CO-REDUCE: Collaborative Redundancy Reduction Service in Software-Defined Networks

Sejun Song[1] Daehee Kim[1] Hyungbae Park[1] Baek-Young Choi[1] Taesang Choi[2]
[1]University of Missouri-Kansas City, Missouri, USA
[2]Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea
{sjsong,daehee.kim,hpark,choiby}@umkc.edu, choits@etri.re.kr

ABSTRACT

A large portion of digital data is transferred repeatedly across networks and duplicated in storage systems, which costs excessive bandwidth, storage, energy, and operations. Thus, great effort has been made in both areas of networks and storage systems to lower the redundancies. However, due to the lack of the coordination capabilities, expensive procedures of C-H-I (Chunking, Hashing, and Indexing) are incurring recursively on the path of data processing. In this paper, we propose a collaborative redundancy reduction service (CO-REDUCE) in Software-Defined Networks (SDN). Taking advantage of SDN control, CO-REDUCE renders the promising vision of Redundancy Elimination as a network service (REaaS) as a real practical service. CO-REDUCE is a new virtualized network function service that dynamically offloads computational operations and memory management tasks of deduplication to the group of the software designed network middleboxes. Chaining various redundant REs of both storage and network into a service, CO-REDUCE consolidates and simplifies the expensive C-H-I processes. We develop service coordination protocols and virtualization and control mechanisms in SDN, and indexing algorithms for CO-REDUCE software-designed middleboxes (SDMB). Our evaluation results from the system and Mininet-based prototypes show that CO-REDUCE achieves 2-4 times more bandwidth reduction than existing RE technologies and has compatible storage space savings to existing storage de-duplication techniques while reducing expensive overhead of processing time and memory size.

CCS Concepts

•**Networks → Middle boxes / network appliances;**

Keywords

Data de-duplication, Network redundancy elimination, Software-Defined Networking

HotMiddlebox'15, August 17-21, 2015, London, United Kingdom
© 2015 ACM. ISBN 978-1-4503-3540-9/15/08...$15.00
DOI: http://dx.doi.org/10.1145/2785989.2786001

1. INTRODUCTION

The explosively growing digital data volumes and network traffic are greatly contributed by redundancies [1]. A large amount of redundant data is produced in various forms such as snapshots, disaster recovery mirrors, periodic backups, and compliance archives over the storage systems and repeatedly transferred to multiple users across network links. For instance, a study [12] shows there were 70% of redundancies in datasets collected from file systems of an enterprise.

The redundancies consume extra storage space, network bandwidth, and energy while incurring operational overhead of CPU and memory. Thus, as shown in Figure 1, many redundancy reduction technologies such as *Data deduplication (Dedup)*, *Network Redundancy Elimination (NRE)* [2, 3], WAN Optimization and Acceleration (WOA) [5, 18], Information-Centric Network (ICN) [6, 15], Content Delivery Network (CDN), and IP Multicast have been used in both areas of networks and storage systems. For identifying and removing the redundancies, they perform three common processes, which are Chunking, Hashing, and Indexing (C-H-I). As illustrated in Figure 2, *Chunking* splits a data into a sequence of data blocks through a customized chunking policy (fixed or variable size); *Hashing* creates a fixed size fingerprint for comparison and identification using a cryptographic hash algorithm, such as SHA-1 or MD5; and *Indexing* [16] is a memory I/O intensive process of identifying and retrieving duplicated chunks by comparing with the cached indexes. As the inline C-H-I process is very expensive – increasing proportionally to the data amount, it mostly has been deployed in the form of vendor specific special purpose middleboxes. Although they share the common motivations and processes of identifying and removing the redundant data or packets, due to the lack of coordination capabilities among them, they do not provide any benefits for each other and even introduce the overhead of recursive C-H-I procedures on the path of a data processing. For example, as illustrated in Figure 3, when data from a data center is transferred across network links to be stored in a storage server via enterprises, while each Dedup, WOA, and NRE can be performed for a benefit of its own domain, the data goes through multiple sequences of C-H-I processes, which incur significant computational and memory overhead.

