Experimental Study of Protocol-independent Redundancy Elimination Algorithms

Maxim Martynov
Cisco Systems
maxim@cisco.com

ABSTRACT

The idea of identifying and removing repetitive patterns in the network data transfers, also known as protocol-independent *redundancy elimination*, and its benefits have received thorough consideration. However, actual implementation of such systems received much less attention. The intention of the redundancy elimination is to increase capacity of low-bandwidth network connections, when searching for redundancies and replacing them is faster than transmitting unprocessed redundant data. As long as network is slow, any reasonable implementation is beneficial. But as network capacities grow, the maximal throughput that system can provide becomes critical for its deployment. Thus, an appropriate choice of redundancy eliminating algorithm and its parameters becomes very important. This work addresses the problem of algorithm and parameter selection. We describe possible variations of the basic scheme, and demonstrate experiments that we have conducted for each variation. We discuss the trends observed in the results and explain their nature. We then propose a methodology to make a choice of the algorithm and its parameters, based on the obtained measurements.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Algorithms, Measurements, Performance

Keywords

Redundancy elimination

1. INTRODUCTION

The key ideas behind protocol-independent network redundancy elimination (RE) were originally proposed in [12] and [8], and remained unchanged so far. The basic principle of the RE approach is to identify repeating patterns in the outgoing traffic and replace them with labels, so that the original raw data could be reconstructed on the receiver side. The mechanism to find repetitions is borrowed from existing algorithms that look for similarities between files, such as the one proposed in [6]. The outgoing data is split into sub-strings (also referred to as *chunks*) and the redundancy is detected by looking for repeating chunks. The approach has two major advantages. First, it is *protocol-independent*. Because the method is applied to raw data, there is no need to know which protocol is used for the particular data transfer; the redundancies are identified across the whole protocol stack. Second, the mechanism tolerates *modifications* of the original objects. If some data object, such as file, is partially modified before it is transferred for the second time, those parts of it which remain unchanged will still benefit from the optimization. These two features made an approach extremely attractive for users, which, on one hand, have a low bandwidth network connection, and, on the other hand, transfer significant amount of redundant traffic. As the results obtained in [4, 5, 12] suggest, the average redundancy of a regular office traffic, consisting mostly of Web and E-mail data, can be from 20% to 60%, and it can get much higher in case of large data distributions or backups. These observations lead to emergence and success of commercial network appliances that offer data redundancy elimination among other services [1–3].

Consider the deployment for RE systems, shown in Figure 1. There is a central facility, a data center (DC), and a set of branch offices (BR), connected to it through WAN. It is assumed that WAN speed in this case is in the range of T1–T3, and the RE appliances are inserted between DC and BRs. Suppose that RE device can process data at 300 Mbps, and data is 99% redundant, so expected speed-up is 100×. For the T1 link it evaluates to 150 Mbps throughput, and this speed can be achieved by the appliance. For the T3 link 100× speed-up means 4500 Mbps, which cannot be achieved by the appliance, but still there is a beneficial speed-up of about 6×. However, lately even small office Internet access links are getting faster, becoming less of a bottleneck. Instead, RE overhead may become a bottleneck itself. In order to design an efficient RE system we must know what are the key factors affecting processing speed, and what are the key factors affecting compression. We must define how to evaluate RE system performance and how to compare RE systems. This paper attempts to give answers to these questions.

WOSP/SIPEW'10, January 28–30, 2010, San Jose, California, USA.

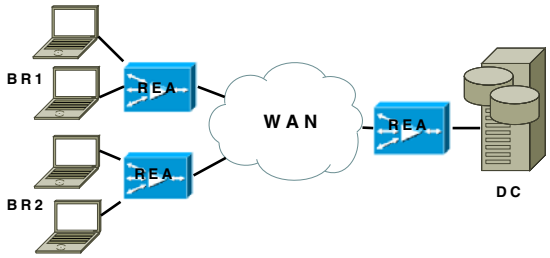

Figure 1: Typical RE system deployment

The rest of the paper is structured as follows. Section 2 discusses related work. Section 3 addresses the parameters and choices that contribute to the performance, and demonstrates system behavior on a set of benchmarks. Section 4 describes modifications to the basic scheme that may improve the performance. Section 5 describes a unified methodology for system evaluation. Section 6 briefly touches on further issues faced by the RE systems design.

