Effective Network Management via System-Wide Coordination and Optimization

Vyas Sekar

CMU-CS-10-137

Aug 2010

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Michael K. Reiter, Co-Chair
Hui Zhang, Co-Chair
David G. Andersen
Walter Willinger, AT&T Labs-Research

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant numbers ANI-0331653, CNS-0756998, and CNS-0433540, the U.S. Army Research Office under grant number DAAD190210389, and the International Collaboration for Advancing Security Technology. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Network Management, Traffic Monitoring, Redundancy Elimination, Network Security, Intrusion Detection

To anyone who believes in my abilities more than I ever will!

Abstract

As networked systems grow and traffic patterns evolve, management applications are increasing in complexity and functionality. To address the requirements of these management applications, equipment vendors and administrators today depend on incremental solutions that increase the complexity of network elements and deployment costs for operators. Despite this increased complexity and cost, the incremental nature of these solutions still leaves a significant gap between the policy objectives of system administrators and today's mechanisms. These challenges arise in several application contexts in different networking domains: ISPs, enterprise settings, and data centers.

Much of this disconnect arises from the narrow device-centric view of current solutions. Such piecemeal solutions are inefficient: network elements duplicate tasks and some locations become overloaded. Worse still, administrators struggle to retrofit their high-level goals within device-centric configurations. This dissertation argues for a clean-slate system-wide approach for resource management in large-scale networked systems based on three high-level principles: (1) systematic selection and placement of device-level primitives, (2) lightweight coordination mechanisms that enable different network elements to effectively complement one another, and (3) practical optimization models that capture operating constraints and policy objectives.

This dissertation demonstrates the benefits of this system-wide approach in three application contexts: (1) meeting fine-grained coverage and accuracy requirements in traffic monitoring, (2) implementing a redundancy elimination service to improve network performance, and (3) managing the deployment of intrusion detection and prevention systems.

Acknowledgments

First, I would like to thank my advisors Mike Reiter and Hui Zhang for their guidance, seemingly endless reserves of patience in tolerating my rather whimsical views at times, and flexibility in providing me the freedom to pursue interesting research directions. I have been and continue be amazed by Mike's incredible conscientiousness and attention to technical detail. Equally, I have been surprised on many occasions by Hui's insights to get to the root of a research question even with cursory discussions. I hope I have inherited (at least some fraction) of these characteristics from them during this time.

Second, I would like to acknowledge the support and confidence that my committee members Dave Andersen and Walter Willinger have shown for my work. Dave has always been ready to offer his time and act as a great sounding board for different ideas. I hope that some of his energy and enthusiasm have rubbed off on me in that process! Walter, from the outset when I first interacted with him as a rather apprehensive graduate student, has only had words of encouragement for me and my work. That support has served as a great morale booster when things weren't looking too good (read paper rejections). Even though they are not officially on my committee, Aditya Akella, Anupam Gupta, and Srinivasan Seshan have contributed significantly to the ideas presented here and to my overall development as a researcher. Anupam's help and guidance have shaped much of the theoretical and algorithmic work in this thesis. The work on SmartRE would not have been possible without Aditya's willingness to listen to my half-baked idea at Sigcomm 2008 and trusting me enough to pursue that direction! Dave, Walter, Aditya, Anupam, and Srini have been instrumental in ensuring that I always had several (proxy) advisors when Mike and Hui were not around. Thanks!

Over the years, I have had the opportunity to interact with several researchers and faculty who have mentored me in various capacities– Bill Aiello, Nick Duffield, Anna Gilbert, Ramana Kompella, Balachander Krishnamurthy, Dave Maltz, Kobus van der Merwe, Anees Shaikh, Oliver Spatscheck, and Yinglian Xie. Their advice and suggestions have been helpful both in the specific projects I worked with them and beyond.

I would also like to thank the Computer Science Department and the faculty for providing an ideal collegial atmosphere that shields students from unnecessary distractions and overheads. In particular, I should thank Sharon Burks, Deborah Cavlovich, Catherine Copetas, and Kathy McNiff for being incredibly resourceful–I don't think I have been able to come up with something that they did not know the answer to.

I have been extremely fortunate to meet and interact with several talented, interesting, and fun fellow graduate students: Mukesh Agrawal, Ashok Anand, Pranjal Awasthi, Narayanaswamy Balakrishnan, Sivaraman Balakrishnan, Amitabh Basu, Ashwin Bharambe, David Brumley, Haowen Chan, Michael Collins, Elisabeth Crawford, Debabrata Dash, Michael Dinitz, Fahad Dogar, Bin Fan, Jason Franklin, Rashmi Gangadhariah, Aditya Ganjam, Deepak Garg, Satashu Goel, Daniel Golovin, Vineet Goyal, Varun Gupta, Dongsu Han, Ningning Hu, Abhishek Jajoo, Himanshu Jain, Hetunandan Kamisetty, Neel Krishnaswami, Ravishankar Krishnaswamy, Avijit Kumar, Kaushik Lakshminarayanan, Franck Le, Pratyusa Manadhata, Amit Manjhi, Mahim Mishra, Srivatsan Narayanan, George Nychis, Mugizi Rwebangira, Jeff Pang, Sandeep Pandey, Bryan Parno, Swapnil Patil, Amar Phanishayee, Bodicherla Aditya Prakash, Apurva Samudra, Samir Sapra, Vivek Seshadri, Runting Shi, Suyash Shringarpure, Harshavardhan Simhadri, Ali Sinop, Srinath Sridhar, Matthew Streeter, Jimeng Sun, Michael Tschantz, Rangarajan Vasudevan, Vijay Vasudevan, Gaurav Veda, Shobha Venkataraman, Justin Weisz, Dan Wendlandt, Adam Wierman, Hong Yan, Noam Zeilberger, and Xin Zhang. Thank you for ensuring that there were very few, if any, dull moments over the years.

