Exploiting Non-Uniformities in Redundant Traffic Elimination

Emir Halepovic Carey Williamson Majid Ghaderi
Department of Computer Science
University of Calgary
email: {emirh, carey, mghaderi}@cpsc.ucalgary.ca

ABSTRACT

Protocol-independent redundant traffic elimination (RTE) at the network layer is a method of detecting and removing redundant chunks of data from data packets 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. Both bypass techniques are effective in reducing the processing requirements of RTE, with little or no adverse impact on redundancy detection. The bypass techniques apply at the front-end of the RTE pipeline. Within the RTE pipeline, we propose chunk overlap and oversampling as techniques that can improve redundancy detection, while obviating the storage and processing requirements associated with chunk expansion at the network endpoints as suggested by previous research. Finally, we propose savings-based cache management at the back-end of the RTE pipeline, as an improvement to the commonly used FIFO-based cache management. We evaluate our techniques on full-payload packet-level traces from a university environment. Our results show that the 11-12% savings achieved with typical RTE can be improved to 16-18% with our techniques.

Keywords: Traffic redundancy elimination, cache management, measurement.

1. INTRODUCTION

As Internet traffic volumes have continued to grow in recent years, redundant traffic elimination (RTE) has attracted more and more research attention [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., 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 IP 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].

In practical implementations of RTE, the data chunk size, the chunk selection algorithm, the sampling period, and the cache replacement policy are important factors that influence the effectiveness of the approach (i.e., the byte savings achieved on the network link). In the literature, the bandwidth savings reported for RTE are typically 10-12% for university campus Internet traffic, and sometimes as high as 30-60% for outbound traffic from a single organization [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, and savings-based cache management. We evaluate these techniques on full-payload packet traces from a university environment, and demonstrate their effectiveness. We also carefully investigate the factors mentioned previously (i.e., chunk size, chunk selection, sampling period, and cache management) to quantify their RTE benefits.

The unifying theme in our work is the exploitation

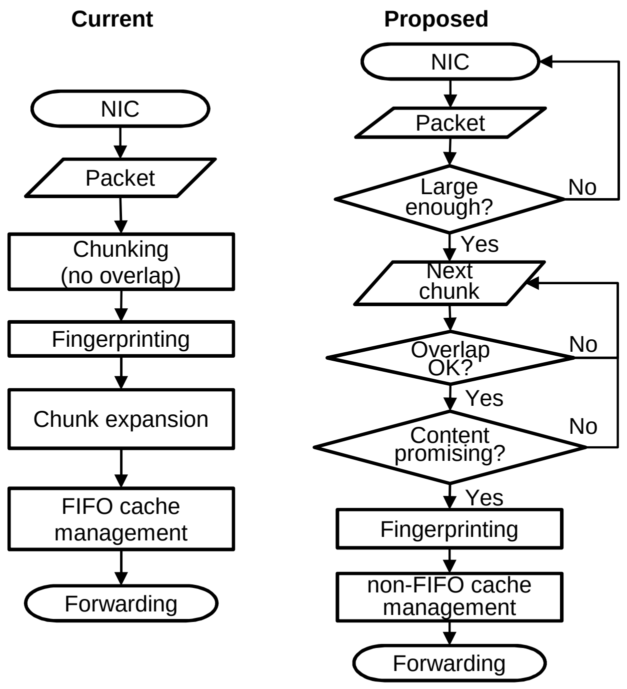


Figure 1: Processing pipeline for RTE

of *non-uniformities* in Internet traffic. For example, the bimodal distribution of Internet packet sizes 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 motivates a savings-based approach to cache management that retains the most valuable chunks, in terms of cumulative byte savings. 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%.
- Content-aware RTE can improve redundancy detection by 14-38%.
- The cumulative effects of the aforementioned techniques can boost existing RTE performance from 11-12% savings to 16-18% savings, representing an improvement of 40-50%.