2. RELATED WORK

Several projects and ideas related to RE emerged from the academic community. Most of them focus on integrating RE into a larger data transfer architecture, such as peer-to-peer networks [10] or CDN [9]. These are slightly different from the problem we are considering in that they introduce a form of protocol dependency. In [4] it is proposed to push RE capabilities down to Internet core. In [5] many valuable traffic redundancy measurements and observations are presented.

In contrast, our work is more focused on system implementation and performance aspects, rather than on RE algorithms or traffic properties.

3. ORIGINAL REDUNDANCY ELIMINATION

In this section we discuss the basic redundancy eliminating system. Due to space limitation, we omit some details, which can be found in [7]

3.1 Main processing stages

Fingerprinting data. The goal of this operation is to facilitate identification of repeating patterns in data stream. It is desired that minor modifications to the original data only cause minor changes to the result of splitting. Thus, for example, splitting data by simply dividing it into chunks of equal size and calculating their fingerprints does not work — any insertion or deletion in the original data will change all chunks and no redundancies will be found. It is more efficient to use pseudo-random mechanisms, which depend on the data itself, of which Rabin fingerprints is the common approach [11]. The algorithm calculates fingerprint over a sliding window of data. The byte at which special fingerprint is found becomes the last byte of a current data chunk. The result of a fingerprinting is a set of fingerprints and corresponding byte positions.

Indexing and lookup. The goal of a lookup is to use fingerprints from a previous step to search for repeating data in the local cache. If the data has been seen earlier and was saved to the cache, the cache must contain corresponding fingerprints. Finding them means that redundancy is detected. The lookup can be subdivided into two stages: fingerprint lookup and data lookup. To illustrate the difference, consider the lookup algorithm proposed in [12]. There, each fingerprint identifies a block of data of fixed size. However, the repeated data pattern might be larger, so the fingerprint is used only as a hint to point to the possible redundancy. To find the exact matched pattern the bytes of the received data are compared to the stored bytes, forward and backward from the fingerprinted block. Thus, the redundant data is represented as a fingerprint and byte offsets. In this example, the first stage of the lookup finds a fingerprint and its corresponding position in the cache, and the second stage reads the stored data and compares it to the received data to find the longest match.

Storing data. At this stage of processing the system has to decide which data to save in its cache. RE systems tend to increase compression ratio by increasing storage capacity, thus eventually the data has to be stored on hard drive. Assuming that the system is completely protocol unaware (i.e. cannot tell, in general, whether given piece of data is unique or highly repetitive), the decision can be based on whether data is new or it is already present in the cache. On one hand, saving all data to the cache consumes more storage space; on the other hand, it improves data access locality and may improve compression.

Reconstructing data. The task of data reconstruction is to restore the original data from the compressed one. While this stage poses several interesting problems, we will not be addressing them in this paper. The problems of the data restoration are mostly error detection and correction problems, which are corner cases, not directly affecting the performance. We are limiting the paper to the other stages, leaving data reconstruction details for the future work.

3.2 Choices and alternatives

All processing stage descriptions above allow for certain flexibility in the implementation. One example was already mentioned — when storing data there is a choice between caching all data, or caching new data only. Here we further focus on several key alternatives that can be considered by a system designer.

Expected inter-fingerprint distance. This is the parameter (further denoted *inter-FP distance* for brevity) of a chosen fingerprinting method, which determines how frequent, on average, the special fingerprints are. The trade-off here is: in general more frequent fingerprints require more work and more space for lookup structure, but may provide better compression on modified files. Basically, the distance defines granularity of detected redundancies; shorter distance will cause more fine-grain redundant patterns eliminated.