I would like to thank the Carnegie library of Pittsburgh, the CMU frisbee group, the CMU quiz club, the CMU cricket club, the Pittsburgh squash leagues, and the Pittsburgh Cricket Association for keeping me entertained and helping me retain my sanity.

If I do get asked sometime in the future about my time in grad school, here is a reminder for my future self to quote Albert Einstein:
Put your hand on a hot stove for a minute, and it seems like an hour. Sit with a pretty girl for an hour, and it seems like a minute. That's relativity.

Finally, I would like to thank my family for their willingness to put up with my idiosyncrasies and uncommunicativeness; and their unwavering support for what I chose to do. I have been lucky to have that luxury.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Network management challenges arise in domains such as ISPs, enterprise networks, and data centers. Each domain involves several management tasks such as traffic monitoring, security, and performance optimization. These management applications have natural high-level policy goals. For example, network operators may want: (1) good monitoring coverage in order to understand end-to-end traffic patterns for detecting anomalous patterns, (2) effective configurations for application acceleration services in order to provide good end-to-end performance for their customers, and (3) effective deployment of intrusion detection and prevention systems to detect and drop malicious traffic as efficiently as possible.

However, as networks and traffic patterns *evolve*, the set of management applications and the requirements of existing applications change as well. This implies the need for new functions, more fine-grained capabilities, and more scalable solutions to understand and adapt to these changes. To put the work presented in this thesis in perspective, we discuss some possible approaches available to network operators today to meet the growing demands of network management applications.

1.1 Current practice

To provide some context, we group the current approaches into four broad classes. As a first-order approximation, the functionality increases as we move along the spectrum from left to right in Figure 1.1. However, the cost of deploying the solutions also increases.

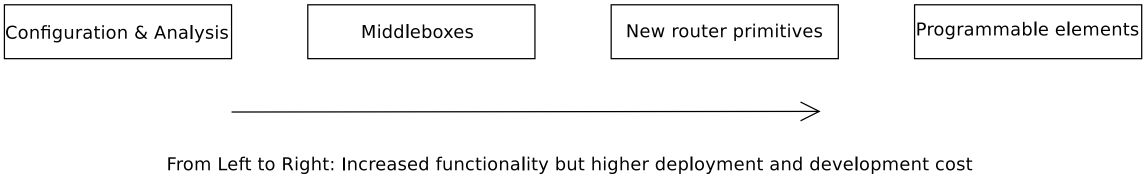

Configuration & Analysis
Middleboxes
New router primitives
Programmable elements
From Left to Right: Increased functionality but higher deployment and development cost

Figure 1.1: Qualitative comparison of four proposed classes of solutions to address the growing demands of network management applications.

Configuration and analysis:

The most common and easiest solution is for network operators to deploy techniques to work with existing router primitives. Router vendors (e.g., Cisco, Juniper) have provided in-built support in terms of configuration tools and router commands to support specific tasks like routing and access control. ISPs and enterprise networks also develop a suite of in-house analysis and configuration tools to simplify such functions. These include techniques that provide support for better traffic engineering and routing configurations (e.g., [66, 184, 43, 148, 68, 149]), techniques for inferring patterns of interesting traffic activity from existing measurement feeds (e.g., [99, 101, 86, 100, 49]), and accounting for potential biases in measurements (e.g., [58, 56, 57, 55]). Because such techniques do not require additional support from network elements, they are easy to develop and inexpensive to deploy. However, it might not always be possible to develop such tools (see [118, 40]).

Deploying middleboxes:

Often, management applications need new capabilities that might not be available on existing network elements. In such cases, network operators can deploy middleboxes developed by third party vendors. For example, these are commonly used for providing new security features (e.g., [1, 34, 12]) and for performance acceleration (e.g., [3, 8, 20, 19, 18, 7, 13]). Unfortunately, such solutions have a narrow scope and each new application context requires additional middleboxes. Further, because these are often proprietary solutions, they run the risk of becoming “black-boxes” to network operators.

New router primitives:

Further down the cost/development cycle is for router vendors to integrate the requisite functionality directly. There are several proposals for better monitoring algorithms (e.g., [97, 75, 168, 90]), in-depth forensic capabilities (e.g., [153, 119]), new diagnostic primitives (e.g., [54, 84]), more efficient data structures (e.g., [94, 111]), etc. While these avoid the problems of having too many middleboxes inside the network, they require router vendors and network managers to commit to a fixed set of capabilities without knowing if these will meet future application requirements.

Programmable network elements:

One possible solution to alleviate the concern of vendors and network operators to commit a priori to specific capabilities is an emerging class of solutions that use programmable network elements (e.g., [41, 116, 9, 46, 125, 180, 82]). However, there are still open issues with respect performance (can they operate at high traffic rates?) and ease of use (does it increase the configuration workload for operators?) that make the adoption of such solutions questionable.