In this paper, we propose a *co*llaborative *redu*ndancy reduction servi*ce* (CO-REDUCE) framework in Software-Defined Networks (SDN). CO-REDUCE renders the promising vision of Redundancy Elimination (RE) as a practical network service by generalizing the deployments of vendor specific middleboxes to that of software designed middle-

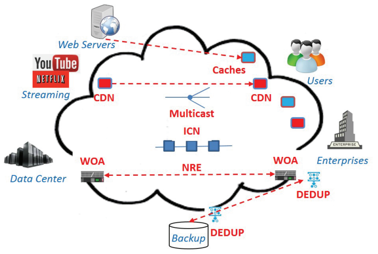


Figure 1: Redundancy reduction techniques

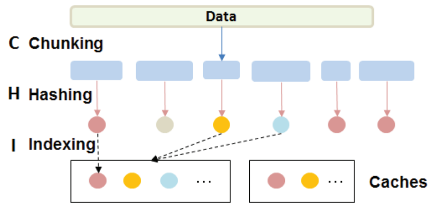


Figure 2: C-H-I process

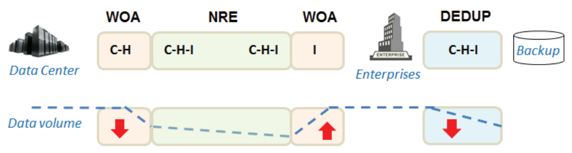


Figure 3: Redundant C-H-I process example

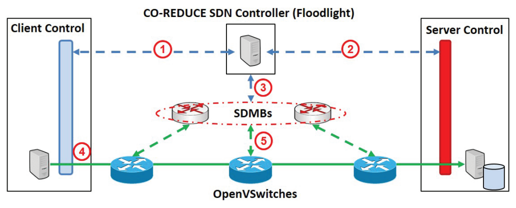


Figure 4: CO-REDUCE architecture

boxes that benefits to all of users, applications, and ISPs. Taking advantage of SDN control, CO-REDUCE provides a new virtualized network function service that dynamically offloads computational operations and memory management tasks to the group of on the software designed network middleboxes. It establishes coordination capabilities to simplify the expensive C-H-I processes into a service by chaining the redundancy reduction services on the path of a data processing. As shown in Figure 4, we design and implement prototypes of 1) service coordination protocols with control modules on both end-systems; 2) cache placement algorithms and cache application policies; 3) encoding and indexing algorithms in CO-REDUCE software-designed middleboxes (SDMB); 4) CO-REDUCE service marking mechanisms; and 5) data encoding and forwarding methods. Our evaluation results from the system prototypes as well as Mininet-based emulations show that CO-REDUCE achieves significantly higher bandwidth reduction than existing RE technologies and has equal/close storage space savings to existing storage de-duplication techniques by reducing expensive overhead of processing time and memory size. Furthermore, in the operations of recursive redundancy reduction, CO-REDUCE leads much less processing and memory overheads.

The rest of the paper is organized as follows. In Section 2, we review existing data reduction approaches. We describe the design and implementation issues of the CO-REDUCE system in Section 3. We evaluate our approach in Section 4, and offer a conclusion in Section 5.

2. RELATED WORK

A number of data reduction techniques have been proposed for both networks and storage domains. Storage domain data reduction techniques are to save storage space, and run at a client's or at a server's side. Client side techniques [7, 13] can reduce the network bandwidth by eliminating redundancy before the data transfer. However, the de-duplication ratio can be inefficient due to the limited data set and the processing cost can be too high for a client with limited capacity. Server side data de-duplication approaches have been used in traditional storage systems, and they mainly differ in the granularity of units for de-duplication, such as data chunks [11, 21], files [12], hybrid [9], and semantic granularity [10].