Caching all data vs. caching new data only. Storing all incoming data affects both performance and storage utilization. In this paper we are not considering storage utilization, but performance is important to us. There are two performance impacts: adding unnecessary writes causes latency increase at the moment of writing, but it also improves data locality and compression, thus may bring performance benefits in future.

TCP vs. IP layer. RE system may operate on an established TCP connection or on each IP packet independently. Operating on IP layer has several disadvantages: compression ratio becomes limited, as the length of the maximal match is bounded by the IP packet size; also, IP packets from concurrent connections are mingled in cache, which

results in poor data locality and will decrease compression even further. On the other hand, the IP based systems are faster as there is no TCP overhead.

These are the three basic factors that we are going to investigate. As each alternative presents certain advantages and disadvantages, our goal is to quantify them in order to make an optimal choice. We will start by observing the general behavior, exposed by these alternatives on several selected benchmarks.

3.3 Experimental system and benchmarks

To measure the impact of the factors above, we have created a simplified RE system, which replicates data encoding path discussed in Section 3. For simplicity we did not include a decoding path. As decoder's main task is to read indexed data, its performance depends more the storage I/O speed than on algorithm's parameters; besides, decoder does not affect the compression ratio in any way. Our implementation includes Rabin fingerprinting with a sliding window of 32 B, 2 GB hash table for fingerprint indexing, and a set of flat files for data storage. The system runs on a Linux box with `ext3` file system, Intel Xeon 2.33GHz CPU, and 4 GB RAM, which is very close to a mid-range commercial RE device hardware platform.

To evaluate the system behavior, we selected four benchmarks. *Benchmark-1*: randomly generated 1 GB file passed through the system twice. This benchmark is to get the compression upper bound and does not represent a common case. *Benchmark-2*: collection of 20 Linux kernel source trees (4 GB). This represents the transfers which gradually change. *Benchmark-3*: the contents of the popular web-site (15 GB), intended to simulate web browsing. *Benchmark-4*: 10 GB (5 hours) of the traffic captured on a WAN link between the regional offices of a large corporation.

Being deployed in a real network the system would operate by reading a unit of data (IP packet or TCP buffer), processing it, and sending it over the WAN. As such, read sizes randomly oscillate around some value, which depends on the network conditions and specific application. To emulate that, for each test we first select the basic buffer size from 512 B to 256 KB; the actual read size is obtained by adding a random variance of 5% to the basic size. Thus, on the benchmarks 1–3 we ran each test several times — one for each basic buffer size. On the Benchmark-4 sizes of the read data buffers are known from the trace, however, to compare TCP- and IP-layer processing we ran each test twice: first, accumulating packets into TCP buffers; second, processing packets one by one.

3.4 Observing basic trends

Figures 2, 3 and 4 demonstrate the overall trends observed on the benchmarks (results for Benchmark-2 are similar to Benchmark-3 and are omitted due to space limitation). In these experiments all data was written to the cache, no matter if it was already present there or not. Note that the graph for Benchmark-4 differs from the others, as the tests differ too. Also note that the results for benchmarks 1–3 are plotted on a logarithmic scale for better visibility.

On the graphs we can see the following tendencies. First of all, we see how the compression depends on the inter-FP distance, with a higher compression corresponding to a smaller distance, which is expected. What is worth noticing is that difference in compression due to inter-FP distance may be

