SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination

Ashok Anand*, Vyas Sekar† and Aditya Akella*
*University of Wisconsin-Madison, †Carnegie Mellon University
{ashok,akella}@cs.wisc.edu, vyass@cs.cmu.edu

ABSTRACT

Application-independent Redundancy Elimination (RE), or identifying and removing repeated content from network transfers, has been used with great success for improving network performance on enterprise access links. Recently, there is growing interest for supporting RE as a network-wide service. Such a network-wide RE service benefits ISPs by reducing link loads and increasing the effective network capacity to better accommodate the increasing number of bandwidth-intensive applications. Further, a network-wide RE service democratizes the benefits of RE to all end-to-end traffic and improves application performance by increasing throughput and reducing latencies.

While the vision of a network-wide RE service is appealing, realizing it in practice is challenging. In particular, extending single-vantage-point RE solutions designed for enterprise access links to the network-wide case is inefficient and/or requires modifying routing policies. We present SmartRE, a practical and efficient architecture for network-wide RE. We show that SmartRE can enable more effective utilization of the available resources at network devices, and thus can magnify the overall benefits of network-wide RE. We prototype our algorithms using Click and test our framework extensively using several real and synthetic traces.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*

General Terms

Algorithms, Design, Management

Keywords

Redundancy Elimination, Caching

1. INTRODUCTION

Redundancy Elimination (RE) for network transfers has gained a lot of traction in recent years. RE is widely used by data centers and enterprise networks to improve their effective network capacity, to reduce their wide-area footprint, and to improve end-to-end application performance. The importance of RE is reflected in the emergence of a huge market for RE solutions (e.g., [4, 3, 2, 8, 5]) and their rapidly growing adoption [6, 9].

The success of such deployments has motivated researchers, equipment vendors, and ISPs to explore the potential of network-wide RE. For example, Anand et al. [12] have recently shown the benefits of supporting RE as a primitive IP-layer service on network routers. In similar vein, network equipment vendors have highlighted network-wide support for content caching and duplicate suppression as a key focus area in their future development efforts [3, 2]. Broadly speaking, these efforts argue for deploying RE at multiple points across a large network and using it as a generic service which is transparent to end-to-end applications.

This vision of network-wide RE is promising for two reasons. First, a network-wide deployment spreads the benefits of RE to all end-to-end applications, as opposed to just benefiting transfers on the individual links of enterprises. Second, it benefits ISPs by improving their effective network capacity and allowing them to better accommodate the increasing number of bandwidth intensive multimedia and file-sharing applications we see today, and by giving them better control over traffic engineering operations [12].

While RE has been well-studied in the context of point deployments (e.g., enterprise WAN access links), there has been little work on how best to design network-wide RE. Thus, the promise of network-wide RE remains unfulfilled. In this paper, we study how to build an effective and practical network-wide RE architecture.

We start by observing that a network-wide RE architecture should meet three key requirements:

(1) **Resource-awareness:** RE involves resource-intensive operations such as indexing content, looking up content fingerprints and compressing data, and reconstructing the original content from locally stored information. An ideal approach must explicitly account for the resource constraints on network elements in performing these RE functions. These constraints arise mainly from (a) throughput bounds which depend on the number of memory operations possible per second and (b) memory capacity which limits the amount of data that can be cached for RE purposes. Naive approaches that do not account for these constraints, such as the strawman framework of Anand et al. [12], offer sub-optimal performance. In contrast, using the limited resources available at each node intelligently can offer close to the best possible benefits.

(2) **Network-wide goals:** The architecture should allow network operators to specify network-wide goals such as increasing overall efficiency (e.g., improving the network throughput) or to achieve specific traffic engineering goals (e.g., alleviating congested hotspots).

(3) **Flexibility:** The architecture must be incrementally adoptable providing benefits even under partial deployment, and must supplement, not replace, current network operations such as existing routing and network management practices.

SIGCOMM'09, August 17–21, 2009, Barcelona, Spain.

We present the design, implementation, and evaluation of SmartRE, an architecture for network-wide RE that meets the above requirements. In SmartRE, redundancy elimination is performed in a coordinated fashion by multiple devices. SmartRE uses the available resources on RE devices efficiently and naturally accommodates several network-wide objectives.

In describing SmartRE, we focus largely on packet-level RE in ISP networks [12], where RE devices on routers cache packet payloads and strip duplicate strings from individual packets. However, we believe that our design can apply to other deployment scenarios, e.g., in multi-hop wireless networks and datacenters.

In SmartRE, a packet can potentially be reconstructed or decoded several hops downstream from the location where it was compressed or encoded. In this respect, SmartRE represents a significant departure from packet-level RE designs proposed in prior solutions [29, 12], where each compressed packet is reconstructed at the immediate downstream router. Further, SmartRE uses a network-wide coordinated approach for intelligently allocating encoding and decoding responsibilities across network elements.

In general, encoding incurs greater overhead than decoding. Thus, SmartRE allocates encoding to ingress routers to avoid overloading interior routers that operate at higher line-rates and thus have stricter resource constraints. Since the number of edge routers is large, a large number of encoded packets are introduced into the network. Interior routers in SmartRE perform less expensive decoding actions. Decoding is performed in a coordinated fashion with each interior router responsible for storing and reconstructing a fraction of the encoded packets on a path. We use hash-based sampling techniques [31] to facilitate coordination across interior routers while incurring negligible overhead.

When allocating encoding and decoding responsibilities across a network, SmartRE takes into account the memory capacity and packet processing throughput at each RE device along with the prevailing traffic conditions, and configures the actions of different devices so as to best meet an operator-specified network-wide goal. This ensures that no device is overwhelmed and that RE is used optimally to meet the network's objectives.

The duplicate removal and reconstruction logic in SmartRE can be implemented in high-speed two-port switches or middleboxes, which can then be deployed across specific ISP links. These enable incremental adoption in an ISP network. We develop prototypes of the two-port switches in the Click modular router [21]. Using real packet traces, we find that the prototypes can perform duplicate removal at 2.2 Gbps and reconstruction at 8 Gbps.

We conduct an in-depth evaluation of SmartRE as applied to IP-layer RE in ISP networks using controlled simulations based on synthetic and real packet traces over several real and inferred ISP topologies. Across a range of topologies and traffic patterns, the performance of SmartRE is 4-5$\times$ better than naively extending a single-vantage point RE solution to the network-wide case. Further, and more significantly, SmartRE achieves 80-90% of the absolute network footprint reduction of the optimal possible case where RE devices are not limited by any throughput or capacity constraints. We also evaluate partial deployment scenarios and find that enabling SmartRE on a small set of strategically selected routers can offer significant network-wide benefits.

2. RELATED WORK AND BACKGROUND

We start by describing prior work on removing duplicate data from network links, ranging from full object-based approaches to partial packet-based ones. We then present details of packet-level RE and describe prior work on enabling packet-level RE as a router service across ISP networks that forms a key focus in our work.

2.1 Related Work

Object-level caching: Several systems in the past have explored how to remove duplicate data from network links. "Classical" approaches such as Web caches work at the object level, serving popular HTTP objects locally [32]. In similar spirit, CDNs and peer-to-peer caches [7, 1] perform object-level duplicate removal.

Protocol-independent RE mechanisms: In recent years, a class of *application- and protocol-independent* techniques have been developed which can remove redundant strings from any traffic flow. Starting with the pioneering work of Spring et al. [29], several commercial vendors have introduced "WAN optimizers" which remove duplicate content from network transfers. Many of these products [4, 2, 8, 5] work at the level of chunks inside objects and we refer to them as *chunk-level* approaches. In contrast, both Spring et al. [29] and Anand et al. [12] adopt techniques which are similar at the high level but operate at a *packet-level*.

Content-based naming for RE: Content-based naming has emerged as an alternative to enhance web caching (e.g., [19, 26]), content distribution (e.g., [30, 23, 22]), and distributed file systems (e.g., [11]). These approaches use fingerprinting mechanisms [24] similar to packet-level RE to identify addressable chunks. However, these approaches require modifications to end-systems to fully realize the benefits of RE. Network-based, protocol-independent RE approaches are transparent to end-systems and offers the benefits of RE to end-systems that are not content-aware.

2.2 Packet-level RE Explained

The central idea of packet-level RE is to remove strings in packets that have appeared in earlier packets. To perform RE across a single link, the upstream device stores (in memory) packets it has transferred on the link over a certain period of time. Packet contents are indexed using *fingerprints* which essentially form content-based hooks pointing to content in random locations within the packet. For each incoming packet, the upstream RE device checks if the packet's fingerprints have appeared in earlier in-memory packets. Each matching fingerprint indicates a certain region of partial overlap between the incoming packet and some earlier packet. The matching packets are compared to identify the maximal region of overlap. Such overlapping regions are removed from the incoming packet and a shim is inserted to tell the downstream device how to decode the packet using its local memory. A packet can carry multiple shims, each potentially matching a different in-memory packet. Decoding is simple: the downstream device uses the shim in the encoded packet to retrieve the matching packet(s), and fills in the corresponding missing byte range(s). Chunk-level approaches work similarly.