Network domain data reduction techniques are to save bandwidth and reduce latency by reducing repeating transfers through network links. End-to-end Redundancy Elimination (EndRE) and WAN optimizers [5, 18] remove redundant network traffic at two end points (e.g., branch to head quarter). Network Redundancy Elimination (NRE) techniques [2, 3] eliminate repeating network traffic across network elements such as routers and switches. NRE computes indexes [16] for the incoming packet payload and eliminates redundant packets by comparing indexes with the packets saved previously. The redundant payload is encoded by small sized shims and decoded before exiting networks. However, this approach suffers from high processing time due to sliding fingerprinting [20] at the routers and high memory overhead to save packets and indexes.

The ICN [6, 15] aims to reduce latency by caching data packets toward receiving clients. In addition, ICN uses name based forwarding table that causes extra table lookup time and raises scalability issues. Meanwhile, Content Delivery Networks (CDN) [4] can reduce redundant data traffic by preventing a long path to an origin server after locating files close to users, and IP Multicast [17] reduces traffic by allowing a sender to send multiple recipients a file with only a copy.

In summary, aforementioned redundancy reduction processes are very expensive, mostly performed by using vendor specific special purpose middleboxes. Furthermore, the costly processes are designed and performed independently, i.e., redundantly. As CO-REDUCE efficiently chains storage de-duplication and network redundancy elimination functions and virtualizes de-duplication processes, it achieves effective performance without introducing high processing and memory overhead.

3. CO-REDUCE DESIGN AND IMPLEMENTATION

In this section, we present the CO-REDUCE control and data flows to discuss how CO-REDUCE chains the entire service elements. We then describe encoding and indexing algorithms of SDMBs. CO-REDUCE's elastic control capabilities would provide better facilities for the interpath and

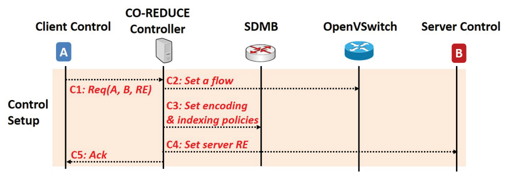
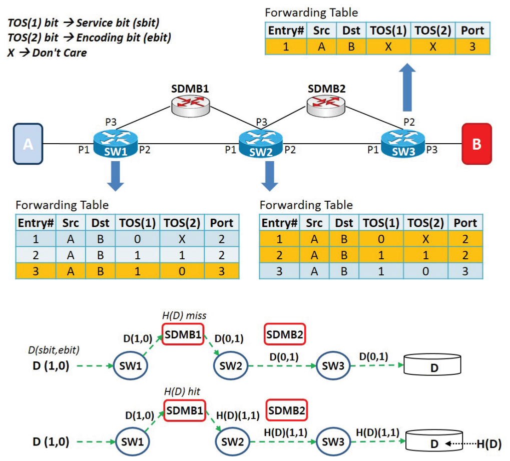


Figure 5: CO-REDUCE control and data flows



Forwarding Table

Entry#	Src	Dst	TOS(1)	TOS(2)	Port
1	A	B	X	X	3

Forwarding Table

Entry#	Src	Dst	TOS(1)	TOS(2)	Port
1	A	B	0	X	2
2	A	B	1	1	2
3	A	B	1	0	3

Forwarding Table

Entry#	Src	Dst	TOS(1)	TOS(2)	Port
1	A	B	0	X	2
2	A	B	1	1	2
3	A	B	1	0	3

Figure 6: CO-REDUCE forwarding operation example

intrapath caching, cache consistency, resource constraints, and traffic modification issues. However, due to the space limitation, in this paper, we focus on the fundamental design ideas and the operation processes. The details can be found in the extended version of the paper.

3.1 CO-REDUCE Control and Data Flows

The CO-REDUCE controller's main responsibilities include end-to-end element coordination, and cache/index placement and application policy management.

Control Setup: A CO-REDUCE process is initiated by a service request from one of the end systems. As illustrated in Figure 5, a client sends a service request with the source and destination information **(C1)** to the CO-REDUCE controller. Upon receiving the request, as presented in Algorithm 1 the CO-REDUCE controller calculates a route from the source to the destination, figures out the SDMB locations along the data path, and computes encoding and indexing policies for each SDMB. The CO-REDUCE controller then adds a new flow to the Openflow switches by sending flow setup messages **(C2)**. Once a forwarding table for a flow is established, the CO-REDUCE controller assigns encoding and indexing policies to each SDMB **(C3)**. A service coordination message is sent to the destination **(C4)**. Finally, the CO-REDUCE controller completes a client's service request by sending an acknowledgement to the client so that the client transfers data **(C5)**. This loosely coupled control setup renders the RE as a network service practice.

