Enhancing Redundant Network Traffic Elimination

Emir Halepovic Carey Williamson Majid Ghaderi

Department of Computer Science
University of Calgary
2500 University Dr. NW, Calgary, Alberta, Canada T2N 1N4

Abstract

Protocol-independent redundant traffic elimination (RTE) is a method to detect and remove redundant chunks of data from network-layer packets by using caching at both ends of a network link or path. In this paper, we propose a set of techniques to improve the effectiveness of packet-level RTE. In particular, we consider two bypass techniques, with one based on packet size, and the other based on content type. The bypass techniques apply at the front-end of the RTE pipeline. Within the RTE pipeline, we propose chunk overlap and oversampling as techniques to improve redundancy detection, while obviating the need for chunk expansion at the network endpoints. Finally, we propose savings-based cache management at the back-end of the RTE pipeline, as an improvement over FIFO-based cache management. We evaluate our techniques on full-payload packet-level traces from university and enterprise environments. Our results show that the proposed techniques improve detected redundancy by up to 50% for university traffic, and up to 54% for enterprise Web server traffic.

Keywords: Redundant traffic elimination, content-aware chunking, packet-level caching

1. Introduction

As Internet traffic volumes continue to grow, redundant traffic elimination (RTE) has attracted a lot of research attention in recent years [1, 2, 3, 4, 8, 11, 14]. The redundancy in Internet traffic arises naturally from the large number of users, as well as the highly-skewed popularity distribution for Internet content [5, 6]. As a result, there are many repeated transfers of the same (or similar) content, both in client-server and in peer-to-peer networking applications.

From a philosophical viewpoint, repeated transfers of similar data represent a waste of network resources. In more practical terms, redundant traffic can be a particularly acute problem for limited-bandwidth Internet access links (e.g.,

Email address: `{emirh, carey, mghaderi}@cpsc.ucalgary.ca` (Emir Halepovic Carey Williamson Majid Ghaderi)

wireless or cellular access networks), or even for high-bandwidth links operating at or near their capacity. Redundant traffic can also be an issue economically, if Internet providers (or users) are charged based on the traffic volumes sent and received between peering points (i.e., usage-based billing).

Many techniques have been proposed for RTE, including *protocol-independent* RTE [14], which operates at the network layer. Protocol-independent RTE divides packet payload into *chunks*, and uses unique hash values to detect redundant content [14].

The RTE process can be viewed as a pipeline, as shown in Figure 1. A chunk selection algorithm has a sampling parameter that determines how many chunks are chosen, and a heuristic to determine which ones are chosen. The selected chunks are usually non-overlapping. There is a finite cache of recently-observed packets at each endpoint, usually with a First-In-First-Out (FIFO) *cache replacement policy.* When matching chunks are detected within a packet, an optional *chunk expansion* algorithm can be run at the endpoints to expand the matched region, increasing byte savings [2, 14].

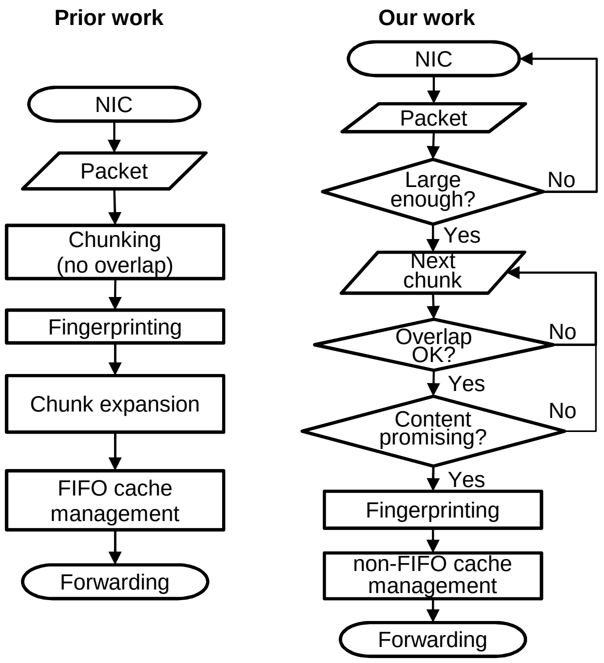


Figure 1: Processing pipeline for RTE

There are many important factors that influence the practical effectiveness of RTE (i.e., the byte savings achieved on a network link). These factors include the data chunk size, the chunk selection algorithm, the sampling period, and the cache replacement policy. In the literature, the bandwidth savings reported for RTE are typically 10-12% for university campus Internet traffic, and as high as 30-60% for outbound enterprise traffic [3].

In this paper, we augment the RTE pipeline with additional functionality, as shown on the right hand side of Figure 1. Our new proposed techniques include size-based bypass, chunk overlap, savings-based cache management, and content-aware chunk selection. We evaluate these techniques on full-payload packet traces from university and enterprise environments, and demonstrate their effectiveness. We also carefully investigate the impacts of chunk size, sampling period, and cache management on the RTE benefits.

The unifying theme in our work is exploiting *non-uniformities* in Internet traffic. For example, the bimodal distribution of Internet packet sizes [5] motivates our size-based bypass technique, wherein large data-carrying IP packets are subjected to full RTE processing, while small IP packets are not. As another example, the Zipf-like skew observed in redundant chunk popularity [3] motivates a savings-based approach to cache management that retains the most valuable chunks, in terms of cumulative byte savings. Finally, non-uniform chunk popularity is a basis for our novel chunk selection scheme, called *content-aware RTE*, which preferentially chooses promising chunks, rather than random or non-redundant chunks. We exploit these non-uniformities to improve RTE performance.

Our key results include the following:

- Using 64-byte chunks rather than 32-byte chunks improves RTE by up to 21%, while reducing execution time.
- Size-based bypass reduces execution time by 2-25%, while *improving* RTE up to 4% in some cases.
- Chunk overlap can improve RTE by 9-14%.
- A savings-based cache replacement policy can improve the effectiveness of RTE by up to 12%.
- The cumulative effects of the foregoing techniques improve existing RTE savings by up to 54%.
- Content-aware RTE can improve redundancy detection by up to 37%.

The rest of this paper is organized as follows. Section 2 reviews prior related work. The data sets and methodology used in this study are described in Section 3. Techniques for improvement of the RTE pipeline are presented in Section 4, while the notion of content-aware RTE is discussed in Section 5. Finally, Section 6 concludes the paper.

2. Background and Related Work

In general, RTE techniques deploy extra hardware or software resources in the network to detect and eliminate repeated data transfers. For example, Web proxy caches have been successfully deployed and used for well over a decade [5].

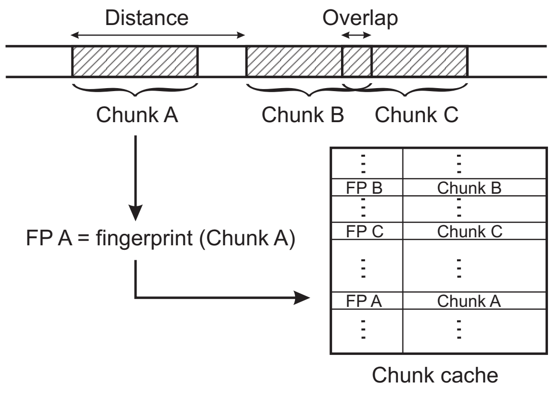


Figure 2: RTE terminology

However, object-level caching alone cannot eliminate all redundancy [14]. Furthermore, traditional object-level caching does not work well for personalized Web pages that have slight changes to the base content. Delta encoding [8] can help in this case, but it requires that both client and server share the same base version of the document, which is not applicable for general Internet traffic.

2.1. Content-Defined Chunking

Content-Defined Chunking (CDC) is a general-purpose RTE technique that works within individual objects (packets, files, or Web pages) as well as across objects. Objects are divided into *chunks* to facilitate comparisons within an object or across objects (see Figure 2). Data chunks could be fixed-size or variable-size. Chunks generally do not correspond to a specific location inside the object; rather, they are defined by their content. Based on the chunk selection algorithm, chunks might be separated by some *distance*, or they might *overlap*.