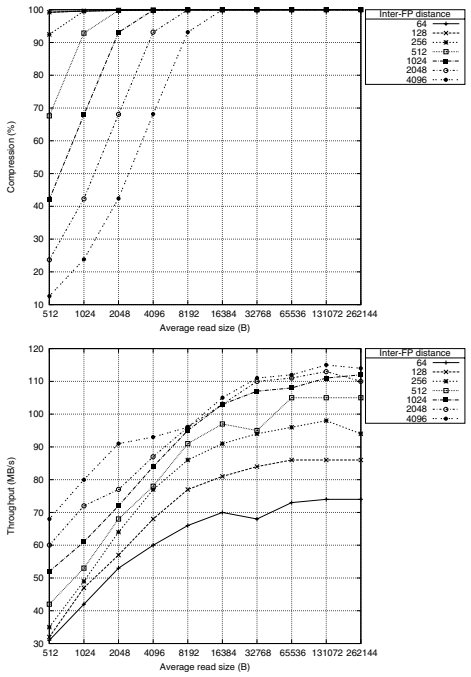

Figure 2: Benchmark-1: Basic algorithm, all data written

very significant (e.g. by factor of 10, as shown in Figure 2). The throughput also depends on inter-FP distance, but in the opposite way. The difference in throughput, however, is not as significant as the difference in compression. Another important observation (Figure 4) is that TCP-layer processing outperforms IP-layer both in compression (which is expected) and in throughput (which is less expected). The throughput difference is due to the constant processing overhead per each data unit. There is some unavoidable processing overhead per each piece of processed data, and, as there are more pieces processed in IP mode (IP packets) than in TCP mode (TCP buffers), the overhead has a noticeable impact. Remember, however, that we are discussing sequential processing. In case of parallel processing, IP mode throughput might get higher, as multiple IP packets of the same connection can be processed in parallel. However, it will also cause problems related to the packets being delivered out of order and creating inconsistencies between sender and receiver caches. Third observation, related to the second one, is that both compression and throughput tend to grow with the processed unit size. In general, a system cannot control read data sizes, because they depend on application behavior. Thus, while the RE systems favors larger amounts of read data, evaluation must take into account that these amounts might be small. So, unlike the TCP or IP mode, we do not consider a read size to be a part of the algorithm configuration.

On the throughput graph in Figure 2 it can be seen that the throughput for the higher inter-FP distances first drops, while read size grows, which seems to contradict with statements in a previous paragraph. The reason for that is that with large inter-FP distance and small read size the amount of detected redundancies is so small, that the system does

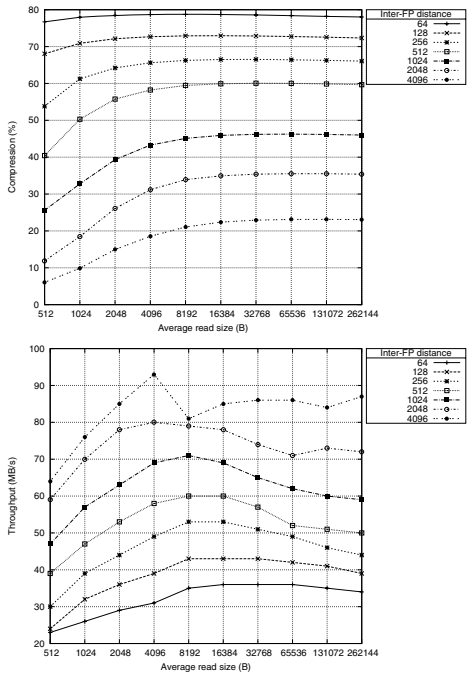

Figure 3: Benchmark-3: Basic algorithm, all data written

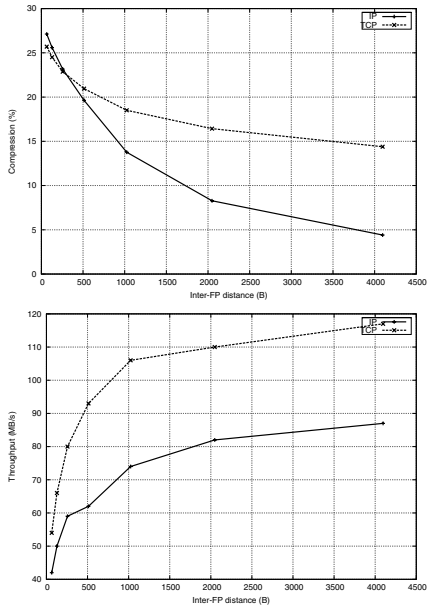

Figure 4: Benchmark-4: Basic algorithm, all data written

not perform any compression work — the data simply goes as is, with much higher throughput. As more redundancies are detected, the throughput goes down. This also explains the intersection of IP and TCP throughput graphs in Figure 4. These points illustrate that a high throughput is not always a good thing: there is no reason to deploy an RE system if it does not provide a good compression.