Forwarding Operation: CO-REDUCE supports a *zero chunking mechanism* that uses a network packet payload itself as a chunk. Hence, instead of performing an expensive chunking in the client, it will set a couple of TOS bits (a service bit and an encoding bit) to indicate if the packet is involved in the CO-REDUCE service. The *service bit* indicates if the packet needs a CO-REDUCE service in the network and the *encoding bit* represents if the packet is already encoded by the previous SDMB. Likewise, all the switches need to do is a regular forwarding. When an OpenVSwitch receives a packet, it simply forwards the packet according to its flow table. The forwarding decision is configured by the controller in the flow table associated with a specific flow with the TOS bits. For example, as shown in Figure 6, when a data packet *D (1,0)* arrives to a switch 1 from a port 1, the switch 1 will forward the packet to an SDMB 1 via a port 3 according to the forwarding entry 3. After the service and encoding bits are updated by the SDMB 1 according to the cache miss/hit to either *D (0,1)* or *D (1,1)*, the packet is forwarded through the switches 2 and 3. When the packet arrives in the destination server, the CO-REDUCE server finds the place to store the packet either into index list or storage according to the TOS bits.

Algorithm 1 CO-REDUCE Controller Algorithm

```
Input: inPacket(ServiceRequest)
Output: outPacket
 1: controlType = getType(payload)
 2: senderIP = getSenderIP(inPacket)
 3: // set up service
 4: srcIP = getSrcIP(inPacket)
 5: dstIP = getDstIP(inPacket)
 6: registerToService(srcIP, dstIP)
 7: (SDMBList, switchList) ← setupPath(srcIP, dstIP)
 8: computeHashRange(SDMBList)
 9: assignHashRange(SDMBList)
10: pushFlowEntry(switchList, SDMBList)
11: outPacket ← "confirm"
12: forward outPacket(senderIP)
```

CO-REDUCE Controller Implementation: We use Floodlight [8] to implement a CO-REDUCE controller. We implement a Floodlight module that computes hash ranges. A client module, SDMBs, and a server module communicate with a CO-REDUCE controller through REST API using cURL. We add REST API URIs into the Floodlight module for the communication. SDMBs use C++ JSON parser to parse JSON data (with hash ranges) that is delivered from a CO-REDUCE controller.

Hash Range	Path	R1	R2	R3	R4
CR-uniform	H1-H4	-	[0,0.33)	[0.33,0.66)	[0.66,1)
	H2-H4	[0,0.25)	[0.25,0.5)	[0.5,0.75)	[0.75,1)
	H3-H4	-	-	[0,0.5)	[0.5,1)
CR-MP	H1-H4	-	[0,0.5)	[0.5,1)	-
	H2-H4	-	[0,0.5)	[0.5,1)	-
	H3-H4	-	-	[0,1)	-

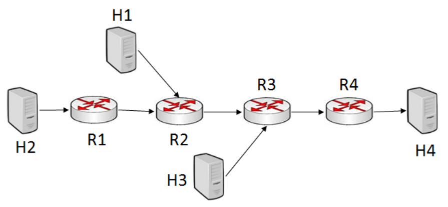


Figure 7: A network with three routes (H1-H4, H2-H4, and H3-H4) and four SDMBs (R1, R2, R3, and R4)

Algorithm 2 Packet Processing in SDMB

```
Input: inPacket
Output: outPacket
 1: // pathID is <srcIP>_<dstIP>
 2: pathID = getPathID(inPacket)
 3: payload = getPayload(inPacket)
 4: hashKey = computeHash(payload)
 5: hashRangeKey = computeHashRangeKey(hashKey)
 6: if hashRangeKey ∈ hashRange(pathID) then
 7:     if hashKey ∈ indexTable then
 8:         // redundant packet - encode
 9:         replacePayload(hashKey, inPacket, outPacket)
10:         recomputeChecksum(outPacket)
11:     else
12:         // unique packet
13:         saveToIndexTable(hashKey)
14:         outPacket ← inPacket
15:     end if
16: else
17:     outPacket ← inPacket
18: end if
19: forward(outPacket)
```

