCP over a testbed and re-evaluating the burstiness metric.

In conclusion, our temporal variability analysis presents a mixed picture of the benefits of redundancy suppression. *While the average savings of up to 60% are significant, peak savings can be significantly lower than the average savings. We do find that the 95th-percentile savings is closer to mean savings*, at least over certain time scales, which may be helpful in curtailing usage costs in certain situations. Finally, *the overall traffic burstiness is not significantly reduced in the enterprise case*, implying that redundancy elimination is not too helpful in making traffic more predictable or enabling more effective traffic engineering on enterprise access links.

5.3 Redundancy in protocols

Port #	Protocol	Univ-In-60s		Univ-Out-60s	
		% of bytes	*% redundancy*	*% of 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
		Large Enterprise-In		**Large Enterprise-Out**	
445	SMB	45.46	21.40	45.44	17.18
80	HTTP	16.8	29.45	14.41	76.31
139	NetBios	2.88	7.98	0.8	36.52
389	LDAP	4.85	44.33	12.5	71.68
-	Src Code Crtl	17.96	50.32	0.1	72.31

Table 4: Redundancy in key protocols

In Table 4, we show the redundancy that we observe in popular network protocols that traverse the WAN access link in the university and a large enterprise trace. For each protocol, we show the fraction of total bytes that belong to the protocol, and the fraction of redundancy in the protocol's payload. Our observations here are different from Spring et al. [26] in a few key ways. For example, Spring et al. found HTTP traffic to be highly redundant ($\sim 30\%$ after excluding web caching, $\sim 54\%$ overall) and SMTP traffic to be modestly redundant ($\sim 20\%$). In contrast, we see that the redundancy in SMTP traffic is much higher (70% in the outgoing trace), while the redundancy in HTTP traffic is lower (16% and 32% in university and enterprise traces, respectively, using weighted incoming+outgoing bytes). The reduction in HTTP redundancy, as compared to the results reported in [26], is likely due to the increasing usage of port 80 for all types of traffic such as media streaming, games, etc. We also note that 5% of all bytes belong to HTTPS and since the HTTPS payload is encrypted, it shows minimal redundancy. Also, note the mix of protocols in the enterprise trace is quite different from what was observed in the enterprise traces in [26], where the top three protocols were HTTP (64%), RTSP (7%) and Napster (3%).

The changing composition of protocols and also the evolution of what comprises traffic over a well-known port such as port 80, argues for a protocol-independent redundancy elimination solution, assuming it performs as well or better than alternative protocol-specific solutions. We next focus on redundancy in HTTP and compare protocol-independent redundancy elimination with object-level compression and caching.

Trace	Object-GZIP	MAXP	MAXP+GZIP+10ms
Univ-In-long	7.69	10.94	14.93
Univ-Out-long	10.1	20.52	23.75
Univ-Out/24	6.25	53.49	54.69
Large Enterprise	24.45	29.45	34.1

Table 5: Redundancy in HTTP traffic

5.3.1 Redundancy in HTTP

In order to estimate the performance of HTTP with object-level compression, we reverse-engineer HTTP object-level compression from the network-level traces. We first parse each network trace to extract different flows, and then use simple pattern matching to identify the start of HTTP objects within each flow. We are able to extract most of the HTTP objects into separate files in this manner. We then apply gzip compression on each of the objects and compare the compression savings against the savings from redundancy elimination using the MAXP algorithm. The results are shown in Table 5.

Note that the compression savings for GZIP represents an optimistic scenario since many objects may be dynamically generated or composed of latency sensitive parts and thus may not be amenable to GZIP compression over the entire object. Even with this optimistic assumption, we find that redundancy elimination delivers at-least 5-10% additional savings in all the traces and results in significant out-performance (almost 50% additional savings) in the case of university outgoing/24 trace (a popular web site), because of its ability to exploit redundancy across traffic from different users.

Finally, we examined if HTTP object-level caching could help in bandwidth savings. We analyzed cache control headers, pragma responses and other metadata as in [26] to identify the cacheability of HTTP objects. We found that many of the HTTP objects were deemed non-cacheable and only about 5% bandwidth savings would be accrued due to the deployment of a web proxy in the enterprise trace.

Note that, the authors in [26] did not compare their approach against HTTP object-level compression and while they compared against web caching, the cacheable percentage in our case is lower. This is likely due to a combination of higher percentage of dynamically generated web pages and the advertising-related incentives for web sites to serve pages directly to clients.