1.2 Thesis Approach and Contributions

As the previous discussion shows, to meet the growing and evolving requirements of management applications, equipment vendors and administrators today depend on incremental solutions. This increases the complexity of network elements and deployment costs for operators. However, in spite this increased complexity and cost, there is still a significant gap between the policy objectives of system administrators and the capabilities provided by today's mechanisms. In particular, administrators have high-level *network-wide* objectives are often difficult to translate into router/device configurations that will meet the goals.

Our hypothesis, in the spirit of the recent proposals for centralized network management (e.g. [73, 43, 37, 69, 160, 60, 126, 125]), is that much of the disconnect between the goals of network operators and the tools available to them arises from the narrow *device-centric* view of current solutions. Such piecemeal solutions are inefficient: network elements duplicate tasks and some locations become overloaded. Worse still, administrators struggle to implement their high-level goals within device-centric configurations.

A key concern in achieving these high-level objectives is that the network elements (e.g., routers, middleboxes) that enable such management tasks have constraints on processing, memory, and storage capabilities. Even though network devices are becoming more powerful with advances in technology, the traffic workloads and usage patterns are scaling nearly as fast (if not faster) than these technology advances. Thus, these resource constraints are fundamental. As a result, these network management tasks can be broadly viewed as resource management problems in large networked systems.

Having cast the management tasks as resource management problems, we argue that the policy goals can be best achieved using a *network-wide approach* rather than a device-centric approach. The network-wide approach we advocate in this thesis is based on three guiding principles:

1. Systematic selection and placement of device-level primitives.
2. Lightweight coordination mechanisms that enable different network elements to effectively complement each other.
3. Practical optimization models that capture operating constraints and policy objectives, and produce close to optimal ways to configure the device-level primitives within their technological constraints.

At a high-level, we can think of this approach as being a middle ground between the *configuration and analysis* and the *new middleboxes and router primitives* approaches.[1] That is, we need (1) practical, efficient primitives that do not significantly increase the complexity and resource requirements of network elements and (2) frameworks to reason about how to configure/analyze these primitives to meet the high-level objectives of network operators.

In this dissertation, we demonstrate the benefits of this approach in three contexts:

- Flow-level traffic monitoring (Chapters 2–4).
- Performance acceleration using redundancy elimination (Chapter 5).
- Deploying intrusion detection and prevention systems (Chapter 6).

Next, we outline the key contributions in this dissertation for each application context.

1.2.1 Building a Robust Flow Monitoring Infrastructure

Networks use flow-level[2] measurements for traffic engineering, analyzing user applications, detecting attacks, and forensic analysis. Because of resource constraints, routers sample some of the traffic that pass through them to generate these measurements. Several studies have shown the limitations of current packet sampling based solutions (e.g., Cisco

[1] The specific techniques we outline are amenable to both middlebox and in-router deployments.

[2] A flow is a sequence of packets that have the same source/destination IP addresses, source/destination ports, and protocol that occur within a short span of time.

NetFlow) in providing the coverage and accuracy for many such tasks. This dissertation addresses two key challenges:
(1) How can we increase coverage (monitor many flows) and support fine-grained network-wide measurement objectives?
(2) How can a monitoring infrastructure be designed to support a wide spectrum of current and future management applications?

Improved Coverage using Coordinated Sampling:

To address the first challenge, we present a system called cSamp [147] in Chapter 2. cSamp combines three ideas: (1) flow sampling to increase flow coverage by avoiding the biases of existing monitoring primitives such as packet sampling, (2) coordination via hash-based packet selection to avoid redundant monitoring, and (3) network-wide optimization. These ideas have been proposed independently in other contexts. The key contribution in this dissertation is in their synergistic combination for flow monitoring.

Chapter 2 presents efficient algorithms for generating the optimal sampling strategies for very large ISPs. The chapter also outlines developed practical solutions to handle changes in traffic patterns and estimation errors in inputs. Across several topologies, cSamp achieves more than 2× the total coverage and 8× the performance for fine-grained objectives compared to existing monitoring solutions.

Coordination With Partial Information:

A natural unit of coordination in cSamp is an Origin-Destination pair (traffic with the same ingress and egress routers). Each router determines the OD-pair for each packet and decide whether or not to process it based on the monitoring responsibilities assigned to it for this OD-pair. However, determining the OD-pair may be difficult; e.g., due to routing table aggregation or multi-exit peers that advertise the same IP addresses at multiple points.

A practical challenge is to provide the benefits of a cSamp-like system when routers only work with local information [146]. In this case, maximizing the total flow coverage or fine-grained network-wide flow coverage goals become NP-hard. Consequently, a central algorithmic challenge is to design mechanisms that optimize these measures. We address this challenge in Chapter 3.

For the total coverage, we extend results from the theory of optimizing submodular functions [71] to achieve near-optimal performance. For other (non-submodular) objectives, we design practical heuristics for resource augmentation and partial deployment. In practice, only a few such upgrades are necessary to achieve near-optimal performance.

A Case for a Minimalist Flow Monitoring Architecture:

The inadequacy of current solutions for flow-level monitoring has led to the development of several application-specific algorithms specialized for monitoring tasks such as detecting "heavy-hitters" or large changes in traffic patterns [96, 102, 168, 176, 94]. However, these increase router complexity and require vendors and operators to commit to hardware capabilities without knowing if they are necessary or sufficient for future requirements. In this context, Chapter 4 revisits the case for a *minimalist* approach where each router implements a few generic primitives instead of several application-specific ones.