2.3 Network-wide RE

Why packet-level RE: Both packet- and chunk-level RE are agnostic to application protocols and can be implemented as generic network services that need not understand the semantics of specific applications. Prior studies have shown that both approaches are significantly better than caching entire objects [29]. However, chunk-level approaches require terminating TCP connections and partially reconstructing objects before applying compression. This interferes with the end-to-end semantics of connections and also imposes high overhead on the RE devices since they must maintain per-flow state. Packet-level approaches do not interfere with end-to-end semantics of connections, and where technology permits, can be transparently supported in routers or middleboxes.

Extending packet-level RE to a network: Since packet-level RE brings significant compression benefits while operating in a transparent and application-agnostic fashion, Anand et al advocate its

use as a router primitive for network-wide RE [12]. In their proposal, each router in an ISP network maintains a cache of recently forwarded packets. Upstream routers on a link use the cache to identify common content with new incoming packets and strip these redundant bytes on the fly. Downstream routers reconstruct packets from their local cache. This process repeats in a *hop-by-hop* fashion along a network path inside an ISP. Anand et al. evaluate an ideal, unconstrained setting where they assume memory operations take negligible time and that the caches on each router are infinite. Under this model, they show that network-wide RE could offer significant benefits in terms of reducing overall network load and absorbing sudden traffic overload in situations such as flash crowds. The central goal of our paper is to design a practical architecture that can achieve these benefits when RE elements operate within realistic throughput and memory capacity constraints.

The hop-by-hop approach proposed by Anand et al. is a naive approach because it takes a very link-local view of RE and does not account for constraints of the RE devices. In the next section, we discuss why this naive approach offers poor performance in practice and show how smarter caching and coordination can offer vastly improved benefits.

3. BENEFITS OF COORDINATION

We start by describing the practical limits on the throughput of the two packet-level RE primitives, namely, encoding and decoding. Then, we present qualitative examples highlighting the benefits arising from assigning encoding and decoding responsibilities across a collection of routers in an intelligent, coordinated fashion. In particular, we show how this: (1) leads to efficient memory usage, (2) ensures RE-related tasks can be performed at full capacity, and (3) enables incremental deployment. We contrast this against a naive approach that does not account for resource constraints.

In this section, we assume a hypothetical intelligent, coordinated approach. This has two implications. First, we have the flexibility to specify where a packet should be cached along a routing path. In particular, this allows us to split caching responsibilities along a path. This is in contrast to the hop-by-hop approach, where each packet is explicitly cached at every hop along the path. For example, if packets $p_1, \ldots, p_4$ traverse a path $I, R_1, \ldots, R_4$, we can specify that each p_i is cached at (and only at) R_i. Second, we assume that RE devices that are separated by multiple hops in the network can either implicitly or explicitly maintain a consistent view of each other's caches. This means that an encoded packet can potentially be decoded several hops downstream from the point where it was encoded. In the above example, this means that I can encode packet p_4 with respect to p_3 and R_3 is responsible for decoding it. Again, in the hop-by-hop approach, this would not be possible; each packet would have to be encoded and decoded per-link.

3.1 Encoding and Decoding Throughput

Standalone throughput: The main bottleneck affecting the processing throughput of packet-level RE operations is *memory access*. Encoding a packet requires multiple memory accesses and is much slower than decoding. To see why, suppose that the memory hardware can support R random memory accesses per second. For modern DRAMs, the random access latency is 50ns, hence $R = 2 \times 10^7$. Suppose that each packet has at most k matches, and that we compute F fingerprints for each packet. (Note that since the number of matches can never be more than the number of fingerprints that were computed, $k \leq F$.) Typical values are $F = 10$ and $k = 3$ [12].

The encoding throughput for a standalone RE device is *at most* R/F packets per second. This is because each packet, whether it

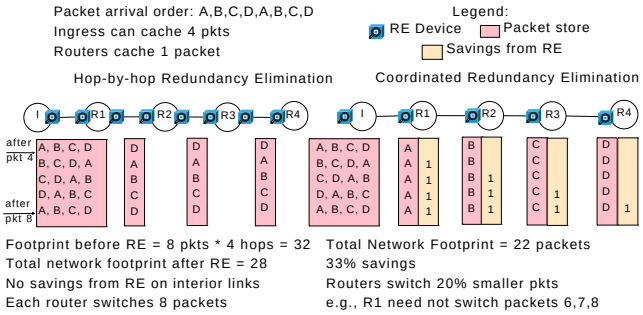


Figure 1: Benefits of a coordinated approach when RE devices have constraints on memory size.

can be encoded or not, requires F random accesses to determine if there is a match or not. Once matches are found, further processing is required to actually create the encodings. On the other hand, decoding throughput is *at least* R/k. This is because each packet has between 0 and k encodings. Thus, in this standalone case, decoding is $\geq F/k$ times faster than encoding. Since $k \leq F$, the decoding throughput is clearly higher.

Throughput on a single link: Given this understanding of the standalone encoding and decoding throughput, we can now consider the throughput across a single link. For simplicity, let us assume all packets are of the same size MSS. Suppose that the link capacity is such that it can carry P MSS-sized packets per second. For instance, if the link speed is 2.4Gbps (OC48), and $MSS = 500B$, then $P = 6 \times 10^5$ and for an OC192 link $P = 2.4 \times 10^6$. Two cases arise:

1. **Slow link** ($R/F \geq P$): This means that *line rate* encoding and decoding are possible; e.g., for an OC48 link where $R/F = 2 \times 10^6 \geq P = 6 \times 10^5$. In this case, the encoder can encode up to P packets per second, each carrying up to k matches. The decoder can decode each encoded packet.
2. **Fast link** ($R/F < P$): This means that *line rate encoding* is not possible. This is the case for OC192 and higher speed links. ($R/F = 2 \times 10^6 < P = 2.4 \times 10^6$). In this case, the encoder can encode no more than R/F packets per second; a fraction of packets are left un-encoded to ensure line-rate operation. Even though the decoder as a standalone operates F/k times faster, its decoding throughput is now limited by the encoding throughput immediately upstream. Thus, it is limited to decoding R/F packets per second.

3.2 Motivating Examples

We present the examples in the context of a "bump-in-the-wire" deployment where an RE middlebox is attached to router linecards. Each RE device has pre-specified *resource constraints*. These capture hardware limitations (e.g., how many decoding actions can the device perform per unit time?) or economic constraints (e.g., DRAM cost which could limit total memory per device).

These examples also apply when there are resource budgets *per router*. For example, processing constraints induced by power/cooling requirements are better modeled on a per-router/per-PoP basis rather than per-middlebox. Also, software or virtualized RE deployments (e.g., [14, 21]) would be characterized by per-router constraints.

As the following examples show, the naive hop-by-hop approach described in the previous section severely constrains the effectiveness of redundancy elimination.

Memory efficiency and router benefits: Consider the scenario in Figure 1. Suppose each RE device on the path has memory to store only 1 packet for this path (since the devices are shared among the paths that traverse the link), but the RE devices on the first link can

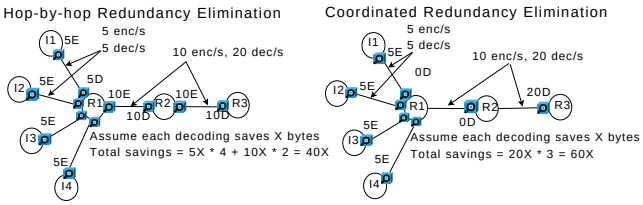


Figure 2: Benefits of coordination when RE devices have constraints on encoding/decoding throughput.

store 4 packets. Each store is managed in a FIFO fashion. The hop-by-hop model yields no benefits from RE on the interior links. A coordinated approach can ensure that the different packets are stored and decoded at different routers. This helps reduce the total traffic by 33%. There are secondary benefits in that routers have to switch smaller packets internally, thereby improving their effective switching capacity. This example shows that a coordinated approach can more effectively use a given amount of memory.

Memory access constraints: Consider the example shown in Figure 2. Here, the links between ingresses I1. . .I4 and the core router R1 are much slower than the core-core links. Assume that the encoding RE device at the slow link can perform 5 packet encodings per second (this corresponds to case #1 from §3.1 where $P = 5$). The encoding RE device at the fast links can perform 10 packet encodings per second (this corresponds to case #2 from §3.1 where $R/F = 10$). Now, consider the decoding devices. The ones on the slow links can decode 5 packets per second, while the ones on the fast link can decode up to 20 packets per second ($R/k = 20$).

In the hop-by-hop case, the number of packets decoded by a downstream RE device is the same as the number of packets encoded by the immediate upstream device. Assuming each decoding saves X bytes, the hop-by-hop approach removes $40X$ bytes ($5X$ on 4 ingress-core router links, and $10X$ on two core-core links). Consider an alternative coordinated scenario, in which the RE devices on interior routers are not involved in encoding and can decode at the maximum rate. In this case, devices on R1 and R2 can just forward encoded packets and R3 can allot its full decoding capacity. This will reduce the total network footprint by $20 \times 3 \times X$. (Since R3 is 3 hops away from the ingress, for each decoded packet we save 3 hops in the network footprint). Also, some of the devices perform no RE function; yet this architecture is $1.5\times$ better than the hop-by-hop approach.