In conclusion, protocol-independent redundancy elimination is an effective technique for suppressing redundancy and is not impacted by the changing composition of the protocols that traverse the access links of universities and enterprises. Furthermore, in the case of HTTP, we found that *redundancy elimination performs better than both object-level compression and caching, thus diminishing the need for deploying protocol-specific solutions*.

6. MICROSCOPIC VIEW: UNDERSTANDING REDUNDANCY

In this section, we perform in-depth empirical analyses to understand various redundancy characteristics. Our goal is to leverage these empirically observed properties to design effective redundancy elimination techniques.

We first focus on the origins of redundancy: is the observed redundancy mostly due to content common to different users or is it mostly content from within each user's protocol/traffic? If it is mostly content from within each user's protocol/traffic, we could simply employ redundancy elimination at each client-server endpoints, and argue for an end-to-end solution instead of deploying expensive middleboxes. We then examine spatial characteristics of redundant chunks in order to answer the following question: is most of the savings from full packet matches or partial packet matches? More generally, what is the distribution of sizes of the identified redundant chunks? Next, we evaluate the temporal characteristics of matches: are matches mostly from recent packets in the cache, or, more generally, what is the temporal distribution of matches? Finally, we study the hit characteristics of redundant chunks: is redundancy suppression due to a few popular chunks? Does chunk hits have zipf-like characteristics that are seen in web page requests?

One important caveat to note with respect to redundancy characteristics studied in this section: these characteristics are identified in the context of packet-level redundancy elimination approaches, that use caches limited by DRAM sizes (GBs) and history in the order of minutes/hours; these results may not be applicable to file-level redundancy elimination approaches such as [21, 22] that typically store and index terabytes (days/months) of data (history).

6.1 Redundancy: Origins

Given that one of the major claims/advantages of middlebox-based redundancy elimination devices is the ability to leverage redundancy across traffic from different users and flows, it is important to understand the composition of redundancy due to matches between traffic from different users. In general, given a four-tuple of source/destination IP addresses and ports, it would be interesting to know, for each match, how many of these four-tuples were common between the current packet and the matched packet. For example, if most of the savings are due to matches between packets with the same source and destination IP addresses, a purely end-to-end solution would suffice, diminishing the need for middleboxes that are being deployed today.

In this section, we quantify the contribution of matches to bandwidth savings by dividing up the matches into the following five classifications: a) intraflow (match was from a packet with the same four-tuple), b) interflow (match from same source-dest IP but different ports), c) interdst (match from same source IP but different destination IPs), d) intersrc (match from same destination IP but

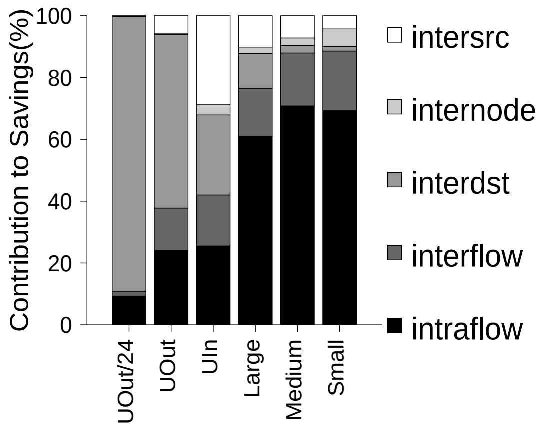


Figure 7: Composition of redundancy (intra/inter-flow/user)

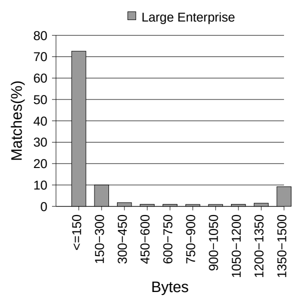
different source IPs), and e) internode (match from different four-tuples). In [26], since the clients in the traces were anonymized, they were unable to correlate redundancy by four-tuples. They found that redundant traffic is mostly (78%) from the same server (interdst) and hypothesized that a server to proxy redundancy elimination would be advantageous as compared to a pure end-to-end solution.