The case for a minimalist approach is motivated both by router technology trends and the structure of monitoring applications. First, the main bottleneck for monitoring is keeping counters in fast memory. By aggregating the memory used individually by several application-specific primitives, generic primitives can run with high enough sampling rates to support a wide spectrum of applications. Second, monitoring tasks fall into two broad classes that analyze either volume structure (e.g., traffic engineering) or communication structure (e.g., network security). Based on these insights, we present a candidate for a minimalist approach that combines flow sampling (to capture communication structure) [79] and sample-and-hold (for volume structure) [62], and use cSamp for network-wide management. We show using trace-driven evaluations that this combination performs as well or even better than several application-specific approaches. These results have both immediate benefits and long-term implications for both equipment vendors and network operators.

1.2.2 Improving Network Performance via Coordinated Redundancy Elimination

Redundancy Elimination (RE) to avoid duplicate delivery of content that is common across different network transfers can improve network performance and reduce bandwidth costs. Today, this is widely used on enterprise access links (e.g., [19]). This success has sparked interest in a network-wide RE service that would improve the effective capacity of ISPs and socializes the performance benefits to all end-to-end traffic [30]. However, extending single-vantage solutions to a network-wide service is challenging because RE involves expensive operations for indexing and caching content and compressing and reconstructing packets.

In Chapter 5, we present the design and implementation of SmartRE, an architecture that makes network-wide RE practical [31]. Unlike single-point solutions that tightly couple compression and reconstruction per link, SmartRE spatially decouples encoding and decoding operations to magnify the benefits of each such pair of operations. It uses hash-based coordination to divide caching tasks without needing complex cache consistency protocols. SmartRE's optimization framework models device constraints and traffic/redundancy patterns and optimizes the network's traffic engineering goals (e.g., reducing the overall traffic footprint). A prototype implementation shows that SmartRE is 4-5× better than current solutions and achieves close to 90% of the performance of an ideal unconstrained system.

1.2.3 Deploying Network Intrusion Detection and Prevention Systems

Network intrusion detection (NIDS) and prevention systems (NIPS) serve a critical role in detecting and dropping malicious traffic. The traditional view has treated these as single-vantage-point systems at the boundary between the internal network and the Internet. However, the limitations of traditional approaches for scaling such single-vantage-point solutions is increasingly evident in the context of: (1) large enterprise networks and in new domains such as data centers and (2) ISPs deploying ’‘in-network” defenses to provide security services to their customers. These trends require us to look beyond the traditional view of perimeter defense and provide network-wide visibility in deploying these systems [112].

In Chapter 6 we design a framework for partitioning NIDS functions across a network to ensure that no node is overloaded. This takes into account the resource footprints of each NIDS component, the capabilities of different nodes, and placement constraints specifying where each function is most effective (e.g., ingress nodes are best suited for scan detection). For NIPS, we show how to maximally reduce unwanted traffic without affecting the performance of benign traffic using specialized and power-intensive hardware with limited capacity (e.g., content addressable memories). We also present preliminary results extending techniques from online learning to combat strategic adversaries who try to evade these defenses.

1.3 Outline

The rest of this dissertation is organized as follows:

- Chapter 2 describes the design and implementation of cSamp.
- Chapter 3 shows how we can achieve the performance benefits of cSamp even when each router only has access to local routing information (versus OD-pair information for each packet).
- Chapter 4 presents the quantitative comparison between our minimalist architecture for monitoring and an architecture using application-specific algorithms on routers.
- Chapter 5 describes the design and implementation of SmartRE and evaluates it on real/synthetic packet traces.
- Chapter 6 shows how a system-wide approach can be used to manage a network-wide deployment of intrusion detection and prevention systems.
- We summarize the key contributions and the implications of the work presented here before highlighting some potential avenues for future work in Chapter 7.

Chapter 2

cSamp: A System for Network-Wide Flow Monitoring

Network operators routinely collect flow-level measurements to guide several network management applications. Traditionally, these measurements were used for customer accounting [55] and traffic engineering [66], which largely rely on aggregate traffic volume statistics. Today, however, flow monitoring assists several other critical network management tasks such as anomaly detection [99], identification of unwanted application traffic [49], and even forensic analysis [173], which need to identify and analyze as many distinct flows as possible. The main consequence of this trend is the increased need to obtain fine-grained flow measurements.

Yet, because of technological and resource constraints, modern routers cannot each record all packets or flows that pass through them. Instead, they rely on a variety of *sampling* techniques to selectively record as many packets as their CPU and memory resources allow. For example, most router vendors today implement uniform packet sampling (e.g., Netflow [48], sFlow [130]); each router independently selects a packet with a sampling probability (typically between 0.001 and 0.01) and aggregates the selected packets into flow records [124]. While sampling makes passive measurement technologically feasible (i.e., operate within the router constraints), the overall fidelity of flow-level measurements is reduced.

There is a disconnect between the increasing requirements of new network management applications and what current sampling techniques can provide. While router resources do scale with technological advances, it is unlikely that this disconnect will disappear entirely, as networks continue to scale as well. We observe that part of this disconnect stems from a router-centric view of current measurement solutions. In today's networks, routers record flow measurements completely *independently* of each other, thus leading to redundant flow measurements and inefficient use of router resources.