Benefits under partial deployment: In Figure 2, consider a partial deployment scenario with no RE devices attached to router R1. In the hop-by-hop approach, the total savings would only be $10X$ (only on link R2-R3). Note that since the coordinated approach did not involve R1, it provides $60X$ savings even with partial deployment. Network operators can thus realize significantly more benefits with partial deployment with a coordinated design.

The above examples demonstrate the benefits of a hypothetical intelligent and coordinated approach. Next, we describe how we can implement this hypothetical approach in practice.

4. SmartRE DESIGN

In this section, we formally describe the design of SmartRE, an architecture for redundancy elimination that draws on the principles of spatially decoupling encoding and decoding responsibilities, and coordinating the actions of RE devices for maximum efficiency. Our description focuses on SmartRE as applied to an ISP network.

SmartRE synthesizes two ideas: packet caches for redundancy elimination [29, 12] and cSamp [31]. SmartRE leverages ideas from cSamp to split caching (and decoding) responsibilities across multiple router hops in a network. It specifies the caching responsibility of each RE device in terms of a *hash-range per path per device*. Each device is responsible for caching packets such that the hash of the packet header falls in its assigned ranges. By using the same hash function across the network and assigning non-overlapping hash ranges across devices on the same path, SmartRE leverages the memory resources efficiently without requiring expensive cache coordination protocols.

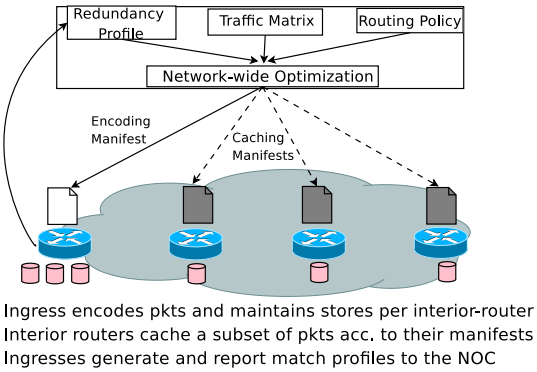


Figure 3: Schematic depiction of SmartRE.

A network operator can specify different ISP-wide objectives, e.g., minimizing network utilization, aiding traffic engineering goals. SmartRE uses a network-wide optimization framework that takes into account the prevailing traffic conditions (volume, redundancy patterns), the network's routing policies, and the capacities of individual RE devices to assign encoding and decoding responsibilities across the network to optimally satisfy the operator's objectives.

4.1 System Overview

We focus our discussion on the design of three key elements (Figure 3): ingress nodes, interior nodes, and a central configuration module. Ingress and interior nodes maintain caches storing a subset of packets they observe.

Ingress nodes *encode* packets. They search for redundant content in incoming packets and encode them with respect to previously seen packets using the mechanism described in §2. In this sense, the role of an ingress node is identical in the naive hop-by-hop approach and SmartRE.

The key difference between the hop-by-hop approach and SmartRE is in the design of *interior* nodes. First, interior elements need not store all packets in their packet cache – they only store a subset as specified by a *caching manifest* produced by the configuration module. Second, they have no encoding responsibilities. Interior nodes only *decode* packets, i.e., expand encoded regions specified by the ingresses using packets in their local packet cache.

The configuration module computes the caching manifests to optimize the ISP objective(s), while operating within the memory and packet processing constraints of network elements. Similar to other proposals for centralized network management (e.g., [18, 15, 13]), we assume that this module will be at the network operations center (NOC), and has access to the network's traffic matrix, routing policies, and the resource configurations of the network elements.

4.2 Network-wide Optimization

The configuration module uses a network-wide view of traffic patterns and resource constraints to compute how and where decoding should be done to optimize ISP objectives.

Assumptions and Terminology: We assume that the traffic matrix (volume of traffic in bytes and packets between every pair of ingress-egress routers) and the routing path(s) between an ingress-egress pair are known and given as inputs. We use the subscripts p and q to indicate paths, r to denote a node (either a router or a

bump-in-the-wire middlebox) and the notation $r \in p$ to denote that node r lies on the path p. v_p is the total traffic volume, in bytes, flowing on path p in a specific measurement interval. $distance_{p,r}$ is the upstream latency (e.g., hop count, OSPF weights, physical fiber distance) of path p up to node r. In our current framework, $distance_{p,r}$ is specified in terms of the hop count.

We also assume that we know the *redundancy profile* of the network from historical traffic data or using periodic reports from ingress nodes. This redundancy profile is specified in terms of two constants for every pair of paths. These are (1) $match_{p,q}$ (measured in packets), the number of matches that traffic flowing through path p observes with traffic on path q and (2) $matchlen_{p,q}$ (in bytes) denoting the average match length observed within these packets (this is bound by the MSS). As a special case, $match_{p,p}$ and $matchlen_{p,p}$ capture intra-path redundancy. As such, our current focus is on redundancy between paths with the same ingress.

The configuration module maximizes the total savings (i.e., minimizing the network footprint or the link utilization-distance product), while respecting the operating resource constraints: i.e., the total available memory (M_r) and the total decoding processing power (L_r) per node. A network operator could specify other network-wide objectives as well.

Formulation: The key variables in the formulation are the $d_{p,r}$ values. Each $d_{p,r}$ specifies the fraction of traffic on path p that node r caches. We now describe how the variables $d_{p,r}$ are determined. First, we model the packet store capacity constraints on each node:

$$\forall r, \sum_{p:r\in p} d_{p,r} \times v_p \leq M_r \quad (1)$$

Next, we model the total packet processing capabilities on each node. The processing capabilities are bound by the number of memory operations that can be performed in unit time.[1] For each interior node, there are two types of memory operations that contribute to the processing load: caching and decoding. We assume for simplicity that both operations are equally expensive per-packet, but it is easy to incorporate other models as well. The total number of packets that will be stored by r on path p is $d_{p,r} \times \frac{v_p}{avgpktsize}$. ($avgpktsize$ appears because v_p is in bytes but the load is per packet.) The total number of matches that will be decoded by node r is $\sum_{p,q:r\in p,r\in q} d_{q,r} \times match_{p,q}$.[2] Thus, we have

$$\forall r, \sum_{p,r\in p} d_{p,r} \frac{v_p}{avgpktsize} + \sum_{p,q:r\in p,q} d_{q,r}\, match_{p,q} \leq L_r \quad (2)$$

There is a natural constraint that the total range covered on each path should be less than or equal to 1:

$$\forall p, \sum_{r:r\in p} d_{p,r} \leq 1 \quad (3)$$

Next, we compute the total savings in the network-wide footprint. The savings provided by node r for traffic on path p ($S_{p,r}$) depends on the redundancy that p shares with other paths that traverse r and the caching responsibility that r has for these paths. It also depends on the location of r on the path p – the more downstream r is (higher $distance_{p,r}$), the greater savings it provides.

$$S_{p,r} = \sum_{q:r\in q} d_{q,r} \times distance_{p,r} \times match_{p,q} \times matchlen_{p,q} \quad (4)$$

The objective then is to maximize $\sum_p \sum_r S_{p,r}$. Note that maximizing this objective, subject to the constraints captured by Equations 1–3 is a linear programming (LP) formulation and thus can be solved efficiently using off-the-shelf LP solvers (we use CPLEX). The output of the LP solver is $d^* = \{d^*_{p,r}\}$, the optimal solution to the formulation.

We can augment this framework to incorporate resource constraints on ingress nodes as well. We omit this extended formulation for brevity, but use it in our evaluation.

4.3 Encoding and Decoding

In the next few sections, we provide details on the actions taken by nodes in the network given the allocations derived by the central configuration module.

Assigning caching responsibilities: The output of the optimization framework is a set of *caching manifests* which specify the caching responsibilities for each node. Each node's manifest is a set of key-value pairs $\{\langle p, HashRange\rangle\}$, indexed by the path identifier p. We use a simple procedure takes in the solution d^* as input and iterates over the paths one by one. For each p, a variable $Range$ (initially zero) is advanced in each iteration per node, in order of location on the path, by the value $d^*_{p,r}$, and node r is assigned the hash range $[Range, Range + d^*_{p,r})$. Thus, nodes on the path p are assigned non-overlapping hash ranges to ensure that the caching responsibilities for nodes on the path are disjoint. We use the on-path ordering to simplify the encoding algorithm (see the discussion in §5.1).

For example, suppose there are three nodes $r1$, $r2$, and $r3$ on path p (in order of distance from the ingress), and the optimal solution has values $d^*_{p,r1} = 0.2$, $d^*_{p,r2} = 0.3$, and $d^*_{p,r3} = 0.1$. The ranges assigned to $r1$, $r2$, and $r3$ for path p will be $[0, 0.2)$, $[0.2, 0.5)$, and $[0.5, 0.6)$.

For each path p, an interior node r only stores packets whose hashes falls within the range assigned to it for p. To do this, the interior node computes a hash over the packet header $\text{HASH}(pkt.header)$ and decides whether or not to cache the packet. HASH is computed over the fields of the packet header that uniquely identify a packet, the src/dst IPs, src/dst ports, protocol, and the IP ID field, and returns a value in the range $[0, 1]$. These are invariant fields that do not change along the routing path [17].