For each chunk, a probabilistically unique hash value is computed using a fingerprinting technique such as SHA-1 or *Rabin fingerprint* [10, 12]. Chunks are typically 32-64 bytes in size, while fingerprints are 8 bytes (64 bits). Rabin fingerprints are especially useful because they can be computed efficiently using a sliding window over a byte stream.

In storage systems, the CDC approach is widely used for *data de-duplication* [10]. That is, chunking is used to reduce the overall disk space required, by writing duplicated chunks only once. A similar technique is used in the `rsync` tool to find portions of a file that need to be updated. Chunking can also be combined with other techniques, such as delta-encoding [8, 15].

In networking, protocol-independent RTE using CDC in conjunction with Rabin fingerprints was originally proposed and evaluated on Web server traffic [14]. Using six Web server traces, the authors show that Web traffic is up to 54% redundant. This approach operates at the packet level, using middle-boxes inserted at the two endpoints of a bandwidth-constrained network link [3, 14].

An RTE process runs on each of these middle-boxes and employs a *cache* of recently transferred packets, together with fingerprints of their selected chunks. If a current packet's payload (or a portion thereof) has been previously seen and cached, then it can be encoded using meta-data and transferred using fewer bytes than the actual packet. This meta-data consists of fingerprints and other information sufficient to reconstruct the whole chunk using the receiver-side cache. Therefore, there is an *overhead penalty* associated with encoding chunks with meta-data inside the packet.

This technique detects redundancy within and across objects, as well as within and across the traffic of individual users. Furthermore, the approach is protocol-independent, since it can find redundancy in all Internet traffic, regardless of the application protocol. This and similar approaches based on CDC are often called WAN optimization in commercial products [7]. Other architectures for RTE include universal deployment across Internet routers [2], coordinated RTE across routers within a network [4], and end-system RTE within enterprises [1].

2.2. Chunk Selection Algorithms

The core of any RTE process is a chunk selection algorithm. Because the sliding window used for chunk analysis advances by one byte at a time, there are many candidate chunks to consider on any given packet (e.g., 37 candidates for 64-byte chunks on 100 bytes of data). Since recording all of these chunks is impractical, a sampling heuristic is used to record only a fraction $1/p$ of all possible chunks for caching. The parameter p is called the *sampling period.* A typical value is $p = 32$, for which about 3% of all possible chunks are selected, on average.

There are many possible ways to select the chunks for caching. In practice, the choice is often based on some property of the chunk (e.g., location, value, byte content) or its fingerprint (e.g., numerical value). Even with the same sampling period, selection algorithms may differ in how many chunks are selected, as well as in how the chunks are spatially distributed within the data. In the following, we briefly review several chunk selection algorithms from the literature, and illustrate their properties in Figure 3. The left-hand side of Figure 3 shows examples of chunk locations within the data packet, while the right-hand side depicts the distribution of inter-chunk distance (the horizontal axis is the inter-chunk distance, while the vertical axis is the relative frequency of occurrence for a given distance).

FIXED: The FIXED approach selects every p-th chunk, as shown in Figure 3(a). This method is computationally efficient, achieves exactly the target sampling rate, and ensures that the chosen chunks are non-overlapping. However, the deterministic sampling approach is not robust against small changes in the data (e.g., inserting a word in a text document). Therefore, this approach should not be used [11, 14], since it is not as effective as other approaches [1].

MODP: MODP is the original chunk selection algorithm proposed for network-layer RTE [14]. It selects chunks for which the fingerprint value is equal to 0 mod p. The MODP method is robust to small changes in objects (files or packets),

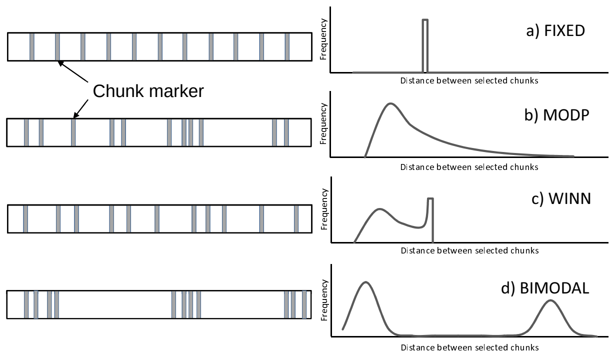


Figure 3: Example chunk selection algorithms

produces a controllable number of fingerprints, and is simple to compute when p is a power of 2. Its main drawback is the unpredictability of its chunk selection. As shown in Figure 3(b), the distance distribution can have a long tail, indicating that large blocks of data may remain unselected for fingerprinting.

WINN: Winnowing is a way to "filter" a data stream. Unlike MODP, the WINN algorithm ensures that at least one chunk is chosen within a specified data interval [13]. It is implemented by tracking a sliding window of recent fingerprints, and explicitly choosing the numerically largest (or smallest) value from the sliding window. The additional data structure imposes processing overhead on WINN causing it to be slower than MODP. Figure 3(c) shows that chunk selection in WINN is approximately uniform, and the distance between chunks is always bounded. The predominating distance is equal to the sampling period. This approach improves RTE, especially on HTML pages [13].

MAXP: One possible concern with MODP and WINN is the processing overhead required to compute Rabin fingerprints for every possible data chunk. Although Rabin fingerprints can be computed efficiently, almost a thousand such computations are needed on a 1 KB packet. A more efficient algorithm would compute fingerprints only when necessary (i.e., when a data chunk is selected for caching). One such algorithm is MAXP [3]. MAXP is based on WINN, but instead of choosing a local maximum among fingerprints, it selects based on the data bytes of the chunk. This reduces the overhead of fingerprint computation. The distribution of chunks is similar to that achieved with WINN.

SAMPLEBYTE: A recently proposed algorithm uses specific byte values in the content as triggers for chunk selection. In particular, selection is based on the most redundant bytes, determined by training the algorithm on sample traffic, and recording values in a lookup table [1]. The sampling pattern achieved (not shown) is unpredictable, since it depends on the locations of the trigger bytes. The main benefit of SAMPLEBYTE is even faster execution than MAXP, while preserving robustness to small perturbations in content.

2.3. Content-Aware Chunk Selection

In our work, we advocate selecting the most promising data chunks, in terms of their potential contribution to RTE.

Our method uses *non-uniform* sampling: little or no sampling for low-redundancy content, and *oversampling* for highly-redundant content. The distribution of selected chunks is shown in Figure 3(d). The distribution is bimodal, reflecting highly concentrated sets of selected chunks, as well as lengthy blocks of bypassed data. In this case, it is acceptable to have expansive regions of skipped data, especially when the data has little redundancy. The details of this approach are discussed later in the paper.

3. Evaluation Methodology

In this section, we describe our data sets and methodology, as well as the basic configuration parameter settings for RTE evaluation.

Table 1: Characteristics of the data traces (GB)

Trace	Date	Time	In	Out	Total
1	Apr 6, 2006	9 am	19.9	15.5	35.4
2	Apr 6, 2006	9 pm	8.2	14.9	23.1
3	Apr 7, 2006	9 am	23.6	16.9	40.5
4	Apr 7, 2006	9 pm	18.3	12.5	30.8
5	Apr 8, 2006	9 am	8.8	8.2	17.0
6	Apr 8, 2006	9 pm	18.9	12.3	31.2
7	Apr 9, 2006	9 am	7.7	10.6	18.3
8	Apr 9, 2006	9 pm	21.9	15.4	37.3
A	Apr 11, 2006	12 am	-	49.5	49.5
B	Apr 12, 2006	12 am	-	46.6	46.6
W 1-20	Apr/May, 2004	12 am	-	6.2	6.2

3.1. Data Sets

We evaluate our proposed RTE approach using three sets of full-payload packet traces (see Table 1 for details about the traces). The first set comes from the Internet access link at the University of Calgary campus. A total of 8 one-hour bi-directional traces were collected, with the total IP payload of 233.6 GB. These traces are labeled 1-8. The second set consists of 2 traces collected from the student residences at the University of Calgary. These are outbound traces with a total IP payload of 96.1 GB, and they are labeled A and B. The third set consists of 20 day-long traces collected at an enterprise Web server, during April and May 2004. These are bi-directional traces, but we use only the outbound portion, which carried most of the data. These traces, which are almost exclusively HTTP, are labeled W 1-20.