Figure 7 presents the match origin classification for various enterprise (incoming-only) and university traces. Based on the figure, we make the following observations. First, in the case of small and medium enterprises, approximately 90% of savings are due to intraflow and interflow matches (same source-dest IP), implying that a pure end-to-end solution would capture the vast majority of the middle-box savings. In fact over 90% of the interflow matches (not shown) also had the same source port number, indicating that the flows are likely part of the same protocol. Second, while the large enterprise does leverage traffic across different users, it still has about 75% of savings due to matches from flows with the same source-dest IP addresses. Third, in the case of university traces, we see only 10-40% contribution to due intraflow/interflow with a large contribution due to interdst, especially in the case of the outgoing/24 trace, representative of a busy web server. While we do see variations of these contributions in different traces (the enterprise results are averages over several days), these results were generally not sensitive either to cache size used for the redundancy elimination or to time of day, i.e., peak/lean hour (results not shown).

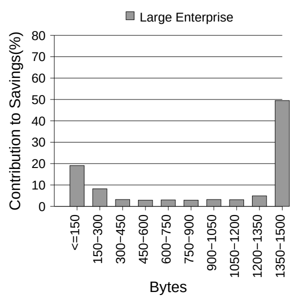
In summary, the key takeaways are: *1) An end-to-end redundancy elimination solution could provide significant portion of the middlebox savings in small/medium enterprises, and to an extent, large enterprises too, diminishing the need for deploying an expensive middlebox-based solution. 2) A middlebox-based solution is more compelling at access links to busy web servers*.

6.2 Redundancy: Spatial view

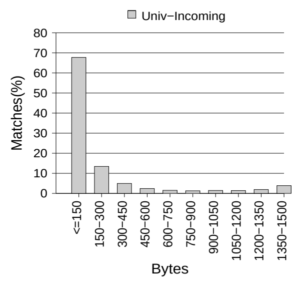
In this section, we seek to understand if the contribution to the observed redundancy comes mainly from full packet matches, or partial packet matches. If the former is true, one can design simpler techniques to index a packet than Spring et al.'s proposal. For instance, rather than store multiple fingerprints per packet, we can store a single hash for the entire packet's content. If the matches are due to partial packet matches, understanding the match length will help advise on the appropriate minimum match size parameter, w, used by the redundancy elimination algorithms.



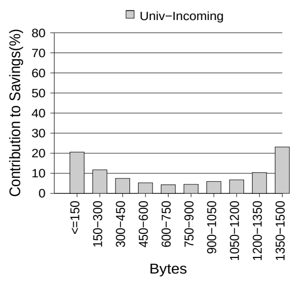
(a) Matches distribution



(b) Savings contribution



(c) Matches distribution



(d) Savings contribution

Figure 8: Match length distribution and contribution to total savings. (a) and (b) are for large enterprise. (c) and (d) are for University inbound traces

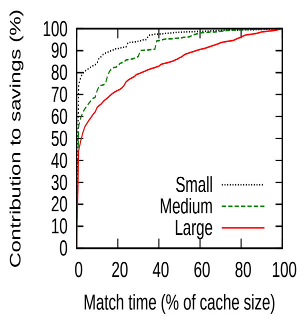
In Figure 8(a), we show the percent of matches for chunks of various sizes for a large enterprise. We see that over 70% of matches are for chunks of size less than 150 bytes and less than 10% of matches are from full 1500 byte packet matches. On the other hand, Figure 8(b) shows the savings contribution of chunks by their sizes and we see that nearly half of the savings are due to large, full packet matches and with approximately 20% due to matches of size less than 150 bytes. Examining at an even finer granularity, we found that approximately 4% of savings was due to 50 bytes or smaller matches (not shown).

Similar results are seen in the University trace as well. In Figure 8(c) for Univ-Incoming trace, we see that around 70% of the matches are for chunks of size less than 150 bytes and full 1500 byte packet matches are less than 5% of the matches. Figure 8(d) shows that less than quarter of the savings are due to large full packet matches, while 20 % of the savings are due to matches of size less than 150 bytes.

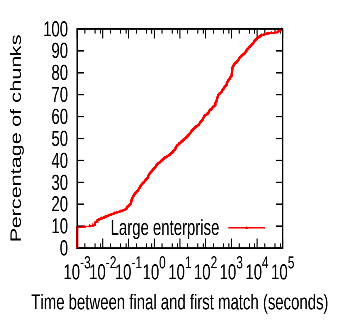
Thus, *while full packet matches provide 20-50% in overall savings, in order to get the maximum benefit of redundancy elimination, we need to index the vast majority of small packet matches of size less than 150 bytes.*

6.3 Redundancy: Temporal view

In this section, we would like to understand the temporal locality of matches, i.e., when a redundant chunk is matched between the current packet and a packet in the cache, how far in the past is the matched packet? We consider two temporal metrics: 1) time between current packet and matched packet and 2) time between the final most recent match and the first match for a given chunk. Lets consider the first metric now. In order to have a normalized metric that works both during peak and lean traffic periods, we use the percent of cache size between the current packet and the matched packet as the normalized temporal metric.