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 improvements 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]. 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.*

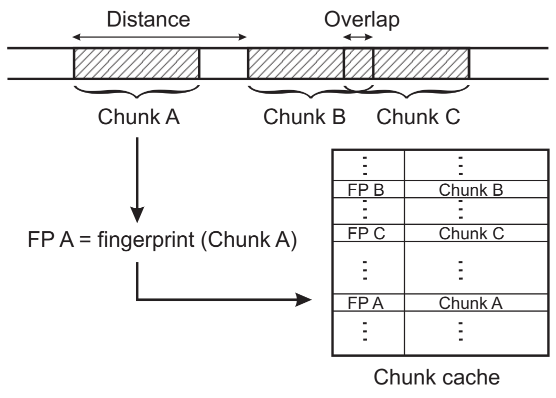


Figure 2: RTE terminology

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 may consist of fingerprints or other information sufficient to reconstruct the whole chunk using the client-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, not just HTTP. 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 based 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.

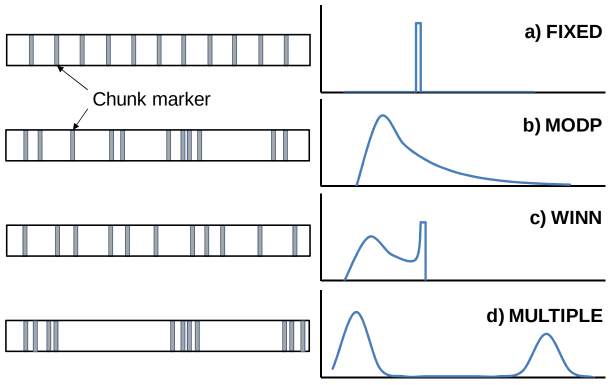


Figure 3: Example chunk selection algorithms

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-level RTE [14]. It selects chunks for which the fingerprint value is perfectly divisible by the sampling period (i.e., equal to 0 mod p). The MODP method is robust to small changes in objects (files or packets), 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 are skipped.

WINN: Winnowing is a way to filter or "thin" 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. 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, about 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 chooses it based on the data content bytes of the packet. 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: no sampling at all for "useless" content, and *oversampling* for the most 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 is not very redundant. The details of this approach are discussed later in the paper.

3. EVALUATION METHODOLOGY

In this section, we set the stage for our research by describing our data sets and methodology, as well as configuration parameter settings for RTE evaluation.

3.1 Data Sets

We evaluate our proposed RTE approach using full-payload packet traces collected from the Internet access link at the University of Calgary. A total of 8 one-hour traces were collected, starting at 9 am and 9 pm on each day in one week of April 2006. All traces are bi-directional. The total IP payload transmitted is 233.6 GB. The details of each trace are shown in Table 1.

While the traces are a few years old (from 2006), there are three reasons why we use them in our experiments. First, the traces are from a university environment comparable to that 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. Figure 4 summarizes the application profile of inbound and outbound traffic, averaged over all of the traces. Our traces have similar composition as those analyzed in [3].

We further divide traces 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 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 the original proposal in [14]. We compute Rabin fingerprints, and select them from within a sampling period of 32, the same as the baseline 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 to the "index off by one" special case. Most of our experiments use default settings from previous work, such as 32-byte chunk size, 500 MB cache, and FIFO cache replacement [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]. Ex-

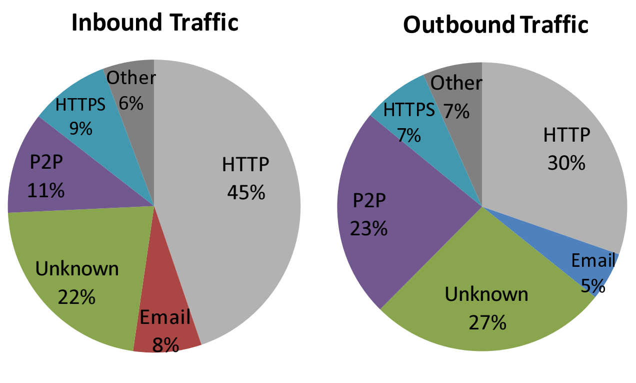


Figure 4: Application profile for all traces

Table 1: Characteristics of the data traces (GB)