Encoding at the ingresses: We first present a high-level overview of the encoding algorithm at each ingress. We defer to more detailed issues in §5.

Figure 4 shows the pseudocode for an ingress node. The ingress encodes packets with respect to packets in its store. When matches are found, it computes a shim header (Figure 5). The shim header has 2 parts: a fixed length path identifier field specifying the path identifier for the current packet[3], and a (possibly variable length) description of the matches. Each match is specified using three fields: (i) the path identifier for the packet in the ingress's cache with which a match was found, (ii) the unique hash for the matching packet computed over the invariant header fields, and (iii) the matched byte region.

The ingress stores packets whose hashes fall in the total covered range for the path. It ignores other packets as matches with these cannot be decoded downstream. When the ingress cache is full, it evicts packets in FIFO order.

[1] We do not explicitly model CPU constraints because these are subsumed by processing constraints imposed by memory accesses.

[2] Strictly speaking, this is an approximation that assumes that the matches are uniformly spread out across the different $d_{q,r}$ ranges. In practice, this is a reasonable assumption.

[3] If interior nodes can get the pathid from MPLS labels or routing information, this is not necessary.

PROCESSPACKETINGRESS(pkt, $ingress$)

// Steps 1–4 are for encoding
// Use routing/MPLS info for the next two steps
1 $egress \leftarrow$ FINDEGRESS(pkt)
2 $pathid \leftarrow$ GETPATHID($ingress$, $egress$)
// this step depends on the $overlapmatrix$ (see §5)
3 $candidates \leftarrow$ GETCANDIDATES($pathid$)
// $encodedpkt$ carries the shim header (Figure 5)
4 $encodedpkt \leftarrow$ ENCODE(pkt, $candidates$)
// Steps 5–7 are for caching
// what is $\sum_{r \in \text{PATH}(pathid)} d_{pathid,r}$ for this path?
5 $coveredrange \leftarrow$ GETCOVEREDRANGE($pathid$)
// only store packets with hash within covered range
6 $h \leftarrow$ HASH($pkt.header$)
7 **if** ($h \in coveredrange$) **then**
ADDPKTTOSTORE(pkt, $pathid$, h)
// forward as usual
8 FORWARD($encodedpkt$)

Figure 4: Pseudocode for ingress node.

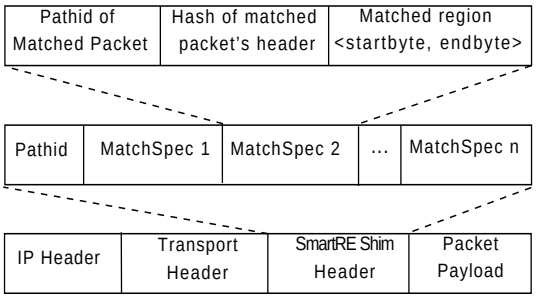


Figure 5: Format of the SmartRE shim header.

Decoding at interior nodes: Figure 6 shows the algorithm at an interior node. The node reads the shim header and checks if any of the matches are in packets that it is currently caching. Each matchspec carries the pathid and the hash of the reference packet with which a match was found. Thus, the interior node can determine if it has cached the reference packet.[4] If so, the node reconstructs the corresponding match region(s). Note that different matched regions may be reconstructed by different downstream nodes as the packet traverses the path.

5. ENSURING CORRECTNESS IN SmartRE

As we saw in the previous section, there are three key features in SmartRE: (1) it allows a packet to be decoded multiple hops downstream from the ingress where it was encoded, (2) it splits caching (and decoding) responsibilities along the RE elements on a path, and (3) it uses a network-wide approach for allocating caching responsibilities.

These three features are essential for efficiently utilizing the available RE resources (e.g., caches, memory accesses) to derive close to optimal network-wide benefits. For example, (1) means that each decoding operation performed by an interior router H hops downstream is H times as effective in reducing the network-wide footprint as the same operation performed by the router adjacent to the ingress. Similarly, (2) means that each cache entry is utilized efficiently. (3) combines these features to achieve network-wide goals; this could mean that RE elements common to paths that share redundant content are assigned inter-path decoding responsibilities.

[4]Errors due to hash collisions are highly unlikely.

PROCESSPACKETINTERIOR($encodedpkt$, r)

// r is the node id
// Steps 1–2 are for decoding
// Check if any decoding needs to be done
1 $mymatches \leftarrow$ PROCESSSHIM($encodedpkt.shim$)
// this may only partially reconstruct the packet
2 $decodedpkt \leftarrow$ DECODE($encodedpkt$, $mymatches$)
// Steps 3–6 are for caching
3 $pathid \leftarrow$ GETPATHID($encodedpkt$)
// what is my assigned hash range for this path?
4 $myrange \leftarrow$ GETRANGE($pathid$, r)
5 $h \leftarrow$ HASH($pkt.header$)
6 **if** ($h \in myrange$) **then**
ADDPKTTOSTORE($decodedpkt$, $pathid$, h)
// forward as usual
7 FORWARD($decodedpkt$)

Figure 6: Pseudocode for an interior node.

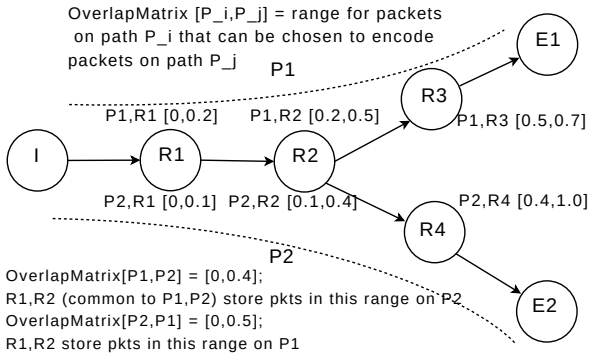


Figure 7: Example showing the overlap matrix.

However, these features raise some issues with respect to correctness; i.e., will an encoded packet be decoded correctly before it leaves the network perimeter. Specifically, we identify three issues:

1. How can an ingress decide if encoding a packet w.r.t a previous packet will be valid–will that previous packet be available in a cache on the path taken by the current packet? (§5.1)
2. Since interior elements may be assigned responsibilities across multiple ingresses, how does each encoder maintain a consistent view of the caches at interior elements? That is, if an ingress encodes a packet, will the decoders have the required matched packets or would they have evicted them? (§5.2)
3. As decoding responsibilities are split across a path, some packets may be encoded when they reach their assigned caching nodes. Should we cache such encoded packets? (§5.3)

We present lightweight solutions to address these issues in the context of SmartRE. However, the issues themselves are more general to the design of network-wide RE solutions.

5.1 Identifying valid inter-path encodings

If the ingress identifies a match with a packet that traversed the same path it can encode the match. However, when the ingress sees a match with a packet from another path, it needs to ensure that this can be successfully decoded downstream. The $overlapmatrix$ specifies *valid* inter-path encodings, and in Figure 4, the function GETCANDIDATES checks $overlapmatrix$ to find valid encodings.

Figure 7 shows a simple example of what the overlap matrix means. We have two paths P1 and P2. The caching responsibilities of each node are specified in terms of hash-ranges per path. Suppose a new packet A belonging to P1 arrives at I. I finds a

match with packet B sent earlier along P2. Now, I has to decide whether A if encoded w.r.t B can be decoded downstream. If HASH$(B) \leq overlapmatrix[P1, P2]$, one of R1 or R2 will be able to decode the match. Otherwise, B is stored on nodes that do not lie on P1 and thus A cannot be encoded with respect to B.

Let us go back to the discussion of on-path ordering (§4.3). The configuration module generates the $overlapmatrix$ from the LP solution and distributes it to the ingresses. On-path ordering ensures that each entry in this matrix is one contiguous range instead of several disjoint ranges. This simplifies the description of the $overlapmatrix$ and also simplifies the process by which the ingresses identify valid encodings.

5.2 Using cache buckets to ensure consistency

In hop-by-hop RE, each node's packet store is perfectly in sync with the upstream node's packet store. However, SmartRE needs to explicitly ensure that ingress and interior caches are consistent.

To see why this is necessary, consider the following scenario. Packet X is initially cached at interior node R and the ingress I. Consider the case when R and I maintain independent FIFO caches. Suppose X is evicted from R's cache due to a sudden increase in traffic along paths from other ingresses. Now, packet Y arrives at I. I finds a match with X and encodes X with respect to Y. Clearly, R will not be able to reconstruct the matched region for Y. The packet Y would thus have to be dropped downstream or rejected by the application at the end-host.

To address this, we use a lightweight, yet robust, consistency mechanism. The main idea is to divide the ingress packet store into *buckets*; each bucket corresponds to a hash range assigned to a specific interior node-path pair. Interior stores are organized similarly. As a packet arrives at the ingress, it is stored into the per-path per-range bucket into which its hash falls. This explains the parameters $pathid$ and h to ADDPKTTOSTORE in Figures 4 and 6 – together they identify the bucket in which to store the packet. Each bucket is a circular buffer; as a bucket gets full, packets get evicted in FIFO order to accommodate newer packets. The size of each bucket is determined by the LP solution and the traffic patterns (i.e., $d^*_{p,r} \times v_p$); the configuration module also specifies these sizes as part of the caching manifests. When new solutions are computed in response to traffic or routing dynamics, the bucket sizes can be reassigned appropriately.