We argue that a centralized system that coordinates monitoring responsibilities across different routers can enhance the flow monitoring capabilities of a network. Moreover, such a centralized system simplifies the process of specifying and realizing network-wide flow measurement objectives. We describe Coordinated Sampling (cSamp), a system for flow monitoring within a single Autonomous System (AS). cSamp treats a network of routers *as a system to be managed in a coordinated fashion* to achieve specific measurement objectives. Our system consists of three design primitives:

- *Flow sampling*: cSamp uses flow sampling [79] instead of traditional packet sampling to avoid the sampling biases against small flows—a feature of particular importance to the new spectrum of security applications. At the same time, flow sampling preserves the fidelity of traffic volume estimation and thus the accuracy of traditional traffic engineering applications.

- *Hash-based coordination*: cSamp uses a hash-based selection primitive to eliminate duplicate measurements in the network. This allows different routers to monitor disjoint sets of flows without requiring explicit communication between routers, thus eliminating redundant and possibly ambiguous measurements across the network.

- *Network-wide optimization*: Finally, cSamp uses an optimization framework to specify and satisfy network-wide monitoring objectives while respecting router resource constraints. The output of this optimization is then translated into per-router *sampling manifests* that specify the set of flows that each router is required to record.

This chapter addresses several practical aspects in the design and implementation of cSamp. We present efficient algorithms for computing sampling manifests that scale to large tier-1 backbone networks with hundreds of routers. We provide practical solutions for handling multi-path routing and realistic changes in traffic patterns. We also implement a prototype using an off-the-shelf flow collection tool.

We demonstrate that cSamp is fast enough to respond in real time to realistic network dynamics. Using network-wide evaluations on the Emulab testbed, we also show that cSamp naturally balances the monitoring load across the network, thereby avoiding reporting hotspots. We evaluate the benefits of cSamp over a wide range of network topologies. cSamp observes more than twice as many flows compared with traditional uniform packet sampling, and is even more effective at achieving system-wide monitoring goals. For example, in the case of the minimum fractional flow coverage across all pairs of ingress-egress pairs, it provides significant improvement over other flow monitoring solutions.

ISPs can derive operational benefits from cSamp, as it reduces the bandwidth and the data management overheads caused by duplicated flow reports. We also show that cSamp is robust with respect to errors in input data and realistic traffic dynamics.

2.1 Related Work

The design of cSamp as a centrally managed network-wide monitoring system is inspired by recent trends in network management. In particular, recent work has demonstrated the benefits of a network-wide approach for traffic engineering [66, 184] and network diagnosis [99, 101, 108, 183]. Other recent proposals suggest that a centralized approach can significantly reduce management complexity and operating costs [37, 43, 73].

Despite the importance of network-wide flow monitoring, there have been few attempts in the past to design such systems. Most of the related work focuses on the single-router case and on providing incremental solutions to work around the limitations of uniform packet sampling. This includes work on adapting the packet sampling rate to changing traffic conditions [61, 89], tracking heavy-hitters [62, 186], obtaining better traffic estimates from sampled measurements [55, 79], reducing the overall amount of measurement traffic [57], and data streaming algorithms for specific applications [96, 102, 151].

Early work on network-wide monitoring has focused on the placement of monitors at appropriate locations to cover all routing paths using as few monitors as possible [47, 159, 128]. The authors show that such a formulation is NP-hard, and propose greedy approximation algorithms. In contrast, cSamp assumes a given set of monitoring locations along with their resource constraints and, therefore, is complementary to these approaches.

There are extensions to the monitor-placement problem to incorporate packet sampling [159]. Cantieni et al. also consider a similar problem [45]. While the optimization formulations in these share some structural similarity to our approach in Section 2.2.2, the specific contexts in which these formulations are applied are different. First, cSamp focuses on flow sampling as opposed to packet sampling. By using flow sampling, cSamp provides a generic flow measurement primitive that subsumes the specific traffic engineering applications that packet sampling (and the frameworks that rely on it) can support. Second, while it is reasonable to assume that the probability of a single packet being sampled multiple times across routers is negligible, this assumption is not valid in the context of flow-level monitoring. The probability of two routers sampling the same flow is high as flow sizes follow heavy-tailed distributions [53, 181]. Hence, cSamp uses mechanisms to coordinate routers to avoid duplicate flow reporting.

To reduce duplicate measurements, Sharma and Byers [150] suggest the use of Bloom filters. While minimizing redundant measurements is a common high-level theme between cSamp and their approach, our work differs on two significant fronts. First, cSamp allows network operators to directly specify and satisfy network-wide objectives, explicitly taking into account (possibly heterogeneous) resource constraints on routers, while their approach does not. Second, cSamp uses hash-based packet selection to implement coordination *without* explicit communication, while their approach requires every router to inform every other router about the set of flows it is monitoring.

Hash-based packet selection as a router-level primitive was suggested in Trajectory Sampling [54, 106]. Trajectory Sampling assigns all routers in the network a *common* hash range. Each router in the network records the passage for all packets that fall in this common hash range. The recorded trajectories of the selected packets are then used for applications such as fault diagnosis and for detecting routing anomalies. In contrast, cSamp uses hash-based selection to achieve the opposite functionality: it assigns *disjoint* hash ranges across multiple routers so that different routers monitor different flows.