Trace	Collection Date	Time	Inbound	Outbound	Total
1	Thursday, April 6, 2006	9:00am	19.9	15.5	35.4
2	Thursday, April 6, 2006	9:00pm	8.2	14.9	23.1
3	Friday, April 7, 2006	9:00am	23.6	16.9	40.5
4	Friday, April 7, 2006	9:00pm	18.3	12.5	30.8
5	Saturday, April 8, 2006	9:00am	8.8	8.2	17.0
6	Saturday, April 8, 2006	9:00pm	18.9	12.3	31.2
7	Sunday, April 9, 2006	9:00am	7.7	10.6	18.3
8	Sunday, April 9, 2006	9:00pm	21.9	15.4	37.3

pansion 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 requiring 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 chunk selection algorithms have only been compared on HTML pages [13], and not on network traces. A comparison of MODP and MAXP is presented in [3], showing that MAXP detects 5-10% greater redundancy in most cases, and up to 35% in specific cases. That study used 32-byte chunks and a sampling period of $p = 32$, with no comment on the execution time or number of chunks selected.

Our comparison explicitly considers the number of chunks selected, as well as the execution time of each algorithm. A more careful comparison considers one algorithm 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.

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.

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.

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. 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 or impossible to obtain, we are satisfied when they are within 2%.

Based on our experiments with one trace, the corresponding values for sampling periods are shown in Table 2. We use these sampling periods in our further comparison of MODP and WINN.

Table 3 compares the detected redundancy and execution time of these two algorithms for all 8 traces with our baseline parameters. For both inbound and outbound traffic, WINN detects more redundancy. Com-

Table 3: Comparison of MODP and WINN algorithms

Packet Trace	Inbound Redundancy		Inbound Time		Outbound Redundancy		Outbound Time	
	MODP	WINN	MODP	WINN	MODP	WINN	MODP	WINN
1	16.3%	16.6%	3478	5969	18.3%	18.8%	2940	4787
2	9.9%	16.9%	1466	2433	8.4%	11.8%	2944	4752
3	8.8%	9.1%	4722	7808	9.4%	9.8%	3401	5498
4	9.8%	10.8%	3511	6004	7.9%	8.4%	2407	4022
5	7.3%	7.7%	1840	2862	15.2%	15.9%	1479	2486
6	6.3%	10.2%	3884	6281	9.5%	11.8%	2339	3966
7	8.2%	8.7%	1548	2428	7.5%	7.9%	2099	3572
8	5.8%	6.1%	4557	7099	11.0%	11.4%	2937	5107
Average	9.0%	10.8%	3126	5111	10.9%	12.0%	2572	4274

Table 2: Sampling periods that generate a comparable number of fingerprints

Algorithm	Sampling Period			
MODP	32	16	8	4
WINN	35	20	10	5

paring 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 more efficient in terms of execution time (expressed in seconds throughout the paper). In fact, one could argue that MODP detects more redundancy per second of execution time than WINN.

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

Table 4: Comparison of MODP and WINN for different sampling periods

Algorithm	Value of p	30 MB cache		131 MB cache	
		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

In the rest of the paper, we use WINN as our baseline, because of its higher 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 due to the differences in implementations and traces used.

3.3.2 *Chunk Size*

An important factor in RTE is the chunk size. There is an inherent tradeoff between chunk size and the effectiveness of RTE. With tiny 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 more worthwhile. However, 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.

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.

4. PROPOSED RTE IMPROVEMENTS

In this section, we present and evaluate our new features for RTE pipeline processing, as illustrated in Figure 1. We begin with the bypass technique at the front of the RTE pipeline, and then discuss results for chunk overlap and non-FIFO cache management. We defer the discussion of content-aware RTE to Section 5.

4.1 Size-based Bypass

When RTE uses a packet cache, as proposed in [14], every packet is cached at least once by selecting the first chunk, even when 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 consider packet size as a factor in RTE performance [2, 3, 14].