On Figure 5 we see what happens if we write only new data instead of writing all data. The key thing to notice here is that the impact on the compression is negligible. The impact on the throughput is more significant. The biggest difference is revealed when the compression drops. When the compression is high, it is the lookup that dominates in the processing, so the overhead from writing data is concealed. When the compression is low, the lookup contribution to the processing decreases, revealing the impact of the disk writes. Also, the special fingerprints are getting sparse as the inter-FP distance grows, which causes more writes to be skipped. However, when the inter-FP distance is small, the throughput is slightly higher when all data is written to the cache. This is where the data access locality comes into play. The results of this test for the other benchmarks are similar and are omitted to avoid redundancy.

4. MODIFIED REDUNDANCY ELIMINATION

We have profiled the system operations (fingerprinting, lookup and store), and discovered that they take approximately equal amounts of time (36%/32%/32%, respectively). The performance of the Rabin fingerprint algorithm is mostly determined by the memory I/O speed. Similarly, store speed is determined by the disk I/O. The lookup also involves disk read, and accesses each byte for comparison. This, however, is not mandatory. In this section we propose a modification of the basic scheme, which allows to speed-up the lookup component, by omitting disk reads and byte comparison. The idea is simple. As the fingerprinting algorithm moves the sliding window, we also maintain a Rabin fingerprint value for the whole block of data between the two special fingerprints. At the time when the next special fingerprint is found, we also have a unique identifier of the byte sequence encountered while searching for it. We use this identifier to search for redundancies. This does not require going to disk at all — a hit in the hash table means that the whole byte sequence is found. The approach has two major drawbacks. First, the granularity of the detected redundancies is strictly limited by the inter-FP distance. Second, calculating the unique identifier (essentially one more Rabin fingerprint) doubles the number of arithmetic operations per each byte of input data. Fortunately it does not cause extra memory access, as the same byte is used for both fingerprints. Thus, while the lookup performance may increase, it might be outweighed by the mentioned disadvantages.

In order to investigate benefits and losses of the proposed approach, we integrated the new lookup scheme into our system, and ran the same benchmark tests. The results are shown in Figures 6, 7, 8 and 9.

While the tendencies remained similar, the absolute values are quite different. Indeed, there is an improvement in the throughput (we observed 3× to 5× increase in the lookup speed), and there is a decrease in the compression (especially on the Benchmark-1, which determines the compression upper bound).

5. SYSTEM EVALUATION METHODOLOGY

The benchmark results help us to reveal the trade-offs in RE systems, but it is not sufficient for choosing the best system configuration. In this section we propose a methodology for RE system evaluation. We will combine experimental data into a single score value, which can be used for comparison.

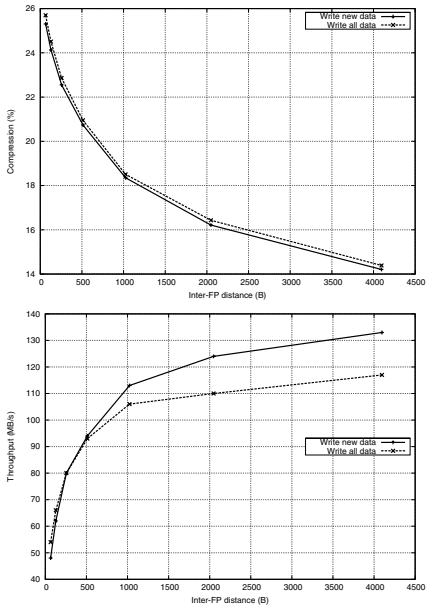

Figure 5: Benchmark-4: Basic algorithm, write comparison in TCP mode

We define a *benefit* function as: $\beta(w) = \frac{T(w)-w}{w}$. Here w is the WAN link bandwidth, and $T(w)$ is the *effective* throughput. If the compression ratio is r, and the maximal throughput that the system can sustain is T_{max}, then $T(w) = \min(T_{max}, \frac{w}{1-r})$. The benefit function determines a benefit for a given WAN link. As WAN links may vary across the system deployment field, we generalize our measure to a range, or a set of links. Let $\mathcal{W}$ be a space of possible WAN link capacities. Then we define a value of a given configuration for a given test data as $\int_{\mathcal{W}} \beta(w)dw$. This approximates how much benefit a system provides on a given set of WAN links. If there are multiple experiments, we assign a weight, α_i, to the i-th experiment, and calculate the final score of the algorithm, S, as follows:

$$S = \sum_i \alpha_i \int_{\mathcal{W}_i} \beta_i(w)dw \quad (1)$$

Let us demonstrate it on our tests. First, we assign weights to the benchmarks 1–3 as 2/49/49 (normalized to 100),because we assume that the first benchmark is rare. We chose minimal WAN speed, w_{min}, to be 256 Kbps, and set $\mathcal{W}_i = [w_{min}, T^i_{max}]$, where T^i_{max} is the maximal throughput, achieved for the i-th test. Then, $\beta_i(w) = \frac{\min(T^i_{max}, \frac{w}{1-r})-w}{w}$, which we substitute into Equation 1.

We use the same approach for Benchmark-4, except all tests there get the weight of 1. We have calculated the scores for our experiments, and, according to them, the best configuration is the one with 128 bytes inter-FP distance, TCP mode, modified algorithm, and only new data being written to cache. The result is consistent between benchmarks 1–3 and Benchmark-4. The scores for Benchmark-4 are listed in Table 1. The scores for 1–3 are similar and are omitted.

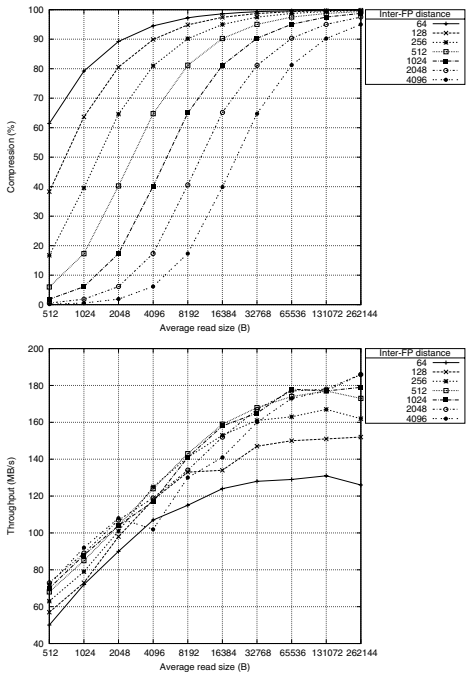

Figure 6: Benchmark-1: Modified algorithm, all data written

Algorithm	Write	Inter-FP distance	TCP or IP	Score
modified	new	128	TCP	23.51
modified	new	64	TCP	22.81
modified	all	128	TCP	22.27
modified	new	256	TCP	21.99
modified	all	64	TCP	21.36
modified	all	256	TCP	20.89
original	new	512	TCP	20.36
original	all	256	TCP	20.26
original	all	512	TCP	19.97
original	new	256	TCP	19.43

Table 1: Rankings and scores of algorithm configurations on Benchmark-4 (top 10)

6. CONCLUSIONS

In this paper we demonstrated the evaluation methodology for the network redundancy eliminating systems. We discussed the design alternatives and tendencies in the system behavior, based on our measurements. We proposed certain modifications that may improve system performance. We also proposed a criteria to compare RE algorithms and system configurations. To our knowledge this is the first attempt of such a criteria, and we believe it might be useful as a starting point.

We deliberately avoided several important aspects related to RE system performance. We have skipped the error detection and correction. We did not address concurrency and parallelism. We assumed no limitations on disk or memory, and did not account for their usage. If these factors are of the importance to the user, they must be included in the measure, and may change the scores. We are leaving these issues for future work.