5.3 Handling gaps in encoded packets

An interior node may not have the full payload for packets for which it is assigned caching responsibilities. This could happen if at the time the packet reaches this node, there is still some decoding to be done downstream. Thus, the node only sees a partially reconstructed packet. This creates a problem if subsequent packets need to be encoded with respect to a packet with some decoding "gaps". To see why this is an issue, consider the example in Figure 8. In the example, even though the ingress can encode C with respect to its cached version of B, $R1$ which is storing an incomplete version of B cannot decode this match.

One option is that the ingress does not use encoded packets for future encodings. Thus, packet B which was encoded with respect to A is not even stored at I. Another option is to use these encoded packets *maximally*, i.e., all non-gap regions in the packet are used to match further packets. Thus, router I in the example stores B but nullifies the bytes in B that matched A. Future packets can only be encoded with respect to non-null regions of B. Both solutions ensure correct end-to-end packet delivery, but provide lower redundancy elimination than the ideal case when there are no decoding gaps. Since the second solution achieves better redundancy elimination, we implement this option. In our experiments with real packet traces, we found that with the second option, the effective loss in redundancy elimination is less than 3%.

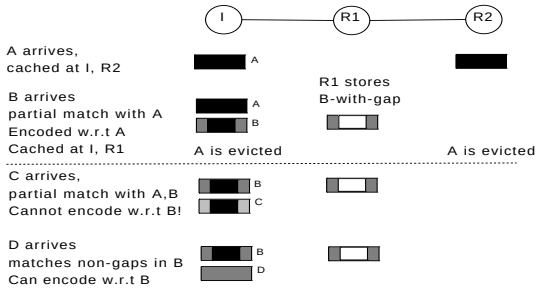


Figure 8: Example of how decoding gaps may occur.

6. IMPLEMENTATION ISSUES

6.1 Encoder and Decoder Implementation

We implement the encoding and decoding algorithms from §4.3 and §5 in Click [21]. The key components of the encoder are: fingerprint computation per packet, a packet store for caching packets, and a hash table for mapping fingerprints to the packets they were found in (similar to [29, 12]).

Like most RE systems, we use Rabin fingerprinting [24]. Each Rabin fingerprint captures a fixed 64 byte region in a packet [12]. We store a maximum of $F = 10$ fingerprints per packet in the fingerprint hash table. This reflects a reasonable throughput-redundancy tradeoff based on real traces.

We segment the packet store into logical buckets per interior-node-path pair (§5.2). The encoder inserts each packet into the appropriate bucket in FIFO order. In addition to payloads, we store the IP headers for each packet because a hash of the headers is used to decide decoding and storage responsibilities (Figure 5). Also, the encoder flags one bit in the IP header (e.g., TOS field) to indicate that the packet has one or more shims that need to be decoded.

In prior RE solutions [29, 12], each fingerprint in the fingerprint hash table is associated with the most recent packet for which it is computed. In SmartRE, this raises issues with packets being undecodable due to gaps. (To elaborate, this most recent packet may itself have been encoded and thus further encodings with respect to this packet will lead to decoding gaps as discussed in §5.) To address this issue, when a packet sees a match and the match region is grown to the maximal byte range, the fingerprints of this packet that mapped into the maximal range are re-associated with the matched in-cache packet. Also, the maximal byte range in the incoming packet is zeroed out. This ensures ensure that bytes in the maximal match region are not used for encoding. Our implementation is thus conservative; we sacrifice some performance in favor of correctness.

The decoder implementation largely follows the discussion in §4.3. The last decoder on a path clears the flag in the header indicating that the packet has been fully decoded.

6.2 Configuration Parameters

Parameters for the LP optimization: To specify parameters to the LP formulation, we need to fix a certain measurement epoch. However, this epoch cannot be arbitrary, as the RE capabilities are limited by the storage available at the ingresses. Thus, we define the notion of a *network data retention time* determined by the size of the ingress packet stores. All values in the formulation (i.e., the match profiles and the traffic matrix) are specified in terms of this common value. In real deployments, we expect ISPs to employ ingress caches storing few tens of seconds worth of data.

Traffic and routing dynamics: The dominant source of traffic dynamics are time-of-day and day-of-week effects [25]. Fortunately, these are predictable and we can use historical traffic matrices to model these effects.

Routing changes are trickier because an ingress may incorrectly assume that a downstream node will be able to decode a match. Two scenarios arise. First, if routes are computed centrally [18], SmartRE can use the new routes to recompute a new caching strategy and disseminate it to the ingresses. However, the recomputation may take few tens of seconds, and we need to ensure correctness during this transient state. Second, the ingresses do not receive new caching strategies, but instead receive the current routing information (e.g., OSPF monitor [27]) and avoid encodings that are non-decodable after the routing change. This ensures correctness but sacrifices some performance. Note that this also solves the transient problems in the first scenario.

Changes in redundancy profiles: To estimate the redundancy profiles, the ingress RE devices maintain simple counters to track matches between paths. The ingresses periodically report these values to the central configuration module. Note that this adds very little overhead to the ingress implementation. However, since these could be large,[5] they will be reported infrequently (e.g., every 30 minutes).

This raises the issue of staleness of redundancy profiles. This may have two effects: (1) It may affect the optimality of the configuration without affecting solution correctness. This is an acceptable operating mode for SmartRE and we evaluate it further in §7. (2) Significant changes in the redundancy profile may increase decoding load on each node (§4.2, Equation 2) and affect solution feasibility. To handle the second issue, each ingress tracks the actual number of matches per interior node and will not burden overloaded interior nodes with additional decoding responsibilities. Thus, changes in redundancy profiles do not affect correctness.

Additionally, SmartRE can use a *triggered* approach. For example, under flash-crowd-like scenarios where traffic patterns change dramatically, the affected ingresses can report the large changes to the NOC. This can trigger an immediate recomputation of the caching manifests instead of the periodic recomputation.

6.3 More on Correctness

Consistent configurations: The bandwidth overhead for dissemination is low as the configuration files are quite small (1-2 KB per device). However, differences in the distances between the devices and the NOC could lead RE devices to use inconsistent caching configurations. To mitigate this, we can use latency information from topology maps to schedule the transfers to ensure that all devices receive the new configurations at approximately the same time. Also, for a small transition interval (few tens of milliseconds), all RE devices honor both configurations. That is, the encoders and decoders store packets assigned by either the old configuration or the new one. (RE devices can allot a small amount of spare memory for this purpose). This may result in a small performance reduction, as some packets may get decoded before their optimally assigned decoders, but it ensures correct packet delivery.

Errors due to packet drops: Packet drops can cause encoder and decoder caches to get out of sync. Packet drops cause two issues: (1) Packets which are encoded w.r.t the dropped packet cannot be decoded downstream; (2) When the higher-layer application retransmits the dropped packet, it is likely that the retransmission will get encoded with respect to the dropped packet, and get dropped again. TCP-based applications can typically recover from single packet drops in a window, but drops of retransmitted packets (case #2) severely impacts TCP throughput. We handle the latter as a special case. If an ingress sees a packet which has a full content match and the same connection 5-tuple match with an in-cache packet, it will not encode this packet.

[5]With n access routers, there are $O(n^2)$ paths. Even restricting to paths with the same ingress, the overhead for transmitting redundancy profiles is $O(n^3)$.

Network (AS#)	PoP-level # PoPs	PoP-level Time	Router-level # Routers	Router-level Time
NTT (2914)	70	0.92	350	55.41
Level3 (3356)	63	0.53	315	30.06
Sprint (1239)	52	0.47	260	21.41
Telstra (1221)	44	0.29	220	16.85
Tiscali (3257)	41	0.21	205	11.05
GÉANT	22	0.07	110	2.48
Internet2	11	0.03	55	0.48

Table 1: LP solution time (in seconds).

7. EVALUATION

Our evaluation is divided into the following sections: (1) Benchmarks of the Click prototype and time taken by the optimization framework. (2) Benefits of SmartRE compared to the ideal and naive approaches using synthetic traces with different redundancy profiles and resource provisioning regimes. (3) Evaluation using real packet traces collected at a large US university's border router and at a university-owned /24 prefix hosting popular Web servers. (4) Impact of staleness of redundancy profiles. (5) Benefits under partial deployment.

For the following results, we use PoP-level ISP topologies from Rocketfuel [28] and add four access routers to each PoP to obtain router-level topologies.

7.1 Performance benchmarks

LP solution time: Table 1 shows the time taken to generate the caching manifests (on a 2.80 GHz machine) for seven PoP- and router-level topologies. Even for the largest router-level topology (NTT), the time to solve (using `CPLEX`) is < 60s. We envision that reconfigurations occur on the scale of a few minutes – this result shows that the optimization step is fast enough to support such reconfigurations.

Encoding and decoding rates: We now try to understand how the encoders and decoders can be used in practical ISP deployments. To do so, we benchmark the implementations on a standard desktop machine and extrapolate the performance to more realistic settings.