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 bypass many small ACK packets traversing the network. Furthermore, many data packets carry just a few data bytes, rendering them "useless" as well for potential RTE savings. Signalling, control information, and secure shell are examples of such packets. The RTE process running at IP layer should therefore ignore all packets that carry insufficient data bytes, regardless of the transport 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. An IP-layer RTE should also bypass the 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 implement an IP-layer lookup of the packet size and bypass all packets whose IP payload is less than

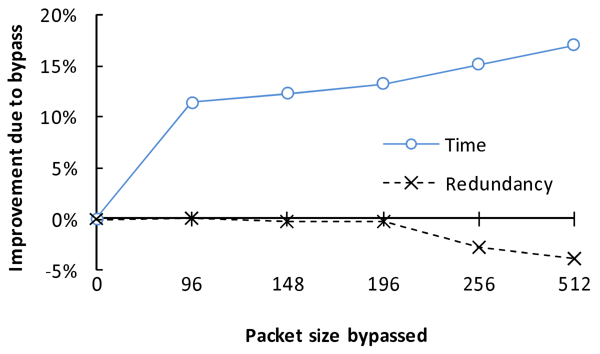


Figure 5: Effects of bypassing larger packets

96 bytes. We apply full RTE processing on the larger packets, which contain at least 64 bytes of payload beyond the extended TCP header of 32 bytes. A sample of our trace data shows that nearly 50% of TCP packets are smaller than 96 bytes, although the vast majority of the data bytes are carried in the larger packets.

Figure 5 shows the percentage improvement in execution time and RTE for Trace 1 inbound traffic, for different choices of the threshold value for packet-size bypass. Negative values indicate that the performance degrades. We can see that the benefits in RTE and execution time are minimal beyond a 96-byte threshold. In fact, at 148 bytes, detected redundancy starts to get slightly worse although execution time continues to improve. We choose the "knee" point in the Time curve at 96 bytes as the threshold for size-based bypass (Figure 5). This setting offers significant improvement in execution time, and slightly *higher* detected redundancy (since the chunk cache is cluttered with fewer useless chunks).

We evaluate the benefits of this approach, by comparing the RTE and execution time results for all traces. Results for the outbound direction are shown in Table 6, using WINN, $p = 35$, 64-byte chunks, and a 500 MB FIFO cache. For this combination of parameters, the improvement in average execution time per trace is 2.1%, and RTE remains the same. This is the worst case observed. For a larger cache, smaller sampling period, and MODP, the benefits from bypassing small packets are up to 25.8% improvement in execution time, and up to 4.2% improvement in RTE (not shown here).

4.2 Chunk Overlap

The goal of our chunk selection algorithm is to choose the best chunks that are most likely to be redundant. 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

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

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

Table 6: Bypass results (threshold: 96 bytes)

Packet Trace	Outbound Savings		Outbound Time	
	None	Bypass	None	Bypass
1	22.2%	22.2%	4257	4215
2	13.9%	13.9%	4318	4191
3	12.3%	12.3%	4829	4808
4	10.3%	10.3%	3562	3559
5	19.0%	19.0%	2297	2197
6	14.6%	14.6%	3511	3479
7	9.5%	9.5%	3174	2979
8	14.4%	14.4%	4405	4274
Average	14.5%	14.5%	3794	3713

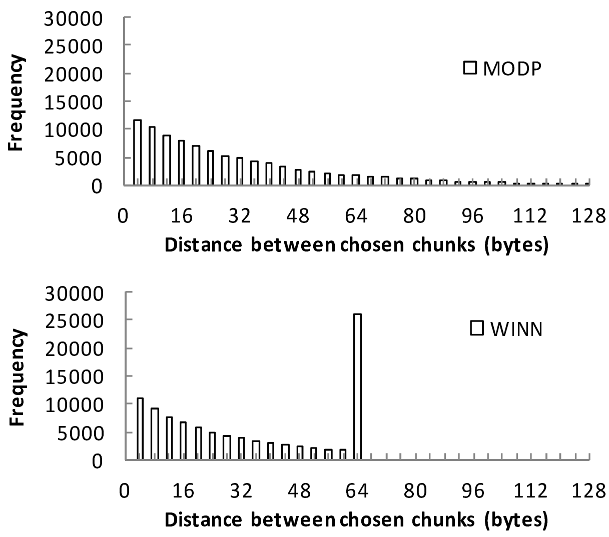


Figure 6: Distance between selected chunks

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 less 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 that would later be covered by expansion has to reduce the number of selected chunks first, and fewer chunks lead to less detected redundancy, since it essentially means an increase in the sampling period.