2.2 Design

In this section, we present the design of the hash-based flow sampling primitive and the optimization engine used in cSamp. In the following discussion, we assume the common 5-tuple (*srcIP, dstIP, srcport, dstport, protocol*) definition of an IP flow.

2.2.1 Router Primitives

Hash-based flow sampling: Each router has a *sampling manifest* – a table of hash ranges indexed using a key. Upon receiving a packet, the router looks up the hash range using a key derived from the packet's header fields. It computes the hash of the packet's 5-tuple and samples the packet if the hash falls within the range obtained from the sampling manifest. In this case, the hash is used as an index into a table of flows that the router is currently monitoring. If the flow already exists in the table, it updates the byte and packet counters (and other statistics) for the flow. Otherwise it creates a new entry in the table.

The above approach implements flow sampling [79], since only those flows whose hash lies within the hash range are monitored. Essentially, we can treat the hash as a function that maps the input 5-tuple into a random value in the interval $[0, 1]$. Thus, the size of each hash range determines the flow sampling rate of the router for each category of flows in the sampling manifest.

Flow sampling requires flow table lookups for each packet; the flow table, therefore, needs to be implemented in fast SRAM. Prior work has shown that maintaining counters in SRAM is feasible in many situations [62]. Even if flow counters in SRAM are not feasible, it is easy to add a packet sampling stage prior to flow sampling to make DRAM implementations possible [89]. For simplicity, however, we assume that the counters can fit in SRAM for the rest of the chapter.

Coordination: If each router operates in isolation, i.e., independently sampling a subset of flows it observes, the resulting measurements from different routers are likely to contain duplicates. These duplicate measurements represent a waste of memory and reporting bandwidth on routers. In addition, processing duplicated flow reports incurs additional data management overheads.

Hash-based sampling enables a simple but powerful coordination strategy to avoid these duplicate measurements. Routers are configured to use the same hash function, but are assigned disjoint hash ranges so that the hash of any flow will match at most one router's hash range. The sets of flows sampled by different routers will therefore not overlap. Importantly, assigning non-overlapping hash ranges achieves coordination *without* explicit communication. Routers can thus achieve coordinated tasks without complex distributed protocols.

2.2.2 Network-wide Optimization

ISPs typically specify their network-wide goals in terms of *Origin-Destination (OD) pairs*, specified by the ingress and egress routers. To achieve flow monitoring goals specified in terms of OD-pairs, cSamp's optimization engine needs the traffic matrix (the number of flows per OD-pair) and routing information (the router-level path(s) per OD-pair), both of which are readily available to network operators [66, 184].

Assumptions and notation: We make two assumptions to simplify the discussion. First, we assume that the traffic matrix (number of IP flows per OD-pair) and routing information for the network are given and that these change infrequently. Second, we assume that each OD-pair has a single router-level path. We relax these assumptions in Section 2.2.4 and Section 2.2.5.

Each OD-pair OD_i ($i = 1, \ldots, M$) is characterized by its router-level path P_i and the number T_i of IP flows in a measurement interval (e.g., five minutes).

Each router R_j ($j = 1, \ldots, N$) is constrained by two resources: memory (per-flow counters in SRAM) and bandwidth (for reporting flow records). (Because we assume that the flow counters are stored in SRAM, we do not model packet processing constraints [62].) We abstract these into a single resource constraint L_j, the number of flows router R_j can record and report in a given measurement interval.

Let d_{ij} denote the fraction of the IP flows of OD_i that router R_j samples. If R_j does not lie on path P_i, then the variable d_{ij} will not appear in the formulation. For $i = 1, \ldots, M$, let C_i denote the fraction of flows on OD_i that is monitored.

Objective: We present a general framework that is flexible enough to support several possible flow monitoring objectives specified as (weighted) combinations of the different C_i values. As a concrete objective, we consider a hybrid measurement objective that maximizes the total flow-coverage across all OD-pairs ($\sum_i T_i \times C_i$) subject to ensuring the optimal minimum fractional coverage per OD-pair ($\mathbf{min}_i\{C_i\}$).

Problem $maxtotgivenfrac(\theta)$**:**

$$\text{Maximize} \sum_i (T_i \times C_i), \quad \text{subject to}$$

$$\forall j, \qquad \sum_{i:R_j \in P_i} (d_{ij} \times T_i) \leq L_j \tag{2.1}$$

$$\forall i, \qquad C_i = \sum_{j:R_j \in P_i} d_{ij} \tag{2.2}$$

$$\forall i,\ \forall j, \quad d_{ij} \geq 0 \tag{2.3}$$

$$\forall i, \qquad C_i \leq 1 \tag{2.4}$$

$$\forall i, \qquad C_i \geq \theta \tag{2.5}$$

We define a linear programming (LP) formulation that takes as a parameter θ, the desired minimum fractional coverage per OD-pair. Given θ, the LP maximizes the total flow coverage subject to ensuring that each OD-pair achieves a fractional coverage at least θ, and that each router operates within its load constraint.

We briefly explain each of the constraints. Eq (2.1) ensures that the number of flows that R_j is required to monitor does not exceed its resource constraint L_j. As we only consider sampling manifests in which the routers on P_i for OD_i will monitor distinct flows, Eq (2.2) says that the fraction of traffic of OD_i that has been covered is simply the sum of the fractional coverages d_{ij} of the different routers on P_i. Because each C_i represents a fractional quantity we have the natural upper bound $C_i \leq 1$ in Eq (2.4).