We run our prototypes on a desktop with 2.4GHz CPU, with a DRAM latency of 90ns (benchmarked using PAPI[6]). We use real packet traces from the /24 prefix. (This trace was 35% redundant using a 600 MB packet cache and 10 fingerprints per packet.) In addition to computing the raw throughput, we also compute the effective throughput after subtracting the overhead due to Click operations. This extrapolates the results to a SmartRE middlebox implemented on an FPGA [20] which would be constrained only by memory accesses and have no software overhead.

First, we benchmark the encoder. To understand the maximum throughput of a memory-bound RE middlebox, we follow the methodology of Anand et al. [12]: (1) load the packet trace into memory, (2) precompute and load fingerprints for all packets into memory, (3) encode packets one by one, and report the throughput.

We configured a packet store to hold 600MB of packet payloads; the corresponding fingerprint index was 400MB in size. Using 10 fingerprints per packet, the effective throughput obtained for encoding was around 2.2Gbps (after subtracting the Click overhead). We also ran this on a machine with 120ns memory latency and

[6]`http://icl.cs.utk.edu/papi/`

# Match Specs	Redundancy	Throughput (Gbps)	
		In software	W/o overhead
1	24.1%	4.9	8.7
2	31.6%	4.5	7.9
3	34.6%	4.3	7.7
4	34.7%	4.3	7.6
5	34.8%	4.3	7.6

Table 2: Trade-off in redundancy and decoding throughput with number of match-specs.

the throughput dropped to 1.5Gbps. Extrapolating, we conclude that with lower DRAM latencies, the encoder can operate at OC-48 linerates. (Today's high-end DRAMs have $\leq$ 50ns latency as opposed to 90ns on our desktop). Other SmartRE operations, e.g., redundancy profile computation, storing in isolated buckets etc., add negligible overhead.

Next, we evaluate the decoding throughput. This depends on the number of match regions encoded in packet shims: as more regions get encoded, more redundancy is identified, but the throughput decreases as the number of memory accesses increases. We study this tradeoff in Table 2. The decoding store size was set to 600MB. We see that decoding is roughly 3-4$\times$ faster than encoding, since it involves fewer memory operations per packet. While decoding throughput does decrease with more matches (due to more memory accesses), the decrease is small for ≥ 2 matches. Our implementation uses a maximum of 3 match-specs as a tradeoff between the amount of redundancy identified and the throughput.

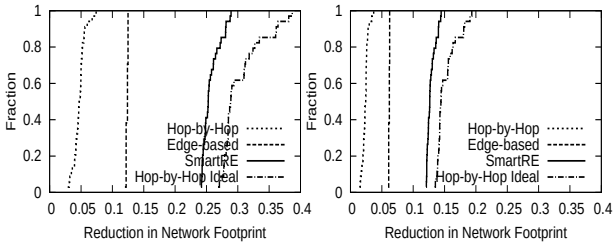
Our simple encoder and decoder implementations can roughly operate on OC-48 (2.5Gbps) and OC-192 links (10Gbps), respectively. In networks where such links are used, SmartRE can leverage the encoding and decoding capabilities of nodes to give optimal benefits. Middleboxes based on these simple designs can also be used in ISPs that employ faster links, e.g., 40Gbps for the core. The only difference is that each decoder may be able to act only on one-fourth of the packets entering the router; the rest of the packets need to be decoded at other locations. In this case, the benefits of SmartRE may not be optimal. We explore the gap between SmartRE and the optimal in greater detail next.

7.2 Synthetic trace study

We compare the benefits of SmartRE, the hop-by-hop approach without any resource constraints (i.e., *hop-by-hop ideal*), the hop-by-hop approach with actual resource constraints, and a special case of SmartRE called edge-based RE. In both SmartRE and edge-based RE, encoding is a one-time task; performed only at the ingresses. However, decoding happens only at the edge of the network in edge-based RE, unlike SmartRE. While SmartRE can effectively operate under all types of redundancy profiles, edge-based RE is effective only when intra-path redundancy is the dominant source of repeated content. Hop-by-hop ideal represents the best possible benefits achievable from network-wide RE assuming that RE devices are unconstrained. Our main goals are to understand how close to ideal SmartRE gets, how much better it is than other approaches, and what factors contribute to SmartRE's performance.

Setup: We implemented an offline emulator using Click to compare different network-wide RE solutions. We assume a middlebox deployment where each network link has RE devices attached on both ends of the link. For SmartRE, the device at one end of a link is used for decoding/encoding packets in one direction, and the one at the other end is used for the reverse direction.

Encoders at each access link store T seconds of packets (e.g., 3 GB memory at 2.4 Gbps implies $T = 10$s). Decoders at the edge have the same cache size as the encoders. Each interior RE device uses a 6GB cache which we consider to be reasonable from a cost view-point in practical settings; we also evaluate the effect of varying this cache size. We model the throughput of each device in terms of the total number of memory operations per second. We select bounds that reflect the throughput achieved by our software prototypes. Assuming (conservative) memory latencies of 100ns, 20 lookups for encoding each packet, and 4 lookups in total for decoding each packet, this translates into 0.5 million encodings and 2.5 million decodings per second respectively.



(a) 50% redundancy trace (b) 25 % redundancy trace

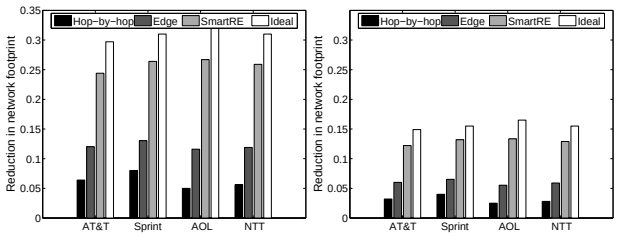
Figure 9: CDF of network footprint reduction across ingresses for Sprint (AS1239) using synthetic traces.

Traffic model: We use a gravity model based on city populations to determine the fraction of traffic from each ingress access router to an egress PoP. Within each PoP, the traffic is divided equally among the 4 access routers. Each trace's redundancy profile is specified by three parameters: γ, $\gamma_{intrapop}$, and $\gamma_{intrapath}$. γ is the overall traffic redundancy per-ingress access link. $\gamma_{intrapop}$ determines the redundancy within the traffic destined for the same egress PoP. Within each egress PoP, $\gamma_{intrapath}$ determines the intra-path redundancy of the end-to-end path between the ingress and egress access routers. These parameters specify how redundant the traffic is, and how localized or how dispersed the redundancy profile is. If γ is high then the traffic is highly redundant; if $\gamma_{intrapop}$ is high then most of this redundant traffic is destined to the same PoP; if $\gamma_{intrapath}$ then most of the intra-PoP redundancy is within the same ingress-egress path.

Results: We first consider the single-ingress case, where traffic originates from a single ISP PoP. In this case, the decoding capabilities in the network are split proportionally by volume across all ingress-access routers; on each link L, each ingress I's share is $\frac{vol_I(L)}{vol(L)}$, where $vol_I(L)$ is the volume of traffic originating at ingress I flowing through link L and $vol(L)$ is the total volume of traffic through L from all ingresses. The following results use two configurations with $\gamma = 25\%$ and $\gamma = 50\%$ redundancy, with $\gamma_{intrapop}$ and $\gamma_{intrapath}$ set to 0.5 in each case. Our choice of γ is based on measurements of redundancy in real traffic traces from enterprise and university networks [10].

Our main metric of interest is the fractional reduction in the network footprint (§4). Figure 9 shows a CDF of the reduction in network footprint for the four solutions for the Sprint topology. The footprint reduction of SmartRE is 24-30% across the ingresses for the 50%-redundant trace (12-15% for the 25%-redundant trace), indicating the extent to which the aggregate utilization of the ISP improves for traffic from the ingress in question. The median fractional reduction across the ingresses for the 50%-redundant trace in SmartRE is 5$\times$ better than the naive approach. More importantly, the median value is less than the ideal unconstrained case with no processing and memory constraints by only 0.04 in absolute terms.

Figure 10 shows the network-wide reduction for 4 tier-1 ISPs. Here, we consider the top 20 PoPs (by degree) in each topology, and assume that the total traffic entering each of the 80 ingresses (4 per PoP) is the same. For simplicity, we also assume that the redundancy profile is the same across all ingresses. Across the different topologies, SmartRE is consistently $4\times$ better than the naive approach; even the edge-only variant of SmartRE is roughly $2-3\times$ better than a naive approach. Also, SmartRE is quite close to the unconstrained ideal case and provides 80-90% of the ideal savings.



(a) 50% redundancy trace (b) 25 % redundancy trace

Figure 10: Network-wide footprint reduction for four tier-1 ISP topologies using synthetic traces.

Importance of SmartRE optimizations: SmartRE takes into account three factors while assigning caching responsibilities across RE devices in the network: (1) memory constraints on RE devices, (2) packet processing constraints imposed by memory accesses, and (3) traffic and routing matrices and redundancy profiles. We evaluate the relative importance of these next.