We explore an alternative and simpler approach that covers more data chunks and is based on allowing overlap of chunks that are 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.

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 value ending in certain pattern, whereas WINN selects based on fingerprint value only. Overlap of chunks naturally occurs in both of these algorithms, as seen in Figure 6, 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 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 matching region is done. Overlapping chunks do not offer full savings of a whole chunk and processing overhead is inevitable.

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.

From Table 7, we find that half-chunk overlap increases detected redundancy from 11.1% to 16.7%, depending on the algorithms and cache size, but execution time also increases. 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 8 shows that allowing up to half-chunk overlap consistently improves detected redundancy with a 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 exist for outbound traffic. We find this tradeoff acceptable.

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

To make sure that half-chunk overlap indeed produces actual byte savings, we use the encoding penalty of 5 bytes per 64-byte chunk and evaluate byte savings for WINN. For all traces, the savings with overlap are indeed higher than without overlap (not shown here).

4.3 Cache Replacement Policy

In this section, we discuss alternative cache replacement policies for the chunk cache. In previous works, FIFO cache was assumed to be adequate and it was adopted as the only cache replacement policy [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 that lead to higher 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 promising cache replacement policy. LRU replaces the least recently used chunk when a new chunk is added to the cache.

Table 9 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.

4.3.2 Least Savings with Aging (LSA)

Significant temporal locality in redundant data chunks has not proved beneficial for improving detected redundancy and execution time with LRU. Therefore, we turn to another non-uniformity in network traffic in a quest for a better solution: popularity of data chunks. Earlier work has shown that the popularity of chunks has a Zipf-like, power-law, heavy-tailed distribution [3].

If some data chunks are more popular than others, then it stands to reason that we should try and keep them 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 replaces the one with the fewest hits. In our implementation where chunk overlap is allowed, it would be naive to rank the chunks by number of hits, since we may over-rate many 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 least RTE savings so far.

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

Two important notes on LSA are as follows: (1) A very large majority of chunks in the cache are recent chunks without any detected redundancy, hence they are the ones getting evicted all the time, and (2) Due to the still useful temporal locality behavior of popular chunks, we can afford to simply purge the entire cache periodically as a form of aging. We find that a good purging period is when the cache has added 10 to 20 times as many chunks as its capacity.

Since popular chunks are temporally close, they populate the cache very quickly without significant effect on the overall detected redundancy, i.e. cache warm-up period is extremely short, as seen in Figure 7. We show the start of Trace 3, where detected redundancy

Table 7: 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

Table 8: Benefits of allowing chunk overlap

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

Table 9: Comparison of FIFO, LRU, and LSA

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

Table 10: 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%	Combination of plain text and other characters	`14pt font`
Binary	5.8%	Combination of apparent random characters	`0x27c46128`
HTML	3.7%	Fragment of HTML code	`<HTML> <p>`
Char+1	3.2%	Repeated text character followed by one different character	`AAAAAAAz`

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.

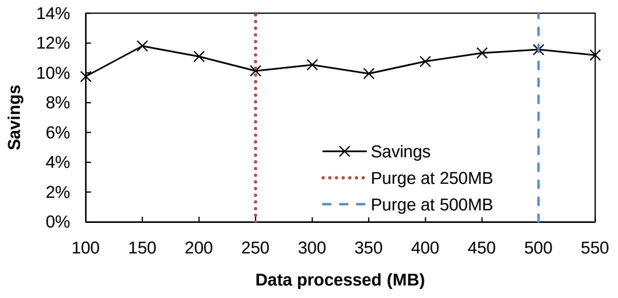


Figure 7: LSA cache has fast warmup

Table 9 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 similar to or faster than FIFO. LSA is faster in most cases because of our implementation. Since evicted chunks in LSA generally have no detected redundancy yet, and there are many of them, we make sure that they are evicted in a FIFO manner, while chunks with detected redundancy stay in the cache. This is implemented using a queue that stores only fingerprints and is shorter compared to the queue that stores full data chunks.