(a) Savings vs Match time

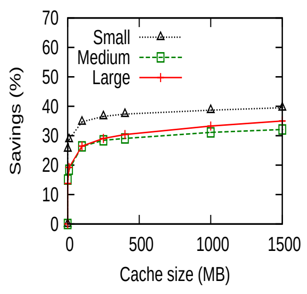


(b) Time between matches

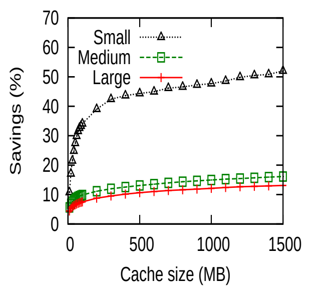
Figure 9: Redundancy: Temporal Characteristics

In Figure 9(a), we plot the redundancy match contribution to overall savings as a percentage vs the recency of match (computed as a percentage of cache size). We use a default cache size of 250MB. The curves rise steeply for all the traces with 60-80% of the savings due to matches with packets in the most recent 10% of the cache. This characteristic implies that a) adding packets to the cache in a FIFO manner and evicting the oldest packet is a good strategy and b) small cache sizes can provide bulk of the savings of a large cache.

Let us now consider the time difference between the final time a specific chunk was matched and the first time the same chunk was matched. This metric captures the duration for which a chunk is useful. Note that the time difference can be much larger than the holding time of the cache since popular chunks can recur throughout the trace and the time difference can be as large as the trace length (24 hours). In Figure 9(b), we plot the CDF of the time difference in log scale between the final and first matches of each unique chunk. Note that for 60% of the chunks, the time difference is less than 100 seconds and for approximately 80% of the chunks, the time difference is less than 1000 seconds. This again highlights the *high degree of temporal locality of matches that a small cache would be able to accommodate*.



(a) Enterprise



(b) University

Figure 10: Redundancy vs Cache size

Figure 10 plots the savings versus cache size for the enterprise and university traces, respectively. We can see that small cache sizes do indeed provide significant percentage of the savings, with the “knee” of the savings curve between 100-250MB.

6.4 Redundancy: Hit Distribution

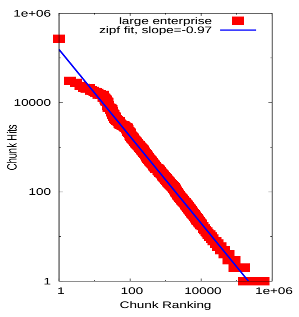
We now examine if the redundancy in network traffic is primarily due to a few pieces of content repeated multiple times or multiple pieces of content repeated a few times each. If the former is

Counts	Length	Protocol	description
268K	128	various	string of zeros
30K	42	SMB	content fragment
28K	68	HTTP	content fragment
24K	50	SMB	content fragment
21K	8	Kerberos	full packet

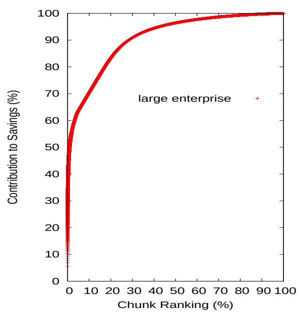
Table 6: Characteristics of popular chunks

true, then a small packet store would suffice to identify a significant fraction of the redundancy. If the latter is true we may have to store many more chunks of data in a much larger packet store.

More generally, we would like to understand the distribution of frequency of unique chunk matches. Given that researchers have shown that web page access frequency exhibits a zipf-like distribution [14], it would be interesting to see if the same phenomena also holds for chunk matches. Zipf-like distributions will have relative probability of a hit for the i^{th} most popular chunk proportional to i^{α} for some α close to -1.



(a) Chunk matches



(b) Match contribution

Figure 11: Chunk match distribution and their contribution to total savings

In order to answer this question, we keep track of each matched chunk and count how many times the identical chunk is matched in the entire trace. In Figure 11(a), we plot the frequency of chunk hits versus the chunk rank, sorted by the number of hits, in a log-log scale for a large enterprise trace. The linear distribution in the log-log plot confirms the zipf-like nature of the chunk hits. We also fit a zipf-distribution for the bulk of the data points, ignoring the top 100 chunks and chunks with exactly 1 hit, and find that the best fit results in $\alpha = -0.97$ (a fit over all the data points results in $\alpha = -0.9$). Similar zipf-like characteristics are also seen in other traces (not shown).