To do so, we consider four hypothetical scenarios:

1. SmartRE with no memory constraints (SmartRE-nomem); setting each $M_r = \infty$ in the LP from § 4.2.
2. SmartRE with no packet processing constraints (SmartRE-noproc); setting each $L_r = \infty$ in the LP .
3. A heuristic (Heur1) where the hash-ranges are divided equally across the RE devices on a path – if there are k RE devices on the path p, each caches $\frac{1}{k}$ of the packets on this path.
4. A second heuristic (Heur2) similar to the one above, except that RE devices further downstream are assigned more caching responsibilities. Specifically, if path p has k hops, then the i^{th} hop caches $\frac{i}{\sum_{j=1}^{k} j}$ of the packets on this path.

Table 3 compares the performance of these schemes with SmartRE and the ideal solution with no resource constraints. Note that Heur1 and Heur2 are also resource aware; the effective caching and decoding responsibilities are capped off by the actual memory and processing constraints. We see three effects. First, SmartRE performs significantly better than both heuristics showing that accounting for traffic, routing, and redundancy patterns while assigning caching responsibilities is necessary. Second, the gap between SmartRE-nomem and SmartRE is negligible. This is because cache size has a natural diminishing property (see Figure 11); it is necessary to have a sufficiently large cache but increasing it further does not help much. Finally, relaxing processing constraints does not help too much. This is because the core RE devices are not overloaded for the redundancy profile we use for this evaluation ($\gamma_{intrapop} = \gamma_{intrapath} = 0.5$) and perform fewer decodings than their effective capacity. However, in other redundancy profiles where the core devices operate at full capacity, the gap between SmartRE and SmartRE-noproc is more noticeable (not shown).

SmartRE with no resource constraints is still 0.04 lower than the ideal solution. This is an effect of enforcing non-overlapping caches. For example, consider two paths $\langle X, A, B\rangle$ and $\langle X, A, C\rangle$ with the same ingress X and a packet P along $\langle X, A, B\rangle$ that matches future packets on both paths. If we allow caches to overlap, P can be stored on both A and B, to achieve optimal RE. If we use non-overlapping caches, P can be on either A or B, but not both. This sacrifices either inter-path RE (if we store P on B alone) or the footprint reduction for intra-path RE (if we store P on A alone). Allowing caches to overlap can yield better RE when there are no memory constraints. However, overlapping caches are not optimal in realistic settings with actual resource constraints. Further, there are other practical difficulties in extending SmartRE to allow overlapping caches (see §8).

Topology	Heur1 (equal)	Heur2 (distance)	SmartRE	SmartRE nomem	SmartRE noproc	Ideal
Sprint	0.145	0.168	0.264	0.267	0.274	0.31
ATT	0.138	0.162	0.244	0.248	0.262	0.297
AOL	0.152	0.178	0.267	0.277	0.278	0.33
NTT	0.142	0.167	0.259	0.264	0.278	0.31

Table 3: Understanding the relative importance of the different components of SmartRE's optimization.

$(\gamma_{intrapop}, \gamma_{intrapath})$	Reduction in network footprint			
	SmartRE	Edge	Hop-by-hop	Ideal
$(0.5, 0.5)$	0.26	0.12	0.08	0.31
$(0.5, 0.75)$	0.28	0.18	0.08	0.31
$(0.75, 0.75)$	0.38	0.27	0.11	0.42
$(0.25, 0.5)$	0.18	0.05	0.06	0.20

Table 4: Exploring different redundancy profiles on the Sprint topology, with total redundancy $\gamma = 0.5$.

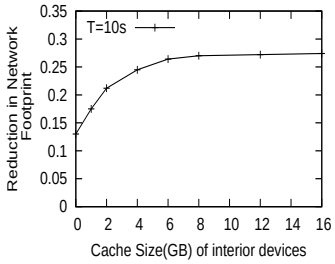


Figure 11: Varying cache size in the interior using a synthetic trace over the Sprint topology.

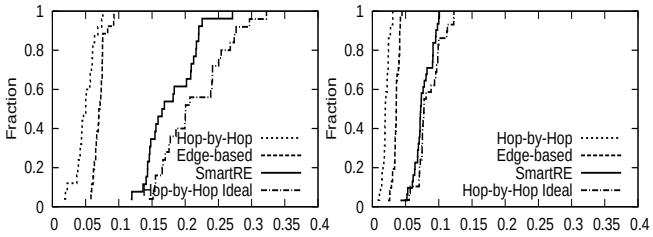
Varying redundancy profiles: Table 4 compares different types of redundancy profiles. While SmartRE is consistently better, the improvement depends on the redundancy profile. For example, when intra-path redundancy dominates $(0.75, 0.75)$, SmartRE is not significantly better than the edge-based variant. Again, across all the profiles, SmartRE is within 0.04 of the ideal unconstrained case.

The configuration $(0.25, 0.5)$ where there is significant redundancy across egress PoPs should be ideal for SmartRE. However, all three approaches fare poorly, and hop-by-hop marginally outperforms the edge-only approach. The latter does poorly in this case because most of the redundancy is inter-path, not intra-path. We were surprised at why SmartRE and even the ideal case did worse in this scenario. We find that shortest path routing between the top-20 PoPs in this ISP does not allow for much scope for on-path coordination between paths because the paths have very few hops in them. In this context, redundancy-aware routing [12] can additionally boost the performance of SmartRE.

Memory provisioning: Figure 11 shows the effect of adding more cache memory to interior devices, while keeping the cache size on the edge devices fixed. Adding cache memory to the interior has two benefits. (1) The total on-path memory increases and greater intra-path redundancy is identified. However, this increase happens only up to a certain point when the total memory on a path matches the memory used for encoding. (2) Interior nodes see redundancy between paths from same ingress destined to different egresses. The amount of inter-path redundancy increases monotonically with memory. Adding more memory to core devices leverages such sources of redundancy that cannot be identified in an edge-only approach. While adding more memory in the core exploits more redundancy, the benefits are marginal beyond 4GB. Beyond this, the amount of inter-path redundancy identified is small.

7.3 Evaluation Using Real Traces

We use packet traces collected at a large US university to examine the effectiveness of SmartRE with real traffic patterns. To simulate a real trace over a specific topology, we map the observed



(a) High volume /24 trace (b) University trace

Figure 12: CDF of network footprint reduction across ingresses on Sprint topology extrapolating from real traces.

IP addresses to the nearest PoP in the ISP topology. We used one trace capturing all traffic leaving the university (which was 15% redundant with 10s of encoding cache) and another trace for traffic leaving the /24 prefix (40% redundant).

We start with the single-ingress case. Figure 12 shows the CDF of footprint reduction on the Sprint topology using both all-university and /24 prefix traces. Again, SmartRE outperforms the hop-by-hop approach by 4-5×. In the University trace, SmartRE is almost indistinguishable from the ideal case; in the /24 trace the median performance difference is 0.04.

We observed substantial variance in the relative performances of the naive approach and SmartRE across different ingresses (not shown). We explored this further, focusing on the top-4 ingress PoPs in the topology (by degree). For two of the PoPs (Seattle and Dallas) SmartRE is 7-8× more effective than the naive approach. For the remaining two (New York, Chicago), it is 3-4× better. There are two factors here. First, a majority of the traffic is destined to New York and Chicago and there is considerable overlap within this traffic. Second, the paths from the other two PoPs to New York and Chicago share many intermediary nodes. Thus, SmartRE can better exploit this inter-path redundancy.

We also conducted the network-wide evaluations across 4 ISP networks. SmartRE reduced the network-wide footprint by 20% and 13% on average across the 4 networks for the /24 and all-university traces respectively.

7.4 Effect of Stale Redundancy Profiles

As discussed in §6, SmartRE uses the redundancy profile observed in the current epoch to compute caching manifests for the next epoch. We evaluate the impact of using stale redundancy profiles (SmartRE-stale) compared to SmartRE-ideal which uses up-to-date information (as in the rest of this section so far).

We study variants of SmartRE-stale which differ in the time between when redundancy profiles were computed and when they are used. We use the real packet traces from §7.3 for this study. We evaluate time lags of 10, 20, 30 and 40 minutes (not shown). We find that SmartRE-stale performs close to SmartRE-ideal (and hence ideal RE), with the worst-case footprint reduction being at most 0.05 worse than SmartRE-ideal. We investigated why SmartRE performs well even with a stale redundancy profile and found that the traffic volume to the large cities (Chicago and New York) dominates the overall benefits and the redundancy profiles for these are stable. While these results are preliminary, they are encouraging– the dominant sources of redundancy appear to be stable and SmartRE can provide benefits even with stale redundancy profiles.

Flash-crowd scenarios: Next, we study how staleness can affect RE performance in more sudden flash-crowd-like scenarios. First, we increase the total traffic volume entering at a particular ingress to saturate its upstream bandwidth, keeping the redundancy at each ingress fixed at 50%. In this setup, the footprint reduction is 0.26

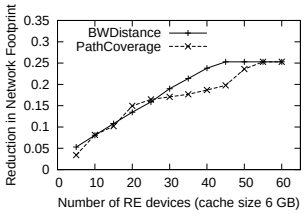


Figure 13: Two partial deployment strategies on the Sprint topology (x=65 represents full deployment). Each device has a 6GB cache.

with an up-to-date traffic matrix and redundancy profile; with older inputs the reduction is $0.23 - 0.25$ depending on the ingress. Second, we increase the aggregate redundancy for a specific ingress from 25% to 50%, keeping the redundancy from other ingresses fixed at 25%. Depending on the ingress that has increased redundancy, the footprint reduction is $0.14 - 0.15$ with up-to-date profiles and $0.10 - 0.11$ with an old profile. These experiments further confirm that while up-to-date profiles yield better RE performance, even stale profiles can yield substantial benefits. However, for dramatic changes, profiles should be updated using the triggered update mechanism discussed in §6.