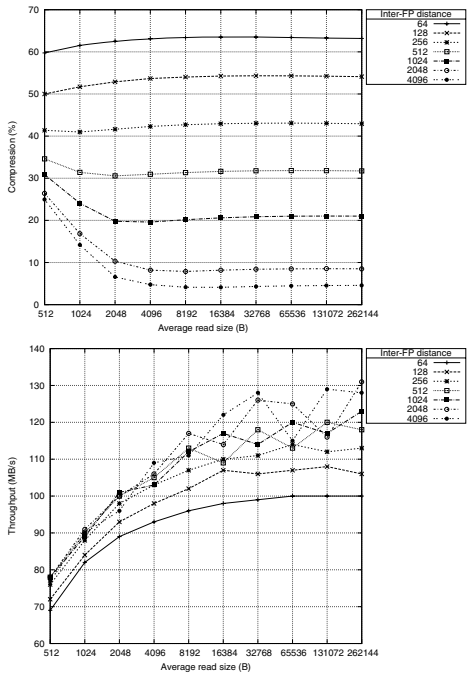

Figure 7: Benchmark-3: Modified algorithm, all data written

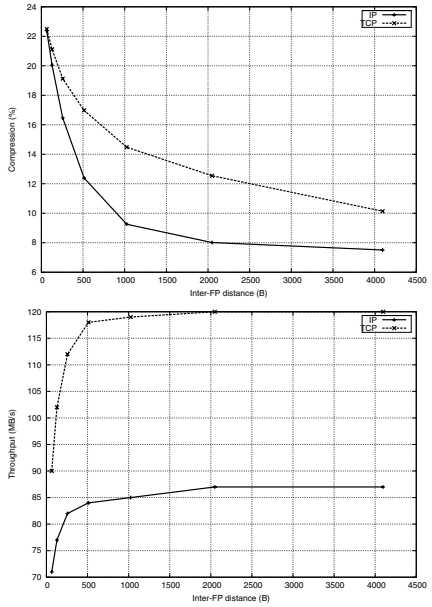

Figure 8: Benchmark-4: Modified algorithm, all data written

7. REFERENCES

[1] Cisco WAAS, `http://www.cisco.com`.

[2] Peribit WAN optimization, `http://www.juniper.net`.

[3] Riverbed networks, `http://www.riverbed.com`.

[4] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: The implications of universal redundant traffic elimination. In *Proc. of ACM SIGCOMM*, 2008.

[5] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramachandran. Redundancy in network traffic: Findings and implications. In *Proc. of ACM SIGMETRICS*, 2009.

[6] U. Manber. Finding similar files in a large file system. In *Proc. of USENIX ATC*, 1994.

[7] M. Martynov. Challenges for high-speed protocol-independent redundancy eliminating systems. In *Proc. of IEEE ICCCN*, 2009.

[8] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proc. of ACM SOSP*, 2001.

[9] K. Park, S. Ihm, M. Bowman, and V. S. Pai. Supporting practical content-addressable caching with CZIP compression. In *Proc. of USENIX ATC*, 2007.

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

[11] M. Rabin. Fingerprinting by random polynomials. Technical Report TR-CSE-03-01, Center for Research in Computing Technology, Harvard University, 1981.

[12] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. of ACM SIGCOMM*, 2000.

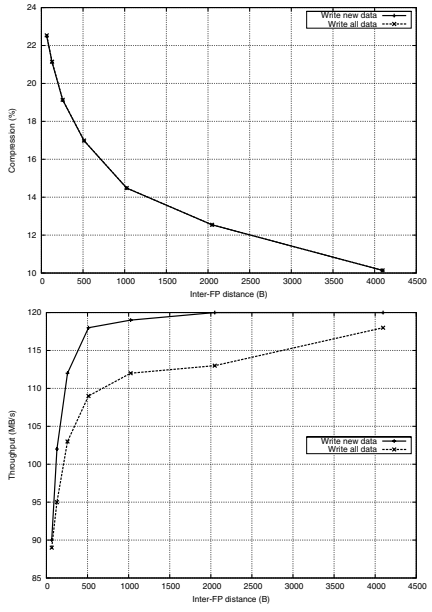

Figure 9: Benchmark-4: Modified algorithm, write comparison in TCP mode