The primary set of traces used for evaluation of each RTE improvement technique is the set of campus traces. The residence and Web server traces are used to demonstrate that our results generalize across a variety of environments.

While the campus traces are several years old (from 2006), there are three reasons why we use them in our experiments. First, the traces are from a university environment and have comparable composition to those studied in prior RTE work [3]. Second, our traces provide snapshots of Internet traffic on mornings and in evenings, as well as on weekdays and weekends, so that we can assess the robustness of RTE across different traffic mixes. Third, these traces were used in prior published work on Internet traffic classification [9], and thus we have detailed knowledge about the traffic composition in these traces. Table 2 summarizes the application profile for all traces.

Table 2: Application profile for all traces

Application	1-8 Out	1-8 In	A	B	W 1-20
HTTP	30%	45%	1%	1%	100%
Email	6%	7%	0%	0%	0%
Unknown	27%	22%	24%	28%	0%
P2P	23%	11%	60%	61%	0%
SSL	7%	9%	9%	6%	0%
Other	7%	6%	6%	4%	0%

The campus traces are divided into the incoming and outgoing traffic, with the rationale that they should be studied separately. The main reason is that different redundancy may exist in each direction. The volume of data in different directions may also merit different cache sizes, RTE algorithms, or parameter settings.

3.2. Implementation Details

We use a custom-written simulator to process the traces and report on detected redundancy. Simulations are performed on a Linux-based server with 3 GHz Quad-core CPU and 32 GB of RAM. We consider the following six factors, and investigate their effect on RTE and execution time: (1) Chunk selection algorithm; (2) Data chunk size; (3) IP packet size; (4) Chunk overlap; (5) Cache replacement policy; and (6) Content-aware RTE.

The primary focus in our work is on redundancy detection. For this reason, we choose WINN as a representative of uniform sampling algorithms, since MAXP and SAMPLEBYTE focus primarily on improving execution time. However, we do not completely disregard execution time in our study. Rather, we use it to quantify the tradeoffs for the factors that we investigate.

We implement MODP using an approach similar to Spring and Wetherall [14]. We compute Rabin fingerprints, and select them using a sampling period of 32, the same as the base case used in [3, 14]. We adjust the sampling period as necessary when exploring improvements in our study. Our WINN implementation follows the algorithm from [13], including a small bug fix for the "index off by one" special case. Most of our experiments use default settings from previous work as a base case scenario, such as 32-byte chunk size, 500 MB cache, and FIFO cache replacement policy [3]. We explicitly indicate when we deviate from these values.

In previous work, chunk expansion is an additional step performed at the endpoints to increase redundancy detection. After a matching chunk is detected, the matching region is expanded byte-by-byte around the chunk to achieve the largest possible match [3, 14]. Expansion improves average RTE by 13.6% over direct chunk match [1], but introduces additional processing and storage overheads.

We use a much simpler approach, with fixed-size data chunks, and introduce *chunk overlap* as a technique to obviate the need for chunk expansion at the endpoints. The benefits of our approach are reduced processing costs, lower encoding overhead for meta-data, and reduced storage that requires only fingerprints and chunks to be stored, rather than whole packets. In general, the meta-data needs to include the chunk identifier (fingerprint), the location inside the packet where the matching chunk starts, and the information about the expanded matching region before and after the matching chunk. With chunk expansion, this overhead can be as high as 12 bytes per chunk, to convey all the necessary information [14]. However, an optimized implementation can reduce this overhead by using cache indexing based on the offset of the matching region inside the cache, rather than the value of the fingerprint [1].

When using our fixed-size chunks, we only need to encode the chunk location in the packet payload, and the offset in the cache. For example, this overhead is 34 bits per chunk, for a 512 MB cache of 64-byte chunks. For larger caches, or any additional meta-data, increasing the penalty to 5 bytes per chunk still represents only 7.8% overhead for the 64-byte chunk. Smaller chunk sizes would have correspondingly higher overhead, since more chunks could be stored in the cache.

Throughout this paper, we use detected redundancy as the metric for evaluating RTE, keeping in mind that the practical implementation will include encoding overhead in the actual byte savings.

3.3. Baseline Experiments

In this section, we establish baseline parameter settings for chunk selection algorithm and chunk size.

3.3.1. WINN versus MODP

We first conduct an apples-to-apples comparison between the MODP and WINN chunk selection algorithms. To the best of our knowledge, these two chunk selection algorithms have only been compared on HTML pages [13], and not on network traces. Anand *et al.* [3] compared MODP and MAXP, showing that MAXP detects 5-10% greater redundancy in most cases, and up to 35% more in specific cases. That study used 32-byte chunks and a sampling period of $p = 32$.

Our comparison explicitly considers the number of chunks selected, as well as the execution time of each algorithm. One algorithm can be considered better than another if it is faster, while offering the same level of RTE, or if it provides better RTE using the same number of chunks.

Table 3: Comparison of MODP and WINN for different sampling periods

	Value	30 MB cache		131 MB cache	
Algorithm	of p	Savings	Time	Savings	Time
MODP	32	8.4%	93	10.0%	104
MODP	16	9.4%	124	11.7%	144
MODP	8	9.4%	171	12.4%	212
MODP	4	9.6%	258	12.8%	335
WINN	35	9.3%	170	11.1%	172
WINN	20	10.0%	191	12.4%	209
WINN	10	9.8%	227	12.7%	581
WINN	5	9.6%	288	12.8%	781

We want to know whether WINN is always better than MODP. We configure MODP with a sampling period of $p = 32$, as used in previous studies [3, 14]. For WINN, the sampling period is defined by a sliding window of fingerprints, from which the local maximum (or minimum) fingerprint value is chosen.

Our preliminary tests showed that the sampling periods for MODP and WINN do not correspond directly. That is, $p = 32$ does not produce the same number of selected chunks for these two algorithms, even though we want them to be comparable. It would be unfair to compare two algorithms based on the same sampling period if that period produces vastly different numbers of chunks. Given that WINN selects chunks more uniformly than MODP, WINN tends to select more chunks than MODP on the same data.

Since the sampling period adjustment for MODP is extremely coarse (power of 2), we make adjustments to the sampling in WINN until we arrive at a similar number of selected chunks for each MODP setting. Since an exact match in the number of fingerprints is difficult to obtain, we are satisfied when they are within 2%.

Based on our experiments with one campus trace, we determined the corresponding values for sampling periods, as shown in Table 3, where MODP and WINN are compared across different sampling periods.

MODP is always more efficient than WINN in terms of execution time (expressed in seconds throughout the paper). However, the RTE benefits of WINN are evident with longer sampling periods, because of its more uniform sampling property. The benefits disappear at higher sampling rates, such as $p = 4$. We have verified that the relative performance of MODP and WINN remains the same when using different cache sizes and other incremental improvements.

Table 4 compares the detected redundancy and execution time of these two algorithms for all 8 campus traces with our baseline parameters. For both inbound and outbound traffic, WINN detects more redundancy. Comparing the average RTE per trace, WINN is better by 19% for inbound traffic, and by 9.7% for outbound traffic. In some cases (Traces 2 and 6), WINN detects significantly more redundancy than MODP. However, MODP is again more efficient in terms of execution time.

In the rest of the paper, we use WINN as our baseline, because of its higher

Table 4: Comparison of MODP and WINN algorithms

Trace	Inbound Redundancy		Inbound Time	
	MODP	WINN	MODP	WINN
1	16.3%	16.6%	3478	5969
2	9.9%	16.9%	1466	2433
3	8.8%	9.1%	4722	7808
4	9.8%	10.8%	3511	6004
5	7.3%	7.7%	1840	2862
6	6.3%	10.2%	3884	6281
7	8.2%	8.7%	1548	2428
8	5.8%	6.1%	4557	7099
Average	9.0%	10.8%	3126	5111

Trace	Outbound Redundancy		Outbound Time	
	MODP	WINN	MODP	WINN
1	18.3%	18.8%	2940	4787
2	8.4%	11.8%	2944	4752
3	9.4%	9.8%	3401	5498
4	7.9%	8.4%	2407	4022
5	15.2%	15.9%	1479	2486
6	9.5%	11.8%	2339	3966
7	7.5%	7.9%	2099	3572
8	11.0%	11.4%	2937	5107
Average	10.9%	12.0%	2572	4274

RTE. The slower execution time of WINN is a secondary concern, since it is known that using MAXP in place of WINN retains the advantages over MODP in both savings and execution time. While our results for MODP and WINN are consistent with those from previous studies, *we do not make direct comparisons to prior work due to the different traces and implementations used.*

3.3.2. Chunk Size

An important factor in RTE is the chunk size, for which there is an inherent tradeoff. With small chunks, more matches occur, but the meta-data overhead to encode and transmit chunks is excessive. With large chunks, the meta-data overhead is manageable, and the byte savings from a chunk match are greater, but fewer chunk matches occur.