3.2 CO-REDUCE Software Designed Middle-Boxes (SDMB)

An SDMB performs encoding and indexing for an incoming packet. As presented in Algorithm 2, an SDMB computes a hash-range key of the packet upon receiving a packet. Although it is implementation specific, we perform a modulo operation with the 18 most significant bits from a SHA1 hash [14] to calculate a range of floating points from 0 to 1. If the calculated hash range is out of the SDMB's hash range, a packet is forwarded to the next hop without any processing (passing). Otherwise, it searches its index cache. If a match is found, it is considered a redundant packet. Thus, an SDMB replaces the original payload with the index (a small and fixed size) and sets an *encoding bit* to forward the next hop. If a match is not found, an SDMB adds the hash key in the table, resets a *service bit*, sets an *encoding bit*, and forwards the original payload to next hop. *It is clearly noticed that the packet encoding is done by only one of the SDMBs.* All others on the data path do a simple test if it needs to encode. Furthermore, once an encoding is done, the packet won't be forwarded to SDMB anymore.

Index/Cache Placement Algorithms: CO-REDUCE supports a distributed indexing mechanism. Using a hash-based sampling [19], a CO-REDUCE controller computes the disjoint *hash ranges* over the route of a flow and assigns them to SDMBs. We have implemented and tested three different index distribution algorithms.

- **CR-full** is an approach that an SDMB holds a full hash ranges [0,1). This will incur more memory overhead, but it can achieve a better redundancy reduction ratio. The index size complexity per route is *O(n*m)* where *n* and *m* are the number of unique packets and the number of SDMBs on a path, respectively.
- **CR-uniform** assigns the hash-ranges uniformly to the SDMBs on a data path. As presented in Algorithm 3, CR-uniform hash ranges are computed using the number of SDMBs in each flow path. In Figure 7, a path *H1-H4* has three SDMBs. Thus, each SDMB has an encoding responsibility of $\frac{1}{3}$ or 0.33. Hash ranges are computed from the first SDMB (*R2*) to the last SDMB (*R4*) on a path, starting from 0 by accumulating the hash ranges: *R2* is assigned [0, 0.33), where 0 is inclusive and 0.33 is exclusive. The complexity of the index size per path is *O(n)*, where *n* is the number of unique packets on a data path.
- **CR-MP** assigns the disjoint hash ranges only for the SDMBs that have more than one incoming flow of the same destination (merge point). The approach is very similar to the reverse multicast or multicast join protocols. As shown in Algorithm 3, the hash range is computed by $\frac{incoming\ degree\ of\ a\ merge\ SDMB}{total\ incoming\ degree\ of\ merge\ SDMBs\ on\ a\ path}$. For example, in Figure 7, a path H1-H4 has two merge SDMBs such as *R2* and *R3*. The hash ranges are assigned only to *R2* and *R3* as [0, 0.5) and [0.5, 1), respectively.

Algorithm 3 Computing Hash Range

```
For CR-uniform & CR-MP
Input: SDMBList, pathList
Output: nodes with hash range
 1: for all path ∈ pathList do                ▷ retrieve each path
 2:     if approach == "CR-uniform" then
 3:         fraction = 1 / numSDMBs(path)
 4:     else if approach == "CR-MP" then
 5:         totalDegree = getTotalDegree(path)
 6:     end if
 7:     SDMBs = getSDMBs(path)
 8:     range = 0
 9:     for all SDMB ∈ SDMBs do ▷ an SDMB in a path
10:         if approach == "CR-MP" then
11:             fraction = SDMB.getDegree() / totalDegree
12:         end if
13:         SDMB.lowerBound = range
14:         SDMB.upperBound = range + fraction
15:         range = range + fraction
16:     end for
17: end for
```