Figure 11(b) shows the contribution of the chunk hits to the bandwidth savings versus the percentage of chunks sorted by their contribution. It is clear from the figure that about 80% of savings come from 20% of the chunks. On the other hand, in order to obtain the remainder 20% of savings, we need to retain 80% of the chunks. This implies that a small cache should be able to capture bulk of the savings but capturing the full savings would require a large amount of cache. The *zipf-like chunk hit distribution thus explains the diminishing returns of the large cache* size seen in Figure 10.

Table 6 lists some of the characteristics of popular chunks in the enterprise trace. Most of these chunks are less than 150 bytes long and are content fragments of a packet except for one chunk of 8 bytes that represents a full packet. The small chunk sizes seen motivate the need for using a small minimum match size parameter, w.

7. FINDINGS AND IMPLICATIONS

Protocol-independent redundancy elimination techniques have become increasingly popular in recent years and are poised to play an important role in the current and future Internet architecture. Our goal in this paper was to conduct an in-depth measurement-based study of the fundamental issues pertaining to the benefits, trade-offs and design issues underlying these techniques.

In this section, we summarize our empirical findings and identify important implications on the design and usage of redundancy elimination and the role it can play in network infrastructure and protocols.

- **Protocol composition:** Protocol composition over the access link has changed significantly since the work by [26]. Specifically, enterprise traffic (25-70%) has significant amount of file access traffic (SMB, NetBios, etc.) due to the likely shift in enterprise management approach towards use of centralized servers in data centers. The changing composition of protocols over time makes a strong argument for the sustained relevance of protocol-independent redundancy elimination techniques.
- **Algorithm:** While network traffic redundancy suppression proposals [12, 26] rely on a fingerprint selection algorithm, termed MODP in this paper, we find that an algorithm that ensures more uniform selection of fingerprints using a local property such as local-maxima, termed MAXP, outperforms MODP by 5-10% on most traces and as much as 35% in some traces.
- **Packet-level compression:** We find that packet-level compression can be effectively applied after redundancy suppression, delivering incremental gains of up to 26%.
- **Object-level compression:** Protocol-independent redundancy elimination outperformed both HTTP object-level compression as well as caching, with significant out-performance in the case of the University /24 trace.
- **Temporal variability:** We find that redundancy elimination does not reduce traffic variability commensurately. While 95th-percentile savings deliver close to average savings over some time intervals, peak savings are generally lower than average savings in many of the traces. The burstiness of traffic, as measured by the energy metric derived using wavelet-based multiresolution analysis, is typically not commensurately lower due to redundancy elimination.
- **Origins:** In the enterprise traces, we found that 75-90% of savings were due to intra-user matches between packets that had the same source and destination IP addresses. This argues for a pure end-to-end redundancy elimination solution, diminishing the need for the deployment of middleboxes in most enterprises. On the other hand, based on our university traces, we found that a middlebox solution is beneficial in busy web server settings where significant portion of redundancy is due to inter-user traffic matches.
- **Spatial view:** We find that most matches (70%) are small in size (less than 150 bytes) while only about 10% of matches are full-packet matches. In terms of contribution to savings, full packet matches contribute to (25-50%) of total savings while packets of size less than 150 (50) bytes contribute to about 20% (4%) of the savings. Thus, simple techniques like indexing only full packets can provide up to half of the total savings, while capturing the full savings involves a significant amount of indexing of small packet fragments.
- **Temporal view:** We find that most matches are from recent packets in the cache. Thus, a FIFO-based approach for storing packets in the cache would work well.

- **Match Distribution:** We find that the chunk match hit follows a zipf-like distribution, with a few chunks that extract a large number of hits and vast majority of chunks with one or two hits. This implies that smaller caches can provide bulk of the gains of redundancy elimination and increasing cache size would provide diminishing returns in terms of bandwidth savings.

8. CONCLUSION

Following the work of Spring et al. in 2000, a slew of commercial WAN optimization middleboxes have emerged which attempt to improve network link performance by suppressing repeated strings of bytes in network packets. Today, there are many deployments of these protocol-independent redundancy elimination techniques at enterprise and data center access links and across congested ISP links. Based on the perceived benefits of these techniques, recent efforts have argued for integrating redundancy elimination into network infrastructure and protocols [12, 19, 16].