A good compromise is a chunk size of 32-64 bytes, which provides sufficient RTE to compensate for the costs associated with storage, processing, and encoding. The analysis of RTE using different chunk sizes for MODP was presented in [14], leading to the adoption of 64-byte size as a good compromise. For MAXP and SAMPLEBYTE, 32-byte chunks were found to be the best, when coupled with chunk expansion [1, 3]. Empirical evidence shows that most of the benefits of RTE arise from matches of less than 150 bytes, in both enterprise and campus traces [3]. Furthermore, 3 of the 5 most popular chunk matches were between 42 and 68 bytes in length [3]. These results justify why 32 and 64 bytes are considered good values for RTE.

Table 5: Comparison of 32-byte and 64-byte chunks

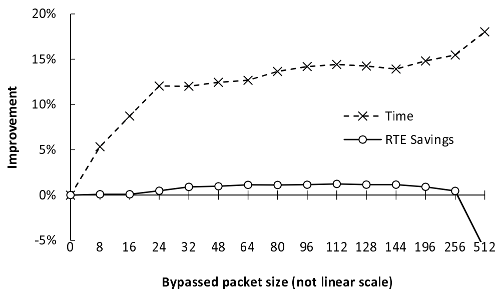
Packet Trace	Inbound Redundancy		Inbound Time	
	32-byte	64-byte	32-byte	64-byte
1	16.6%	19.1%	5969	5453
2	16.9%	19.3%	2433	2327
3	9.1%	10.8%	7808	6850
4	10.8%	13.3%	6004	5129
5	7.7%	9.1%	2862	2493
6	10.2%	11.9%	6281	5357
7	8.7%	10.6%	2428	2158
8	6.1%	7.3%	7099	6307
Average	10.8%	12.7%	5111	4509
	Outbound Redundancy		Outbound Time	
Trace	32-byte	64-byte	32-byte	64-byte
1	18.8%	22.2%	4787	4257
2	11.8%	13.9%	4752	4318
3	9.8%	12.3%	5498	4829
4	8.4%	10.3%	4022	3562
5	15.9%	19.0%	2486	2297
6	11.8%	14.6%	3966	3511
7	7.9%	9.5%	3572	3174
8	11.4%	14.4%	5107	4405
Average	12.0%	14.5%	4274	3794

For our implementation, which uses fixed-size chunks without expansion, chunk size selection is even more important. If expansion is used, every byte of successful expansion amortizes the overhead from encoding the meta-data. Without expansion, however, this overhead is a constant. Therefore, a careful selection of chunk size is required. For this reason, we compare RTE using 32-byte and 64-byte chunks.

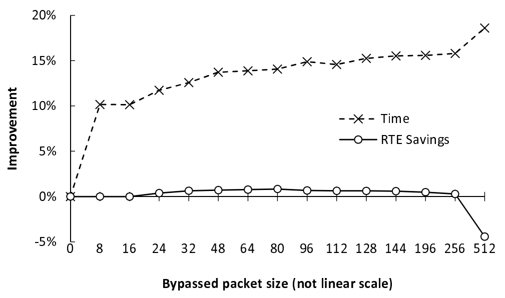
Fortunately, the experimental results are quite definitive. In particular, we find that using 64-byte chunks improves both redundancy detection and execution time (see Table 5). Improvements in RTE are 17.8% and 21.3% on average, for inbound and outbound traffic, respectively. The corresponding results for execution time show reductions by 11.8% and 11.2%, respectively. In other words, having fewer large matches is better for RTE than having more smaller matches. The reduced execution time reflects fewer chunks being processed and managed by the cache. Since smaller chunk sizes imply higher encoding overhead, it is clear that 64-byte chunks are the preferred choice in our work, where fixed-size chunks are used.

4. Proposed RTE Improvements

In this section, we present and evaluate our new network-layer techniques for RTE pipeline processing, as illustrated in Figure 1. We begin with the



(a) Inbound traffic



(b) Outbound traffic

Figure 4: Effects of bypassing small packets

bypass technique at the front of the RTE pipeline, and then discuss results for chunk overlap and non-FIFO cache management. We defer content-aware RTE to Section 5.

4.1. Size-based Bypass

In the RTE packet caching approach described in [14], *every* packet is cached at least once, even if it is smaller than the chunk size. However, there are storage and processing costs associated with each packet.

Our objective is to select only those chunks that could eventually produce savings, which requires at least one full chunk per packet. Clearly, packets that carry just a few bytes, or no data at all, are of no use for RTE. Previous works do not explicitly consider packet size as a factor in RTE performance [2, 3, 14], although some do so implicitly by applying a minimum chunk size.

One of the well-known non-uniformities in Internet traffic is the bimodal packet size distribution induced by TCP [16]. About half of IP packets are "full" data-carrying packets, and about half are minimal size, carrying TCP acknowledgments (ACKs). Specifically, we want to skip the processing for small ACK packets traversing the network. Furthermore, many data packets carry just a few data bytes, rendering them "useless" for potential RTE savings. Signalling, control information, and secure shell are examples of such packets. The RTE process running at the network layer should therefore ignore all packets that carry insufficient data bytes, regardless of the transport or application protocol.

For RTE purposes, the best packet size threshold depends on the chunk size used for fingerprinting. A reasonable heuristic is that the data should provide enough bytes for a complete 64-byte chunk. Network-layer RTE should also ignore transport-layer headers, which have limited redundancy. In our traces, many TCP packets carry an extended header of up to 32 bytes, including TCP options (e.g., SACK, timestamps).

We investigate in detail the benefits of size-based bypass for different packet size thresholds ranging from 8 to 512 bytes. Figure 4 shows the percentage improvement in execution time (upper line) and RTE (lower line) for MODP on Trace 2, with inbound traffic on the left and outbound traffic on the right.

Execution time improves as expected with a larger threshold, since fewer packets are processed. RTE savings improve initially as well, since fewer chunks from small packets occupy the cache. Naturally, savings decrease beyond a certain threshold size, since less and less of the network data is eligible for chunk caching. Negative values indicate that the RTE savings are worse than using no threshold at all.

Since our focus is on improving detected redundancy, the best threshold point to choose is where improvement in RTE peaks. This point is between 96 and 128 bytes for inbound traces, and between 64 and 96 for outbound traces. To maximize the benefit, the threshold should be carefully adjusted for the type of traffic on the particular link, perhaps even over time, as traffic may change.

We choose 96 bytes as the threshold[1] for size-based bypass, providing a compromise between the two traffic directions, and consistent benefits across our university traces. This setting offers significant improvement in execution time, and slightly *higher* detected redundancy.

We have evaluated the benefits of this approach by comparing the RTE and execution time results for all traces. In the best case (MODP, small sampling period, large cache) the benefits from bypassing small packets are up to 25.8% improvement in execution time, and up to 4.2% improvement in RTE. In the worst case (WINN, $p = 35$, 64-byte chunks, 500 MB FIFO cache), the improvement in average execution time per trace is 2.1%, and RTE remains the same.

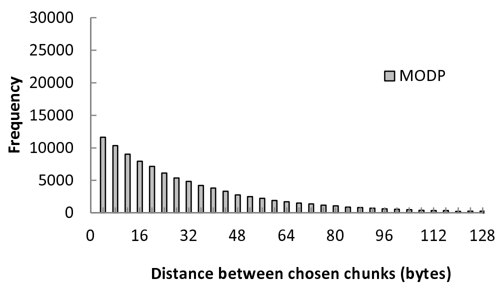
4.2. Chunk Overlap

In previous work, selection of a chunk is followed by a matching region expansion around the chunk to achieve the largest possible match [1, 3, 14]. Expansion introduces overhead in processing cost and storage, but improves average detected redundancy by 13.6% [1].