SDMB Implementation: An SDMB is implemented as a userspace program that is a callback function based on a libnetfilter_queue userspace library. An SDMB runs on a Linux bridge that connects the incoming and outgoing network interfaces. To intercept an incoming packet, we set up the iptables rules in a filter table. The iptables rules are established for coordinating an OUTPUT for a client module, a FORWARD for an SDMB, an INPUT for a server module along with iptables-extension NFQUEUE. Whenever packets come in, packets are punted to a userspace program through a netfilter queue. A userspace program handles an incoming packet and a processed packet is forwarded back to network elements such as switches and SDMBs or a server.

4. EVALUATIONS

We compare the performance of CO-REDUCE with other existing storage and network data reduction techniques in the aspects of the storage and bandwidth savings, processing time, and memory usage.

4.1 Setting

We use a real testbed system as well as a mininet-based emulation. The testbed experiment is to verify CO-REDUCE

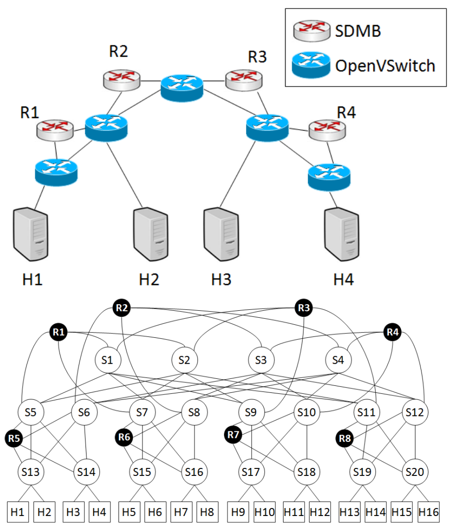
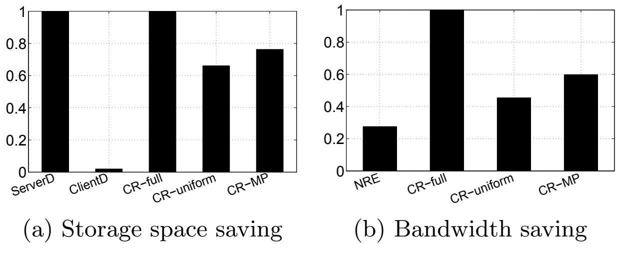


Figure 8: Experiment and fat-tree emulation topologies



(a) Storage space saving (b) Bandwidth saving

Figure 9: Comparison of performance

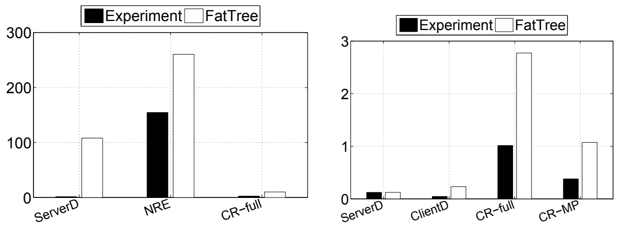
algorithms within a practical system environment as shown in Figure 8. We also set up a large fat-tree topology for the mininet-based emulation as illustrated in Figure 8 to validate CO-REDUCE in the typical data center network topologies. We place a server in the right most host: *H4* in the experiment and *H16* in the fat-tree. Other hosts act as clients. $R\{x\}$ represents an SDMB and $S\{x\}$ is an OpenVSwitch. The path and number of nodes selected for the fat-tree are the same as using a spanning tree, so two topologies are considered as the same case for our evaluation.

In both system experiments and mininet-based emulations, we use a dataset of campus log data that has been captured from the university data center. All the clients send the same dataset as other clients, so redundancy is $\frac{14}{15}$ for fat-tree.