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

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.

5. CONTENT-AWARE RTE

In this section, we propose a content-aware approach to RTE, and quantify its benefits using a separate set of standalone experiments.

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 the application that generates or transmits 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 10.

The results in Table 10 show that most of the redundancy can be discovered 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 contributes 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 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. In terms of traffic volumes, the two applications that contribute the majority of bytes on 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 conducted a standalone experiment on a separate library of data files. We used a collection of HTML files to represent text-based content, PDF files to represent mixed content, and MP3 files as a representation of 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 for PDF and MP3 files.

When fingerprinted separately using baseline parameters 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 negligible redundancy (0.4%). When data sets are combined in a pair-wise fashion, 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. It is clear that MP3 files, with negligible redundancy, 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. This approach is based on file type, and uses oversampling of the redundant file type with the goal of achieving higher overall RTE.

The ideal bypass approach improves the 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 resulting RTE results are 7.7% and 7.6% for MODP and WINN, respectively. The improvement for MODP is 35.7%, while that for WINN is 18.7%. MODP has the advantage of higher overall detected redundancy and less CPU time used. However, the relative cost in CPU time is high for MODP, and CPU time actually drops for WINN.

5.4 Text-based bypass

Achieving ideal bypass by middle-boxes at IP layer would require determining the content type of the payload of each packet. Note that keeping track of the mapping between the TCP connection (IP addresses and ports) and its payload type is not sufficient. This is because a given TCP connection may carry different content types, such as text-based HTML and non-text images, in persistent HTTP connections. A simpler solution, one that requires inspection of the current packet only, is required.

To translate the ideal bypass into the RTE process at the IP 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 the proportion T of plain-text characters within the data chunk, for which we coin the new term "textiness". We know from Table 10 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 values 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 10, indicating that 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. This is the most important finding because this data set is statistically similar to the actual trace traffic in Figure 4.

5.5 Implementation challenges

We identify two main challenges to measuring textiness of a data chunk in practice. First, it is computationally expensive (Table 10). Second, the distribution of textiness of useful chunks does not always correspond to their inherent file type.

We show the histograms of textiness of PDF and MP3 files in Figure 8(b). The similarity of distributions is clear, but we know that PDF files contain more redundancy. When we implemented the content-aware bypass, the detected redundancy did not achieve the same levels as ideal file-based bypass, as seen in the bottom rows of Table 10. The RTE benefit was moderate for MODP, but marginal for WINN. The reason is that MP3 chunks still get selected, even though they are not redundant. This highlights a distinction between protocol-awareness and content-awareness. That is, just knowing that HTTP is in use is insufficient, since HTTP can transfer plain-text HTML files as well as non-text images and multimedia objects, not only within the same connection, but even the same packet.

The problem of using only the distribution of textiness values is further shown in Figure 8(c), where we compare textiness of chunk hits with that for all chunks. The distribution for all chunks is concentrated near 0.4, with a slight peak at 0, and additional peaks beyond 0.9. Null-strings whose textiness is 0 produce many cache hits, approximately 5 times more than their proportion of traffic volume. Similarly, full plain-text chunks (near 1.0) get about 4 times more hits than their proportionate traffic volume. However, chunks near 0.35 account for most of the data volume, yet produce proportionally fewer hits; this is where RTE savings are lost.

This result demonstrates the limitations of our textiness-based approach. One possible solution is effective payload-based traffic classification at the IP layer, but this topic is beyond the scope of our paper.