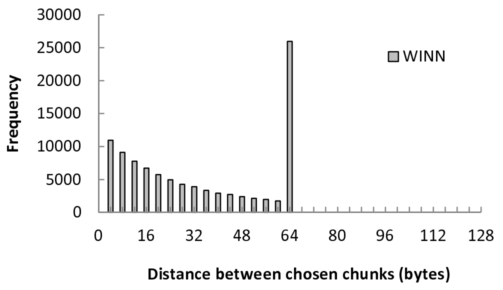
For expansion to make sense, selected chunks should be disjoint, with a gap between them that is filled by expanding the matching region. However, ensuring that such a gap exists will reduce the number of selected chunks in many cases, especially when the sampling period is greater than or equal to the chunk size. In the simplest case of FIXED selection algorithm, 32 byte chunks with sampling period of 32 will cover the block of data completely, without gaps and without overlap. Hence, introducing gaps to enable expansion necessarily reduces the number of initially selected chunks. Fewer chunks lead to lower detected redundancy, since it increases the sampling period.

We explore a simple alternative that covers more data chunks and allows overlap of chunks selected for caching. We start with overlapping chunks, and then prune chunks whose overlap exceeds a threshold, to avoid selecting chunks for which most bytes overlap.

[1]A sample of our trace data shows that nearly 50% of TCP packets are smaller than 96 bytes, with the vast majority of the data bytes carried in the larger packets.



(a) MODP



(b) WINN

Figure 5: Distance between selected chunks

With both MODP and WINN, selected chunks may overlap either because of the selection criteria or relationship between chunk size and sampling period. MODP selects chunks based on fingerprint values ending in a certain pattern, whereas WINN selects based on fingerprint value only. Overlap of chunks naturally occurs in both of these algorithms, as seen in Figure 5, where we show the distribution of distance between consecutive chosen chunks. The distance is the number of bytes between the initial bytes of two chunks. The tail of the MODP distribution extends well over 400 bytes, though the graph is truncated at 128. This confirms that WINN chooses chunks more uniformly, and always within a specified sampling period, whereas MODP can skip long blocks of data.

Overlap can also occur depending on the relationship between chunk size and sampling period. For example, a very large sampling period of 512 and chunk size of 4 would rarely choose overlapping chunks. Another extreme would be a small sampling period of 1, where every chunk is selected, and overlap would occur for every chunk size larger than 1.

Intuitively, overlap should be avoided, especially if expansion of the matching region is done. Overlapping chunks do not offer the full savings of a whole chunk, and additional processing overhead is required.

Surprisingly, overlapping chunks can actually *improve* RTE at a small processing cost, when appropriately parameterized. We start by comparing detected redundancy and execution time for different overlap thresholds on Trace 3 inbound traffic. The overlap thresholds are (1) *None*: no overlap allowed, (2) *Half-Chunk*: overlap allowed up to one half of chunk size (i.e., 32 bytes of overlap for 64-byte chunks), and (3) *Any*: any amount of overlap is allowed, up to the chunk size less 1 byte.

Table 6: Comparison of different chunk overlap thresholds

RTE	Cache	Overlap Redundancy			Overlap Time		
Algorithm	Size (MB)	None	Half-Chunk	Any	None	Half-Chunk	Any
WINN	1000	11.7%	13.0%	13.4%	2373	2545	3049
MODP	1000	10.1%	11.2%	11.2%	1295	1480	1986
MODP	infinite	12.0%	14.0%	14.9%	1236	1314	1531

From Table 6, we find that half-chunk overlap increases detected redundancy by 11.1% to 16.7%, depending on the algorithms and cache size, but execution time also increases by 6-14%. Allowing any overlap can increase detected redundancy even more, but the cost in execution time is too high, up to 53%, which is not acceptable. We therefore adopt the threshold of half-chunk overlap, and proceed with evaluation on all traces, using WINN, $p = 35$, 64-byte chunks, 500 MB cache and FIFO replacement policy.

Table 7 shows that allowing up to half-chunk overlap consistently improves detected redundancy with a slight penalty in execution time for each trace. Actually, an 8.7% average improvement in detected redundancy per trace has a cost of 9% in execution time for inbound traffic. A similar tradeoff exists for outbound traffic. We find this tradeoff acceptable.

The benefits of chunk overlap diminish with either small (4 or 8) or large sampling periods (128 and higher). With small sampling periods, too many chunks are selected, and nearly all of them overlap by at least half, which degenerates to FIXED selection. Long sampling periods select so few chunks that they rarely overlap, and the threshold becomes irrelevant. Therefore, controlled half-chunk overlap makes sense only when sampling period is within 1/4 to 1/2 of the chunk size.

Table 7: Benefits of allowing chunk overlap

Packet	Inbound Savings		Inbound Time	
Trace	None	Half-Chunk	None	Half-Chunk
1	19.1%	20.5%	5453	5805
2	19.3%	20.5%	2327	2384
3	10.8%	11.9%	6850	7425
4	13.3%	14.7%	5129	5551
5	9.1%	10.2%	2493	2757
6	11.9%	12.7%	5357	5853
7	10.6%	11.7%	2158	2393
8	7.3%	8.1%	6307	7166
Average	12.7%	13.8%	4509	4917
Packet	**Outbound Savings**		**Outbound Time**	
Trace	**None**	**Half-Chunk**	**None**	**Half-Chunk**
1	22.2%	23.4%	4257	4828
2	13.9%	15.0%	4318	4759
3	12.3%	13.6%	4829	5204
4	10.3%	11.4%	3562	4048
5	19.0%	20.5%	2297	2447
6	14.6%	16.1%	3511	3679
7	9.5%	10.5%	3174	3164
8	14.4%	16.1%	4405	4596
Average	14.5%	15.8%	3794	4090

To make sure that half-chunk overlap indeed produces actual byte savings, we assess an encoding penalty of 5 bytes per 64-byte chunk and evaluate byte

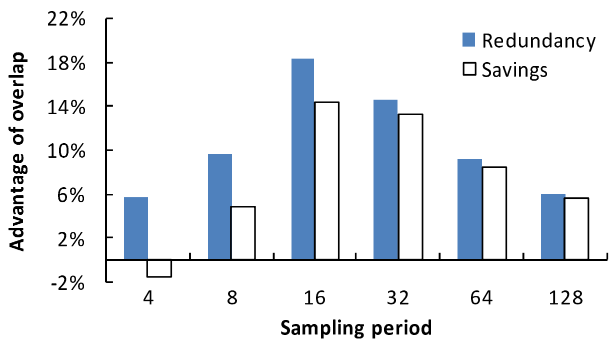


Figure 6: Benefits of chunk overlap

savings for WINN. For all traces, the savings with overlap are indeed higher than without overlap (Figure 6).

In summary, overlap improves RTE in a similar way to chunk expansion, by covering more consecutive matching bytes. It incurs extra overhead by processing more chunks than without overlap, but expansion imposes higher processing cost as well. It should be noted that both approaches must process every chunk using the sliding window to accommodate the Rabin fingerprinting algorithm, regardless of whether a previous chunk expanded the matching region or not. Another advantage of overlap is its reduced storage overhead. Expansion requires full data packets to be stored in the cache regardless of the number of chunks selected from each packet, with many packets providing no savings. The overlap approach, with a fixed chunk size, uses the cache only for chunks, and not for any additional packet data.

4.3. Cache Replacement Policy

In this section, we discuss alternative cache replacement policies for the chunk cache. In previous works, a FIFO cache was assumed [3, 14]. Since our approach caches fixed-size chunks rather than full packets, we can exploit temporal locality and the non-uniform popularity of chunks.

We evaluate two policies other than FIFO to improve RTE. The comparisons between all cache replacement policies are based on WINN with $p = 35$, 64-byte chunks, half-chunk overlap, 1 GB cache, and packet-size bypass of 96 bytes.

4.3.1. LRU

Prior work showed that popular chunks exhibit temporal locality [3], and we find the same property in our traces. Given this behavior, we consider Least Recently Used (LRU) as a cache replacement policy. LRU removes the least recently used chunk when space is needed to add a new chunk to the cache.