We use two metrics for measuring performance: storage space and bandwidth savings, and another two metrics for measuring overhead: processing time and memory size. To present storage space saving, we used a de-duplication ratio. The de-duplication ratio is a typical means to show how much storage space is reduced and is computed by $\frac{volume\ of\ redundant\ data\ eliminated}{total\ volume\ sent\ by\ all\ clients} * 100$. Bandwidth saving is computed by $\frac{Reduced\ traffic\ size}{Total\ traffic\ size\ without\ redundancy\ elimination} *$ 100. For overhead metrics, we measure processing time and memory size occurred at clients, a server, and SDMBs. We compare CO-REDUCE with client Dedup, server Dedup, and network wide RE (NRE) [3].

4.2 Storage and Bandwidth Savings

We first present experimental results of performance improvement in CO-REDUCE compared to the existing techniques. For this purpose, we compare relative value normalized for all metrics as in Figure 9. For storage space saving as shown in Figure 9(a), CO-REDUCE shows the closest performance to server Dedup (ServerD) that is the best for saving storage space in existing techniques. CR-full shows exactly the same space saving as ServerD. This indicates CR-full does not miss any redundancy through the network. CR-uniform, and CR-MP have close space saving to ServerD. Meanwhile, ClientD is the worst for space saving (1.6%). For bandwidth saving in Figure 9(b), CO-REDUCE schemes show more bandwidth saving by 2-4x than NRE. CR-full shows the best bandwidth saving followed by CR-MP and CR-uniform.



(a) Processing time (Sec) (b) Memory size (MB)

Figure 10: Overhead per topology

4.3 Operational Overhead of Memory and Processing Time

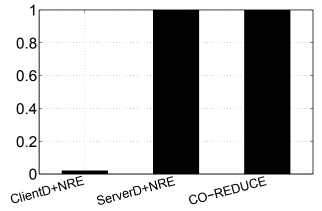
We now compare the overhead for the different network topologies as shown in Figure 8. We mainly consider the differences in the number of clients and the location of clients. The processing time, as shown in Figure 10(a), increases proportional to the number of clients for all techniques. Processing time increases more slowly in CR-full than others. Other CO-REDUCE approaches such as CR-uniform and CR-MP have the almost same processing time as CR-full. Client Dedup and variable-size server Dedup are not shown for readability due to their excessive processing times.

We find that computers used for SDMBs in experiment are much slower than a computer used for emulation (multi-tree/fat-tree), which amplifies the processing time slowed by fingerprinting. The memory size increases proportional to an increase in the number of clients as shown in Figure 10(b). CR-full has more index size than ServerD and ClientD, but CO-REDUCE can reduce the number of the indexes by an indexing scheme like CR-MP reduces the memory size of CR-full in the figure.

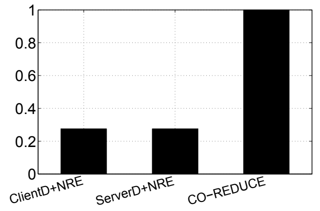
4.4 Performance and Overhead: CO-REDUCE vs. Combined Reduction Techniques

When data is transferred to a server across network links, each Dedup and NRE can be performed for a benefit of their own domains. The data may go through various forms of de-duplication processes redundantly. We compare CO-REDUCE with a couple of redundant de-duplication scenarios such as the ClientD and NRE (denoted as ClientD+NRE) and the ServerD and NRE (denoted as ServerD+NRE).

As shown in Figure 11(a), CO-REDUCE shows the best space savings as good as the ServerD+NRE scenarios. For space saving, both combined approaches rely on storage services including client Dedup and server Dedup because NRE is not applicable for storage space saving. As shown in Figure 11(b), CO-REDUCE saves the most bandwidth compared to two combined approaches. For two combined approaches, bandwidth saving is determined by performance of NRE. For processing time, CO-REDUCE outperforms two

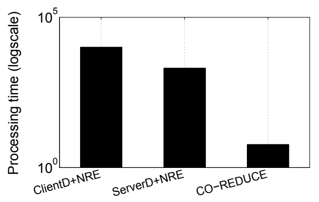


(a) Storage space saving

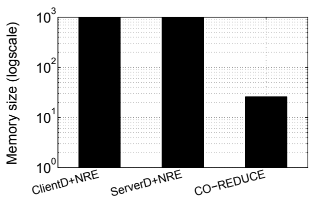


(b) Bandwidth saving

Figure 11: Performance of combined approaches



(a) Processing time (Sec)