Despite the increasingly important role of redundancy elimination in the network infrastructure, very little is known about the range of benefits and trade-offs these approaches offer today, and the fundamental issues underlying their design. Using packet traces collected at twelve distinct network vantage points, we showed that packet-level redundancy elimination techniques can deliver average bandwidth savings of 15-60% for enterprise and university access links as well as the links connecting busy web servers. However, in the case of enterprise traffic, we found that the overall burstiness of traffic was not significantly reduced and the savings during peak traffic periods was variable.

We found several interesting characteristics of redundancy in network traffic, summarized in the previous section. One surprising implication of our findings was that a client-server redundancy elimination solution could provide approximately similar savings as a middlebox in small/medium, and to an extent, large enterprises, obviating the need for deploying an expensive middlebox-based redundancy elimination solution. Designing such an end-to-end redundancy elimination system that is scalable and efficient is a topic for future work.

Acknowledgments. We thank the anonymous reviewers whose comments helped improve our paper. This work was supported in part by an NSF CAREER Award (CNS-0746531) and an NSF NeTS FIND Award (CNS-0626889).

9. REFERENCES

[1] Citrix, application delivery infrastructure. `http://www.citrix.com/`.

[2] Computerworld - WAN optimization continues growth. `www.computerworld.com.au/index.php/id;1174462047;fp;16;fpid;0/`.

[3] F5 Networks: WAN Delivery Products. `http://www.f5.com/`.

[4] Netequalizer Bandwidth Shaper. `http://www.netequalizer.com/`.

[5] Packeteer WAN optimization solutions. `http://www.packeteer.com/`.

[6] PeerApp: P2P and Media Caching. `http://www.peerapp.com`.

[7] Peribit Networks (Acquired by Juniper in 2005): WAN Optimization Solution. `http://www.juniper.net/`.

[8] Riverbed Networks: WAN Optimization. `http://www.riverbed.com/solutions/optimize/`.

[9] WAN optimization revenues grow 16% - IT Facts. `www.itfacts.biz/wan-optimization-market-to-grow-16/1205/`.

[10] WAN Optimization: Wikipedia entry. `http://en.wikipedia.org/wiki/WAN_Optimization`.

[11] P. Abry and D. Veitch. Wavelet analysis of long-range dependent traffic. *IEEE Transactions on Information Theory*, 44(1):2–15, Jan 1998.

[12] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination. In *ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[13] N. Bjorner, A. Blass, and Y. Gurevich. Content-Dependent Chunking for Differential Compression, the Local Maximum Approach. Technical Report 109, Microsoft Research, July 2007.

[14] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *IEEE Infocom*, 1999.

[15] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital SRC Research Report, 1994.

[16] F. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. Andersen. Ditto - A System for Opportunistic Caching in Multi-hop Wireless Mesh Networks. In *Proc. ACM Mobicom*, San Francisco, CA, Sept. 2008.

[17] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. In *SIGCOMM '98*, 1998.

[18] D. Goldenberg, L. Qiu, H. Xie, Y. Yang, and Y. Zhang. Optimizing cost and performance for multihoming. In *ACM SIGCOMM*, 2004.

[19] X. Li, D. Salyers, and A. Striegel. Improving packet caching scalability through the concept of an explicit end of data marker. In *HotWeb*, 2006.

[20] U. Manber. Finding similar files in a large file system. In *USENIX Winter Technical Conference*, 1994.

[21] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. *SIGOPS Oper. Syst. Rev.*, 35(5), 2001.

[22] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Cambridge, MA, Apr. 2007.

[23] M. Rabin. Fingerprinting by random polynomials. Technical report, Harvard University, 1981. Technical Report, TR-15-81.

[24] RouteScience Technologies, Inc. Routescience PathControl. `http://www.routescience.com/products`.

[25] S. Schleimer, D. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *SIGMOD*, 2003.

[26] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *SIGCOMM*, pages 87–95, 2000.

[27] Squid Web Proxy Cache. `http://www.squid-cache.org/`.

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

[29] A. Wolman et al. Organization-based Analysis of Web-Object Sharing and Caching. In *Proceedings of the 2nd USITS*, Oct 1999.

[30] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, 23(3):337–343, 1977.