Table 8 shows the RTE and execution time results for LRU, in a column adjacent to FIFO. The improvement in detected redundancy is negligible, which is a poor tradeoff with the significantly higher execution time.

Table 8: Comparison of FIFO, LRU, and LSA

Packet	Inbound Savings			Inbound Time		
Trace	FIFO	LRU	LSA	FIFO	LRU	LSA
1	22.0%	22.7%	23.2%	5544	7747	5430
2	22.3%	23.0%	23.6%	2203	3028	2177
3	13.5%	14.2%	14.7%	7190	9830	6956
4	16.4%	17.6%	17.8%	5182	7303	5122
5	11.5%	12.0%	12.5%	2475	3404	2501
6	14.2%	15.1%	15.3%	5337	7557	5381
7	13.0%	13.5%	14.1%	2119	2937	2130
8	9.1%	9.5%	9.7%	6325	9193	6393
Average	15.2%	15.9%	16.4%	4547	6375	4511
Packet	Outbound Savings			Outbound Time		
Trace	FIFO	LRU	LSA	FIFO	LRU	LSA
1	26.9%	27.7%	28.1%	4646	5786	4181
2	17.1%	17.7%	18.5%	4487	5738	4041
3	15.3%	16.0%	16.5%	5136	6678	4711
4	12.9%	13.3%	13.9%	3813	4906	3452
5	22.8%	23.5%	24.3%	2384	3025	2171
6	17.9%	18.6%	19.4%	3668	4631	3423
7	12.0%	12.5%	13.3%	3244	4169	2947
8	17.9%	18.3%	18.8%	4672	5991	4373
Average	17.9%	18.4%	19.1%	4006	5116	3662

4.3.2. Least Savings with Aging (LSA)

Since exploiting temporal locality was not beneficial for improving RTE, we turn to another non-uniformity in network traffic: the popularity of data chunks. Earlier work has shown that the popularity of chunks has a Zipf-like power-law distribution [3].

If some data chunks are more popular than others, then they should be kept in the cache as long as they are useful. This is analogous to the Least Frequently Used (LFU) replacement policy. Traditional LFU tracks cache hits for every object, and removes the one with the fewest hits when replacement is needed. In our implementation, where chunk overlap is allowed, it would be

Table 9: Contribution to detected redundancy by most popular chunk content types

Content type	**Redundancy**	**Description**	**Example**
Nulls	57.1%	String of consecutive null bytes	`0x00000000`
Text	16.7%	Plain text (English)	`Gnutella`
HTTP	7.3%	Fragment of HTTP directives	`Content-Type:`
Mixed	6.2%	Plain text and other characters	`14pt font`
Binary	5.8%	Seemingly random characters	`0x27c46128`
HTML	3.7%	Fragment of HTML code	`<HTML> <p>`
Char+1	3.2%	Many repeated text characters	`AAAAAAAz`

naive to rank the chunks solely by number of hits, since we may over-value some chunks whose contribution to detected redundancy is less than the full chunk size. It is thus not the number of hits that is our metric for ranking chunks, but rather the actual byte volume contributed to RTE. Our proposed cache replacement policy removes the chunk with the least RTE savings so far.

A common issue with LFU-based policies is *cache pollution*, wherein objects can stay too long in the cache, following a period of high activity. That is, their hit count becomes so high that they never get replaced. Solutions for this problem include limiting the maximum hit count, or introducing an aging factor. We use a form of aging that purges all chunks from the cache, hence our policy's name Least Savings with Aging (LSA).

Two important notes on LSA are as follows: (1) a large majority of chunks in the cache are recent chunks without any detected redundancy, which are often evicted soon; and (2) due to the temporal locality property of popular chunks, we can simply purge the entire cache periodically as a form of aging. We find that a good purging period is when the cumulative number of chunks processed is 5 to 10 times the number of cache entries.

The cache warm-up period is extremely short, as seen in Figure 7. We show the start of Trace 3, where detected redundancy reaches steady-state within the first 100 MB, and recovers quickly after complete purges were performed at 250 MB and 500 MB of processed trace data. Since popular chunks are temporally close, they populate the cache very quickly without significant effect on the overall detected redundancy.

Table 8 shows that LSA consistently produces higher RTE than FIFO. The average difference is 7.3% and 7.0% for inbound and outbound traffic, respectively. In addition, execution time for LSA is no worse than FIFO. LSA is faster in most cases, since eviction is simple, and it handles less data than other policies, due to purging. Since evicted chunks in LSA generally have no detected redundancy yet, and there are many of them, they are stored and evicted in a FIFO manner.

Overall, LSA performs better than FIFO and LRU. LSA uses a simple method for aging, and exploits the fact that non-redundant chunks are numerous, and almost always are the ones that get evicted.

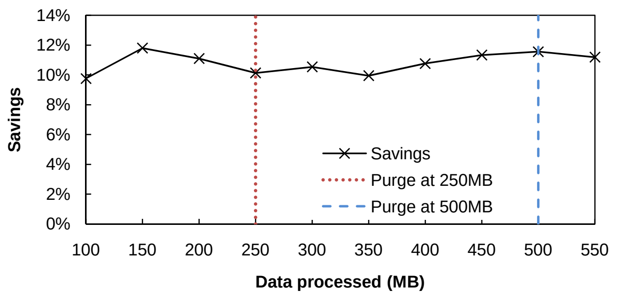


Figure 7: LSA cache has fast warmup

Table 10: Redundancy and execution time before and after improvements

Packet Trace	Before improvements		After improvements	
	Redundancy	Time	Redundancy	Time
Campus In	10.8%	5111	15.5%	5052
Campus Out	12.0%	4274	17.7%	4056
Residential	6.3%	15075	8.2%	13996
Web server	25.7%	2548	40.3%	2625

We have confirmed that the benefits of LSA continue to hold when using a small cache of 100 MB and without chunk overlap (not shown). The advantage of LSA in detected redundancy was 11.7% and 10% for inbound and outbound traffic, respectively, while execution time was slightly faster.

4.4. Summary

The cumulative improvement in detected redundancy using network-layer improvements, compared to the base case, is illustrated in Table 10. The results for all traces are presented as averages per trace.

The absolute increase in RTE due to the individual techniques is about 2-5% for the campus traces, because the level of redundancy in our campus traces is modest. However, the relative increase in RTE, as well as the cumulative effects of the improvements, are significant, particularly since they have minimal computational cost.

To elaborate, the average RTE for campus traces of 11-12% byte savings typically achieved with existing RTE approach can be improved to 16-18% with our combined techniques. This represents a relative improvement of 45-50%. Average RTE for residential traces is improved from 6.3% to 8.2%, which is a 30% relative improvement. For enterprise Web traffic, RTE savings improve from 26% to 40%, which is a 54% relative improvement. Campus and residential traces show a small reduction in execution time, wheres Web server traces take slightly longer to process than in the base case.

The key result of this section is that the network-layer improvement techniques are beneficial and lead to higher detected redundancy for a variety of environments with different traffic mixes, including low-redundancy residence traffic and highly-redundant Web server traffic.

5. Content-Aware RTE

In this section, we consider improvements to end-system RTE. Moving the RTE process from the network to the end system was proposed in earlier work [1]. We propose a content-aware approach to RTE, and quantify its benefits using a separate set of standalone experiments, as well as traffic traces.

5.1. A Posteriori Analysis

The fundamental question in RTE is: how do we choose the best chunks to cache, and do so without having to fingerprint every possible chunk?

Protocol-independent RTE means that the chunk selection algorithm is agnostic of the content type and application that generates the data. All data is fingerprinted regardless of the content type, and chunks for caching are selected from all applications indiscriminately. However, not all applications generate traffic of equal redundancy, nor do they contribute equally to the overall detected redundancy. This is yet another manifestation of traffic non-uniformity.

To provide greater insight into the fundamental RTE question, it is instructive to analyze the content of the most popular chunks observed. We select from all traces the chunks that remained in the cache after RTE processing, and contributed the most to RTE savings. We identify the Top 100 chunks, each of which contributed at least 48 KB of actual RTE savings. As found in [3], the most popular chunk is a string of null bytes, which contributed 57.1% of byte savings within our Top 100. The other top chunks, which differ from those observed in [3], are classified into several categories in Table 9.