(b) Memory size (MB)

Figure 12: Overhead of combined approaches

combined approaches as shown in Figure 12(a). As shown in Figure 12(b), CO-REDUCE requires less memory size than two combined approaches. The gap between CO-REDUCE and combined approaches is mainly caused by NRE that stores packets itself as well as indexes. Though ClientD has slightly more reduction in memory size than ServerD, the slight reduction becomes invalid due to excessive memory size by NRE. Overall, the evaluation results show that in scenarios of both end-systems and networks performing de-duplication redundantly, CO-REDUCE achieves very efficient processing and memory overhead.

5. CONCLUSION

We proposed CO-REDUCE, an efficient software-designed de-duplication as a network and storage service where servers, clients and middleboxes collaboratively operates to provide effective storage space and network bandwidth savings while significantly reducing processing time and memory size. A client only tags a packet on TOS bits for service indication and does no extra chunking, since a packet is the unit of deduplication. All the servers need to do is to check TOS bit and to store the data into its index list or storage block. There is no extra overhead in switches. Only one of the SDMBs on the path will do encoding of a packet - indexing and replace the payload. Other SDMBs on the data path will do a simple test if it needs to encode. If not just pass the packet. Furthermore, once an encoding bit is set, the packet won't be forwarded to SDMB at all. We developed efficient encoding and indexing algorithms for a CO-REDUCE software-designed middlebox (SDMB) and effective control mechanism for an SDN controller. We also built a prototype of testbed experiments and conducted Mininet-based emulations to evaluate CO-REDUCE on real system environments and typical DCN topology, respectively. Our evaluation results show that CO-REDUCE saves 2-4 times more bandwidth than the state-of-art NRE technique and same/close storage space saving to the Dedup technique with low overhead by achieving very efficient processing and memory overhead.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (14.000.05.001, Smart Networking Core Technology Development).

6. REFERENCES

[1] The digital universe in 2020. http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf.

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

[3] A. Anand, V. Sekar, and A. Akella. SmartRE: an architecture for coordinated network-wide redundancy elimination. In *SIGCOMM*, 2009.

[4] J. Apostolopoulos, T. Wong, W. tian Tan, and S. Wee. On multiple description streaming with content delivery networks. In *INFOCOM*, 2002.

[5] Citrix. Cloudbridge. http://www.citrix.com/products/cloudbridge/overview.html.

[6] A. Detti, N. Blefari Melazzi, S. Salsano, and M. Pomposini. CONET: A Content Centric Inter-networking Architecture. In *ICN*, 2011.

[7] EMC. Avamar. http://www.emc.com/backup-and-recovery/avamar/avamar.htm.

[8] Floodlight. http://www.projectfloodlight.org/.

[9] D. Kim and B.-Y. Choi. HEDS: Hybrid Deduplication Approach for Email Servers. In *ICUFN*, 2012.

[10] D. Kim, S. Song, and B.-Y. Choi. SAFE: Structure-Aware File and Email Deduplication for Cloud-based Storage Systems. In *CloudNet*, 2013.

[11] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality. In *FAST*, 2009.

[12] D. T. Meyer and W. J. Bolosky. A Study of Practical Deduplication. In *FAST*, 2011.

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

[14] N. I. of Standards and Technology. Secure Hash Standard. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[15] D. Perino, M. Varvello, and K. P. Puttaswamy. ICN-RE: redundancy elimination for information-centric networking. In *ICN*, 2012.

[16] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard Univ., 1981.

[17] P. Rajvaidya and K. Almeroth. Analysis of routing characteristics in the multicast infrastructure. In *INFOCOM*, 2003.

[18] Riverbed. SteelHead for WAN Optimization. http://www.riverbed.com/products/wan-optimization/.

[19] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen. CSAMP: a system for network-wide flow monitoring. In *NSDI*, 2008.

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

[21] B. Zhu, K. Li, and H. Patterson. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. In *FAST*, 2008.