Since we want to guarantee that the fractional coverage on each OD-pair is greater than the desired minimum fractional coverage, we have the lower bound in Eq (2.5). Since the d_{ij} define fractional coverages, they are constrained to be in the range $[0, 1]$; however, the constraints in Eq (2.4) subsume the upper bound on each d_{ij} and we impose the non-zero constraints in Eq (2.3).

To maximize the total coverage subject to achieving the highest possible minimum fractional coverage, we use a two-step approach. First, we obtain the optimal minimum fractional coverage by considering the problem of maximizing $\mathbf{min}_i\{C_i\}$ subject to constraints Eqs (2.1)–(2.4). Next, the value $OptMinFrac$ obtained from this optimization is used as the input θ to the problem $maxtotgivenfrac$.

The solution to the above two-step procedure, $d^* = \langle d^*_{ij} \rangle_{1 \leq i \leq M, 1 \leq j \leq N}$ provides a sampling strategy that maximizes the total flow coverage subject to achieving the optimal minimum fractional coverage per OD-pair.

2.2.3 Sampling Manifests

The next step is to map the optimal solution into a *sampling manifest* for each router that specifies its monitoring responsibilities (Figure 2.1). The algorithm iterates over the M OD-pairs. For each OD_i, the variable $Range$ is advanced in each iteration (i.e., per router) by the fractional coverage d^*_{ij} provided by the current router (lines 4 and 5 in Figure 2.1). This ensures that routers on the path P_i for OD_i are assigned disjoint ranges. Thus, no flows are monitored redundantly.

Once a router has received its sampling manifest, it implements the algorithm shown in Figure 2.2. For each packet it observes, the router first identifies the OD-pair. Next, it computes a hash of the flow header (the IP 5-tuple) and checks if the hash value lies in the hash range assigned for this OD-pair. (The function HASH returns a value in the range $[0, 1]$). That is, the key used for looking up the hash range (c.f., Section 2.2.1) is the flow's OD-pair. Each router maintains a $Flowtable$ of the set of flows it is currently monitoring. If the packet has been selected, then the router either creates a new entry (if none exists) or updates the counters for the corresponding entry in the $Flowtable$.

2.2.4 Handling Inaccurate Traffic Matrices

The discussion so far assumed that the traffic matrices are known and fixed. Traffic matrices are typically obtained using estimation techniques (e.g., [184, 185]) that may have estimation errors.

```
GENERATESAMPLINGMANIFEST($d^* = \langle d^*_{ij} \rangle$)
    // $i$ ranges over all OD-pairs
    for $i = 1, \dots, M$ do
        $Range \leftarrow 0$
        // $j$ ranges over routers
        for $j = 1, \dots, N$ do
            $HashRange(i, j) \leftarrow [Range, Range + d^*_{ij})$
            $Range \leftarrow Range + d^*_{ij}$
    $\forall j, Manifest(j) \leftarrow \{\langle i, HashRange(i, j) \rangle | d^*_{ij} > 0\}$
```

Figure 2.1: Translating the optimal solution into a sampling manifest for each router

If the estimation errors are bounded, we scale the sampling strategy appropriately to ensure that the new scaled solution will operate within the router resource constraints and be near-optimal in comparison to an optimal solution for the true (but unknown) traffic matrix.

Suppose the estimation errors in the traffic matrix are bounded, i.e., if T_i and $\hat{T}_i$ denote the estimated and actual traffic for OD_i respectively, then $\forall i$, $T_i \in [\hat{T}_i(1-\epsilon), \hat{T}_i(1+\epsilon)]$. Here, ϵ quantifies how much the estimated traffic matrix (i.e., our input data) differs with respect to the true traffic matrix. Suppose the optimal sampling strategy for $\hat{T} = \langle \hat{T}_i \rangle_{1 \leq i \leq M}$ is $\hat{d} = \langle \hat{d}_{ij} \rangle_{1 \leq i \leq M, 1 \leq j \leq N}$, and that the optimal sampling strategy for $T = \langle T_i \rangle_{1 \leq i \leq M}$ is $d^* = \langle d^*_{ij} \rangle_{1 \leq i \leq M, 1 \leq j \leq N}$.

A sampling strategy d is T-feasible if it satisfies conditions Eqs (2.1)–(2.4) for T. For a T-feasible strategy d, let $\beta(d, T) = \mathbf{min}_i\{C_i\}$ denote the minimum fractional coverage, and let $\gamma(d, T) = \sum_i T_i \times C_i = \sum_i T_i \times (\sum_j d_{ij})$ denote the total flow coverage. Setting $d'_{ij} = d^*_{ij}(1-\epsilon)$, we can show that d' is $\hat{T}$-feasible, and

$$\begin{aligned}
\beta(d', \hat{T}) &\geq \left(\frac{1-\epsilon}{1+\epsilon}\right) \beta(\hat{d}, \hat{T}) \\
\gamma(d', \hat{T}) &\geq \left(\frac{1-\epsilon}{1+\epsilon}\right)^2 \gamma(\hat{d}, \hat{T}).
\end{aligned}$$

For example, with $\epsilon = 1\%$, using d' yields a worst case performance reduction of 2% in

```
CoordSampRouter($pkt$, $Manifest$)
    // $Manifest = \langle i, HashRange(i,j) \rangle$
    $OD \leftarrow$ GetODPairID($pkt$)
    // Hash returns a value in $[0,1]$
    $h_{pkt} \leftarrow$ Hash(FlowHeader($pkt$))
    if $h_{pkt} \in Hashrange(OD, j)$ then
        Create an entry in $Flowtable$ if none exists
        Update byte and packet counters for the entry
```

Figure 2.2: Algorithm to implement coordinated sampling on router R_j

the minimum fractional coverage and 4% in the total coverage with respect to the optimal strategy $\hat{d}$.