The results in Table 9 show that much of the redundancy can be detected by searching for strings of zeros. However, there is another important finding as well. In particular, text-based data (Text, HTTP, HTML, and Char+1) in aggregate contribute 30.9%. Only 12% of detected redundancy is contributed by other non-text data (Mixed and Binary).

5.2. Toward Content-Awareness

The low RTE savings for non-text data suggest that such data could be bypassed. Bypassing useless data would save CPU cycles and cache space that could be used for additional sampling of (highly-redundant) text-based data. Doing so would also mean that the cache contains more useful chunks. The two applications that contribute the most bytes on typical access links are Web and file-sharing. The former is primarily (but not exclusively) text-based, while the latter is not.

As a proof-of-concept, we conduct a standalone experiment on a separate library of data files. We use a collection of HTML files to represent text-based content, PDF files to represent mixed content, and MP3 files to represent non-text content (e.g., encoded, compressed, or encrypted objects).

Table 11 provides details on each file set, as well as the redundancy detected. The HTML set contains all pages of a major university Web site, which may contain some duplicate pages. The PDF set contains networking literature, class assignments, and various software manuals, with 5 files having partially repeated content. The MP3 files are all distinct. Therefore, any (intra-object or inter-object) redundancies found do not arise from repeated objects in the PDF and MP3 file sets.

When fingerprinted separately using 64-byte chunks, overlap, MODP 32 and WINN 35, with a 64 MB FIFO cache, all three file types show vast difference in redundancy. The HTML files show high redundancy (72-77%), while the PDF files show moderate redundancy (11-13%), and the MP3 files little redundancy (0.4%). When data sets are combined in a pair-wise fashion (i.e., two at a time), the HTML-MP3 redundancy is 36.5% for MODP and 38.6% for WINN, whereas PDF-MP3 redundancy is 5.7% for MODP and 6.4% for WINN. The MP3 files, with negligible redundancy, clearly reduce the overall RTE savings in the combined data sets.

5.3. Ideal bypass

We next implement an ideal bypass technique at the file granularity, where content type can be determined at the end system. This approach is based on known file types, where we oversample the redundant file type and bypass the non-redundant files type, with the goal of achieving higher overall RTE.

From Table 8, the ideal bypass approach improves RTE for the HTML-MP3 set to 44.1% for both algorithms. This represents a 20.7% improvement for MODP, and a 13.7% improvement for WINN. In the PDF-MP3 case, the RTE results are 7.7% for MODP and 7.6% for WINN. The improvement for MODP is 35.7%, while that for WINN is 18.7%. MODP and WINN achieve nearly the same RTE, but MODP is faster.

These results demonstrate that content-aware RTE at end systems is beneficial. It detects more redundancy, and it can also reduce processing time, compared to the no-bypass case. It is also worth noting that we have applied all network-layer improvements already (except LSA and size-based bypass), and added content-awareness to the RTE pipeline process. This shows the additive effect of network-layer improvements and content-awareness for end-system RTE.

Table 11: Content-aware and file-based bypass on HTML, PDF and MP3 data sets

Data set	Files	Data (MB)	MODP 32	WINN 35	MODP	WINN
HTML	5340	132.7	72.1%	76.5%	18.0 s	39.0 s
PDF	248	132.3	11.0%	12.5%	20.9 s	44.3 s
MP3	40	130.7	0.4%	0.4%	21.7 s	43.6 s
Combination of data sets without bypass						
HTML:MP3	5380	263.4	36.5%	38.8%	42.4 s	88.9 s
PDF:MP3	288	262.9	5.7%	6.4%	45.7 s	88.6 s
Combination of data sets with ideal file-based bypass						
			MODP 4	WINN 5		
HTML:MP3	5380	263.4	44.1%	44.1%	54.5 s	65.4 s
PDF:MP3	288	262.9	7.7%	7.6%	71.2 s	78.5 s
Combination of data sets with content-aware bypass						
			MODP 4	WINN 5		
HTML:MP3	5380	263.4	43.3%	43.4%	174.6 s	159.1 s
PDF:MP3	288	262.9	7.0%	6.9%	199.5 s	205.8 s

5.4. Text-based bypass

The next step is to apply content-awareness to network-layer RTE. Achieving ideal bypass of non-redundant content type by middle-boxes at the network layer would require determining the content type of the payload of each packet. Simply determining the higher layer protocol inside the TCP connection is not sufficient. For example, HTTP may carry different content types, such as text-based HTML and non-text images, within persistent connections. A simpler solution, one that only requires inspection of the current packet, is required.

To translate the ideal bypass into the RTE process at the network layer, we need to use some characteristic of the data chunk that would provide an indication of the underlying file type (i.e., a form of content-awareness). For example, we could use entropy, or the proportion T of plain-text characters within the data chunk, for which we coin the new term "textiness". We know from Table 9 that null-strings and text-based chunks account for most of the detected redundancy. In our file sets, however, null-strings are rare, so we focus on textiness instead.

We start by calculating textiness using characters whose ASCII values[2] are 32 to 126 (inclusive), for all data chunks in the HTML and MP3 sets. Their distributions are shown in Figure 8(a). The expected distinction is clearly shown, with HTML chunks having numerically high textiness, compared to MP3 files. The majority of MP3 chunks have $T \in (0.3, 0.45)$. It is clear that RTE should bypass all data chunks with $T < 0.9$, so that is what we implement for content-awareness for the HTML-MP3 data set.

The results are shown in the bottom part of Table 11. For this data set, the simple implementation of content-awareness achieves nearly the same RTE as the ideal file-based bypass for both MODP and WINN. However, processing times increase significantly (79% for WINN), which is one challenge for content-aware RTE. There are, however, two possible solutions for the high processing cost. One way to reduce processing cost is to track chunk content concurrently with fingerprinting, as opposed to fingerprinting *a posteriori* as in our current implementation. Another improvement would be to apply the sliding window approach to calculation of textiness (or entropy), by removing the first byte in the chunk from the calculation of textiness and adding the new byte as the sliding window advances.

Another challenge becomes evident when we consider the PDF-MP3 data set next. Figure 8(b) shows the histograms of textiness for PDF and MP3 files. The similarity of distributions is clear, but we know that PDF files contain more redundancy. There is no clear-cut difference in textiness of PDF and MP3 files as there was between HTML and MP3 files. We choose to oversample and select chunks with textiness of (0.3, 0.45] and (0.9, 1].

As seen in the bottom rows of Table 11, the detected redundancy did not achieve the same levels as ideal file-based bypass. The RTE benefit was moderate for MODP, but small for WINN. The reason for lower RTE is that MP3

[2]The text-based implementation of content-awareness relies on ASCII values, which are appropriate for English content. Entropy can be used to find redundancy in other languages when characters are represented using encodings other than ASCII. Based on our experiments, entropy offers the same savings as textiness, but at higher processing cost.

chunks still get selected, even though they are not redundant.

5.5. Content-aware RTE at network layer

The standalone experiments with file sets demonstrated the benefit of content-awareness for end-system RTE, but revealed the limitations of a bypass approach if applied at the network layer. Rather than using a single sampling period and a static configuration of textiness values to bypass, we develop a heuristic that uses multiple sampling periods.

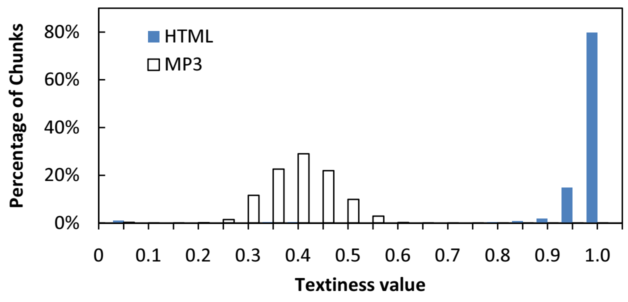
Figure 8(c) compares textiness of chunk hits with that for all chunks on campus traces. Both distributions are multi-modal. The distribution for all chunks is concentrated around 0.4, with a slight peak at 0, and additional peaks beyond 0.9. The distribution for hits looks similar, but with distinct differences. Null-strings whose textiness is 0 produce many cache hits, approximately 5 times more than their proportion of traffic volume. Similarly, plain-text chunks (near 1.0) get about 4 times more hits than their proportion of traffic volume. In these cases we gain RTE savings. However, chunks around 0.35 account for most of the data volume, yet produce proportionally fewer hits; this is where RTE savings are lost.