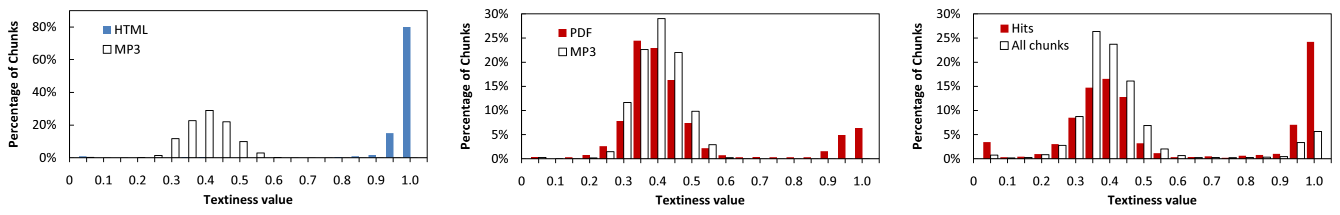
5.6 Summary

The presented results strongly indicate that content-awareness is a good approach to increase detected redundancy. We are working on a content-aware classification algorithm that improves detected redundancy at an acceptable computation cost and is applicable to RTE at IP layer. We also consider using multiple sampling periods depending on content type.

The improvements achieved by content-awareness at the file level make a strong case for implementing RTE at end systems, where file content could 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. Our results provide

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

Data set	Number of files	Total data (MB)	MODP 32	WINN 35	MODP Time	WINN Time
HTML	5340	132.7	72.1%	76.5%	18.0	39.0
PDF	248	132.3	11.0%	12.5%	20.9	44.3
MP3	40	130.7	0.4%	0.4%	21.7	43.6
Combination of data sets without bypass						
HTML:MP3	5380	263.4	36.5%	38.8%	42.4	88.9
PDF:MP3	288	262.9	5.7%	6.4%	45.7	88.6
Combination of data sets with ideal file-based bypass						
			MODP 4	WINN 5		
HTML:MP3	5380	263.4	44.1%	44.1%	54.5	65.4
PDF:MP3	288	262.9	7.7%	7.6%	71.2	78.5
Combination of data sets with content-aware bypass						
			MODP 4	WINN 5		
HTML:MP3	5380	263.4	43.3%	43.4%	361.5	387.0
PDF:MP3	288	262.9	6.4%	6.4%	433.0	446.9



(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

a supporting argument for end-system RTE, which was considered for a different reason in previous work [1].

6. CONCLUSIONS

In this paper, we propose several mechanisms for improving 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. All proposed improvements are applicable to many chunk selection algorithms, and may be used individually or combined. The cumulative improvement in average detected redundancy per trace, compared to the baseline case, is illustrated in Table 12. In particular, the 11-12% byte savings typically achieved with existing RTE solutions can be improved to 16-18% with our combined techniques. This represents an improvement of 45-50%.

In addition, we introduce the first attempt on content-aware RTE using file type and textiness of data chunks, and demonstrate its benefits for redundancy detection.

Our ongoing work is focusing on content-based chunk selection, as well as the applications of RTE to both upload and download directions in WLAN environments.

7. 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.

Table 12: Redundancy and execution time before and after improvements

Packet Trace	Inbound Traffic				Outbound Traffic			
	Before improvements		After improvements		Before improvements		After improvements	
	Redundancy	Time	Redundancy	Time	Redundancy	Time	Redundancy	Time
1	16.6%	5969	22.0%	5860	18.8%	4787	25.8%	4533
2	16.9%	2433	22.6%	2397	11.8%	4752	16.8%	4648
3	9.1%	7808	13.7%	7944	9.8%	5498	15.4%	5254
4	10.8%	6004	17.2%	5681	8.4%	4022	12.9%	3886
5	7.7%	2862	11.6%	2683	15.9%	2486	22.6%	2395
6	10.2%	6281	14.6%	5989	11.8%	3966	18.0%	3673
7	8.7%	2428	13.2%	2462	7.9%	3572	12.0%	3269
8	6.1%	7099	9.0%	7397	11.4%	5107	17.7%	4789
Average	10.8%	5111	15.5%	5052	12.0%	4274	17.7%	4056

[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.