7.5 Partial deployment benefits

The middlebox-style implementation of encoders and encoders makes SmartRE amenable to incremental and partial deployment, in that the encoders/decoders can be installed at locations where reduction in network load is desired most.

We emulate a situation where an ISP would like to mitigate the impact of redundant traffic originating from certain high-volume PoPs (say, top 5 by volume) by deploying RE middleboxes strategically in its network. (Encoding RE boxes are deployed at each of a PoP's ingress access links). We ask if SmartRE is useful even on a limited scale.

We examine two strategies. In both cases, our goal is to deploy RE boxes where there is a lot of traffic aggregation. We first count the number of shortest path routes traversing each interior link. In the first strategy we simply deploy decoders on links which lie on many of the network paths from the 5 ingresses in question to other egresses. The second strategy is smarter, in that it first weighs each path traversing a link by the volume of traffic it carries and the distance of the link from the corresponding ingress, and ranks links according to the total weights of paths traversing them.

Figure 13 shows that in both cases, deploying RE middleboxes on a small number of links (e.g., < 10 out of a maximum of 65) still offers reasonable benefits in network-wide utilization (roughly 10% compared to the best possible 26%). The smarter strategy works better with 50% - 70% deployment. Figure 13 indicates that for partial deployments even simple strategies work well. This can be further enhanced by weighing each path with the expected amount of redundancy based on historical observations.

7.6 Evaluation Summary

- SmartRE is on average 4-5× more effective than a naive hop-by-hop approach.
- SmartRE, even under strict resource constraints on both memory and memory access throughput, achieves 80-90% of the performance of an ideal unconstrained RE solution which assumes no memory or processing constraints.
- The above results are consistent across several redundancy profiles and on both synthetic and real traces.
- The global resource-aware optimization in SmartRE is necessary for good RE performance; simple heuristics for as-

signing caching responsibilities do not yield sufficient network footprint reduction.

- SmartRE can provide benefits comparable to the ideal scenario even under partial deployment or with slightly out-of-date redundancy profiles.

8. DISCUSSION

Multi-hop wireless: We believe that SmartRE can be used to enhance caching systems in other contexts, e.g., multi-hop wireless networks [16]. Coordinated caching can help in two ways here: (1) improving the effective memory usage at multihop nodes by chunking large transfers and apportioning each chunk to a specific node (this replaces blind caching at all on-path routers) and (2) preventing multiple nodes from retrieving a popular chunk from a single cache - this creates contention for the medium and may wipe out the benefits of caching. We can limit each cache's encoding responsibilities and this creates an even distribution of caching/encoding across nodes in the network.

Allowing overlapping ranges in SmartRE: We saw in §7.2 that allowing caches to overlap may improve RE performance. However, there are two practical difficulties. First, the formulation from §4.2 becomes more complicated. Specifically, we can no longer model the second term in Equation 2 and the savings term in Equation 4 as linear expressions; in fact, it is not even clear if we can precisely model these terms. Thus, it is difficult to obtain the optimal caching responsibilities in this setting. Second, in order to maintain a consistent view with every decoder each ingress has to either (a) keep duplicate copies of packets that belong to overlapping ranges or (b) use additional mechanisms to keep track of whether a packet has been evicted from an interior node and also maintain the appropriate mappings between fingerprints to the packets in the store. Additionally, the ingress needs to explicitly decide which of the decoders is responsible for reconstructing encoded regions in case the matched packet is cached on multiple downstream nodes. The performance of SmartRE with non-overlapping ranges is already close to the ideal scenario. Thus, we do not consider this extension to allow overlapping caches because the marginal improvement does not merit the increased implementation complexity.

9. CONCLUSIONS

As Internet traffic volumes increase and more bandwidth-intensive applications appear, redundancy elimination (RE) has emerged as a promising practical solution to increase end-to-end application throughput. More recently, there has been interest in expanding the scope of RE to network-wide scenarios with the grander vision of offering this as a primitive IP-layer service within ISP networks.

This paper takes this vision one step closer to reality. We look beyond a naive link-by-link view and adopt a network-wide coordinated approach. We design and implement a framework called SmartRE based on these high-level design principles. SmartRE is naturally suited to handle heterogeneous resource constraints and traffic patterns and for incremental deployment. We address several practical issues in the design to ensure correctness of operation in the presence of network dynamics. Across a wide range of evaluation scenarios, SmartRE provides 4-5$\times$ improvement over naive solutions and achieves 80-90% of the performance of an ideal, unconstrained RE network-wide alternative.

A natural extension is to apply SmartRE to datacenter and multi-hop wireless networks. Another area of future work is to expand the scope for RE by allowing multiple encoders per-path (in contrast to encoding only at the ingress) and exploring the interplay between RE techniques and network coding.

Acknowledgments

We thank Tom Anderson, Flavio Bonomi, Bruce Davie, K. K. Ramakrishnan, Srini Seshan, David Wetherall, and the anonymous reviewers for their valuable feedback that helped improved our paper. This work was supported in part by an NSF CAREER Award (CNS-0746531) and an NSF NeTS FIND Award (CNS-0626889).

10. REFERENCES

[1] Akamai Technologies. `http://www.akamai.com`.
[2] BlueCoat: WAN Optimization. `http://www.bluecoat.com/`.
[3] Cisco Content Aware Networks – Some Areas of Interest. `http://www.cisco.com/web/about/ac50/ac207/crc_new/ciscoarea/content.html`.
[4] Cisco Wide Area Application Acceleration Services. `http://www.cisco.com/en/US/products/ps5680/Products_Sub_Category_Home.html`.
[5] Citrix, application delivery infrastructure. `http://www.citrix.com/`.
[6] Computerworld - WAN optimization continues growth. `www.computerworld.com.au/index.php/id;1174462047;fp;16;fpid;0/`.
[7] PeerApp: P2P and Media Caching. `http://www.peerapp.com`.
[8] Riverbed Networks: WAN Optimization. `http://www.riverbed.com/solutions/optimize/`.
[9] WAN optimization revenues grow 16% - IT Facts. `www.itfacts.biz/wan-optimization-market-to-grow-16/1205/`.
[10] A. Anand and C. Muthukrishnan and A. Akella and R. Ramachandran. Redundancy in Network Traffic: Findings and Implications. In *Proc. of SIGMETRICS*, 2009.
[11] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *Proc. of SOSP*, 2001.
[12] 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 SIGCOMM*, 2008.
[13] H. Ballani and P. Francis. CONMan: A Step Towards Network Manageability. In *Proc. of SIGCOMM*, 2007.
[14] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: realistic and controlled network experimentation. In *Proc. of SIGCOMM*, 2006.
[15] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a Routing Control Platform. In *Proc. of NSDI*, 2005.
[16] F. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. Andersen. Ditto - A System for Opportunistic Caching in Multi-hop Wireless Mesh Networks. In *Proc. of Mobicom*, 2008.
[17] N. Duffield and M. Grossglauser. Trajectory Sampling for Direct Traffic Observation. In *Proc. of SIGCOMM*, 2001.
[18] A. Greenberg, et al. A Clean Slate 4D Approach to Network Control and Management. *CCR*, 35(5), Oct. 2005.
[19] J. C. Mogul, Y. M. Chan, and T. Kelly. Design, implementation, and evaluation of duplicate transfer detection in HTTP . In *Proc. of NSDI*, 2004.
[20] J. W. Lockwood et al. NetFPGA - An Open Platform for Gigabit-rate Network Switching and Routing . In *Proc. IEEE MSE*, 2007.
[21] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. *SIGOPS Oper. Syst. Rev.*, 33(5):217–231, 1999.
[22] K. Park, S. Ihm, M. Bowman, and V. Pai. Supporting practical content-addressable caching with CZIP compression. In *Proc. of USENIX ATC*, 2007.
[23] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. of NSDI*, 2007.
[24] M. Rabin. Fingerprinting by random polynomials. Technical report, Harvard University, 1981. Technical Report, TR-15-81.
[25] M. Roughan et al. Experience in Measuring Internet Backbone Traffic Variability:Models, Metrics, Measurements and Meaning. In *ITC*, 2003.
[26] S. Rhea, K. Liang, and E. Brewer. Value-based web caching. In *Proc. of WWW*, 2003.
[27] A. Shaikh and A. Greenberg. OSPF Monitoring: Architecture, Design and Deployment Experience. In *Proc. of NSDI*, 2004.
[28] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. of SIGCOMM*, 2002.
[29] N. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. of SIGCOMM*, 2000.
[30] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An architecture for internet data transfer. In *Proc. of NSDI*, 2006.
[31] V. Sekar et al. cSamp: A System for Network-Wide Flow Monitoring. In *Proc. of NSDI*, 2008.
[32] A. Wolman et al. On the scale and performance of cooperative Web proxy caching. In *Proc. of SOSP*, 1999.