It is clear that we should not completely bypass some chunks in favor of others, since most of the chunks offer some savings. Instead, we use multiple sampling periods depending on the ratio of hits to all chunks for each textiness level. We first tried to implement multiple sampling periods with MODP, which was impractical due to coarse (powers of 2) granularity of MODP sampling. WINN, however, is a better choice, and allows fine-grained adjustment of the sampling period. Following the distribution of textiness values from Figure 8(c), we sample chunks according to the following criteria:

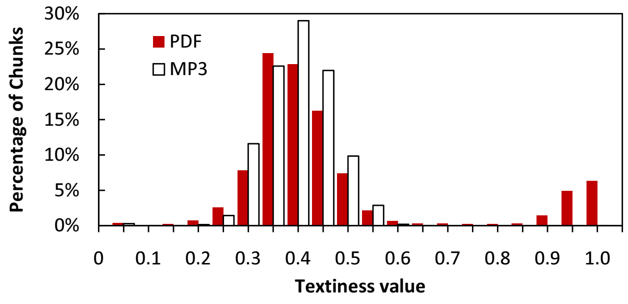
- if $T \in [0.0, 0.2)$: p = 8,
- if $T \in [0.2, 0.3)$: p = 16,
- if $T \in [0.3, 0.5)$: p = 50,
- if $T \in [0.5, 0.8)$: p = 16,
- *otherwise*: p = 8.

Chunks are oversampled if they generate more hits than their proportional volume (sampling period reduced from 35 to 16), and are undersampled otherwise (sampling period increased from 35 to 50). A sampling period of 16 provides very high coverage of data when 32-byte chunks are used without overlap. For extreme (low or high) textiness values, where chunks are highly redundant, we reduce the sampling period from 35 to 8 to ensure even fuller coverage of the data.

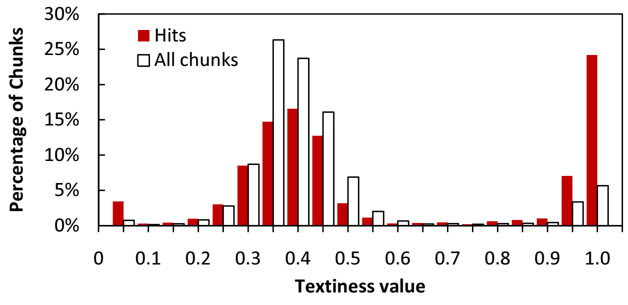
When applied to real traffic traces, the adjustment of sampling period works as follows. Once a chunk is selected, the sampling period is adjusted to the level corresponding to that chunk's textiness, and sampling with that period continues with the expectation that subsequent chunks will have similar textiness.



(a) HTML and MP3 files



(b) PDF and MP3 files



(c) Cache hits and All chunks

Figure 8: Distribution of textiness for the file sets

Table 12: Redundancy and execution time with and without content-awareness

Packet	Content-unaware		Content-aware	
Trace	Redundancy	Time	Redundancy	Time
1	18.8%	4787	25.1%	7816
2	11.8%	4752	15.7%	7085
3	9.8%	5498	15.0%	8598
4	8.4%	4022	12.0%	5804
5	15.9%	2486	21.2%	3953
6	11.8%	3966	15.8%	5817
7	7.9%	3572	11.4%	4948
8	11.4%	5107	15.0%	7363
Average	12.0%	4274	16.4%	6423
A	6.2%	15554	7.2%	20744
B	6.5%	14596	7.6%	19578
Average	6.4%	15075	7.4%	20161

The sampling period is adjusted again after selecting a chunk whose textiness merits a different sampling period. While this heuristic is not a perfect solution, it allows us to achieve similar number of selected chunks so we can fairly compare RTE savings with and without content-awareness.

Table 12 shows a comparison of RTE savings and execution time for campus and residential outbound traces. (Results for campus inbound traces are similar.) The average improvement for campus traces is from 12% to 16.4%, and for residential traces from 6.4% to 7.4%. Content-aware RTE is, therefore, beneficial at network layer, but not as much as network-layer improvement techniques. It is also peculiar that when content-awareness and network-layer improvements (described in Section 4) are applied together at network-layer, the additive effect is very small compared to the case where content-awareness is applied at the end system. Therefore, content-aware RTE is best applied at the end system, where it provides higher savings and reduced processing cost, rather than at network layer, where savings are similar, but processing cost increases.

5.6. Summary

The presented results strongly indicate that content-aware RTE is a good approach to increase detected redundancy. The improvements achieved by content-awareness at the file level make a strong case for implementing RTE at end systems, where content type can be determined more easily. Information about the content type could be passed with the data to the socket layer or transport layer, and used for RTE purposes. The RTE benefit would be reflected in transmitting fewer IP packets, as opposed to reducing the size of packets when using network-layer RTE [1].

6. Conclusions

In this paper, we propose several techniques for improving network-layer redundant traffic elimination (RTE) by exploiting non-uniformities in network traffic with respect to packet size, chunk popularity, and content type. In addition, we demonstrate the benefits of using larger chunks, chunk overlap, and a savings-based cache replacement policy. Our proposed network-layer improvements apply to many chunk selection algorithms, and may be used individually or combined. As illustrated in Table 10, the relative improvement in average detected redundancy per trace is up to 50% for campus traffic, and up to 54% for enterprise Web server traffic.

In addition, we introduce content-aware RTE using file type and textiness of data chunks, and demonstrate its benefits for redundancy detection. When applied at end systems, content-awareness improves detected redundancy by up to 36% on top of network-layer techniques.

References

[1] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "EndRE: An End-System Redundancy Elimination Service for Enterprises", *Proceedings of USENIX NSDI*, San Jose, CA, pp. 419-432, April 2010.

[2] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination", *Proceedings of ACM SIGCOMM*, Seattle, WA, pp. 219-230, August 2008.

[3] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in Network Traffic: Findings and Implications", *Proceedings of ACM SIGMETRICS* Seattle, WA, pp. 37-48, June 2009.

[4] A. Anand, V. Sekar, and A. Akella, "SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination", *Proceedings of ACM SIGCOMM*, Barcelona, Spain, pp. 87-98, September 2009.

[5] M. Arlitt and C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications", *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5, pp. 631-645, October 1997.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", *Proceedings of IEEE INFOCOM*, New York, NY, March 1999.

[7] Cisco, "WAN Optimization and Application Acceleration", http://www.cisco.com/en/US/products/ps6870/.

[8] F. Douglis and A. Iyengar, "Application-specific Delta-encoding via Resemblance Detection", *Proceedings of USENIX Technical Conference*, San Antonio, TX, pp. 113-126, June 2003.

[9] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/Realtime Traffic Classification Using Semi-Supervised Learning", *Performance Evaluation*, Vol. 64, No. 9-12, pp. 1194-1213, October 2007.

[10] P. Kulkarni, F. Douglis, J. Lavoie, and J. Tracey, "Redundancy Elimination within Large Collections of Files", *Proceedings of USENIX Technical Conference*, Boston, MA, pp. 59-72, June/July 2004.

[11] A. Muthitacharoen, B. Chen, and D. Mazieres, "A Low-bandwidth Network File System", *Proceedings of ACM SOSP*, Lake Louise, AB, Canada, pp. 174-187, October 2001.

[12] M. Rabin, "Fingerprinting by Random Polynomials", Technical Report TR-CSE-03-01, Center for Research in Computing Technology, Harvard University, 1981.

[13] S. Schleimer, D. Wilkerson, and A. Aiken, "Winnowing: Local Algorithms for Document Fingerprinting", *Proceedings of ACM SIGMOD*, San Diego, CA, pp. 76-85, June 2003.

[14] N. Spring, and D. Wetherall, "A Protocol-independent Technique for Eliminating Redundant Network Traffic", *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, pp. 87-95, August 2000.

[15] T. Suel, P. Noel, and D. Trendafilov, "Improved File Synchronization Techniques for Maintaining Large Replicated Collections over Slow Networks", *Proceedings of ICDE*, location, pp. 153-164, March/April 2004.

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