Proof sketch:

For clarity, we start by focusing on the minimum fractional coverage objective β. First, let us consider $\hat{d}$. The constraints this satisfies are

$$\forall j, \sum_i \hat{d}_{ij} \hat{T}_i \leq L_j$$

Since $\frac{T_i}{1+\epsilon} \leq \widehat{T}_i$, we also have the inequality,

$$\forall j, \sum_i \hat{d}_{ij} \frac{T_i}{1+\epsilon} \leq L_j$$

Now consider $d'' = \frac{\hat{d}}{(1+\epsilon)}$. By the above equation we note that d'' is feasible for T. Since d'' is feasible for T, the optimal value of d^* on T is related to $\beta(\hat{d}, \widehat{T})$ in the following manner:[1]

$$\beta(d^*, T) \geq \beta(d'', T) = \beta(d'', \widehat{T}) = \frac{\beta(\hat{d}, \widehat{T})}{(1+\epsilon)}$$

Now let us consider d^*. The constraints this satisfies are:

$$\forall j, \sum_i d^*_{ij} T_i \leq L_j$$

[1] Because β is only a function of the d values, $\beta(d'', T) = \beta(d'', \widehat{T})$.

Since $\widehat{T}_i(1-\epsilon) \le T_i$, we have the relationship

$$\forall j, \sum_i d^*_{ij}(1-\epsilon)\widehat{T}_i \le L_j$$

Now consider $d' = d^*(1-\epsilon)$. We observe that (a) d' is feasible for $\widehat{T}$, and (b) the value of the objective function of d' on T is $\beta(d', T) = \beta(d^*, T)(1-\epsilon)$.

So now we have $\beta(d', \widehat{T}) = \beta(d', T) = (1-\epsilon)\beta(d^*, T) \ge \frac{1-\epsilon}{1+\epsilon}\beta(\hat{d}, \widehat{T})$

Similarly, we can prove that $\gamma(d', \widehat{T}) \ge \frac{(1-\epsilon)^2}{(1+\epsilon)^2}\gamma(\hat{d}, \widehat{T})$ as follows. We want to find the relationship between $\gamma(d', \widehat{T})$ and $\gamma(\hat{d}, \widehat{T})$. First, as before let us consider $d'' = \frac{\hat{d}}{1+\epsilon}$ which is a feasible solution for T, and we can show that $\gamma(d, T) \ge \frac{1-\epsilon}{1+\epsilon}\gamma(\hat{d}, \widehat{T})$. Now by construction $\gamma(d', T) = (1-\epsilon)\gamma(d^*, T)$.

Let us consider $\gamma(d', \widehat{T})$. We have

$$\begin{aligned}
\gamma(d', \widehat{T}) &= \sum_i \widehat{T}_i C'_i \\
&\ge \sum_i \frac{T_i}{1+\epsilon} C'_i \\
&= \frac{\gamma(d', T)}{1+\epsilon} \\
&= \frac{1-\epsilon}{1+\epsilon}\gamma(d^*, T) \\
&\ge \frac{(1-\epsilon)^2}{(1+\epsilon)^2}\gamma(\hat{d}, \widehat{T})
\end{aligned}$$

2.2.5 Handling Multiple Paths per OD-pair

Next, we discuss a practical extension to incorporate multiple paths per OD-pair, for example using equal cost multi-path routing (ECMP) [51].[2]

Given the routing and topology information, we can obtain the multiple routing paths for each OD-pair and can compute the number of flows routed across each of the multiple paths. Then, we treat each of the different paths as a distinct logical OD-pair with different

[2] ECMP-enabled routers make forwarding decisions on a per-IP-flow rather than on a per-packet basis. Thus, we need not be concerned with multiple packets from a single flow traversing different router-level paths.

individual traffic demands. As an example, suppose OD_i has two paths P_i^1 and P_i^2. We treat P_i^1 and P_i^2 as independent OD-pairs with traffic values T_i^1 and T_i^2. This means that we introduce additional d_{ij} variables in the formulation. In this example, in Eq (2.1) we expand the term $d_{ij} \times T_i$ for router R_j to be $d_{ij}^1 \times T_i^1 + d_{ij}^2 \times T_i^2$ if R_j lies on both P_i^1 and P_i^2.

However, when we specify the objective function and the sampling manifests, we merge these logical OD-pairs. In the above example, we would specify the network-wide objectives in terms of the total coverage for the OD_i, $C_i = C_i^1 + C_i^2$. This merging procedure also applies to the sampling manifests. For example, suppose R_j occurs on the two paths in the above example, and the optimal solution has values d_{ij}^1 and d_{ij}^2 corresponding to P_i^1 and P_i^2. The sampling manifest simply specifies that R_j is responsible for a total fraction $d_{ij} = d_{ij}^1 + d_{ij}^2$ of the flows in OD_i.

2.3 System Architecture

Figure 2.3 depicts the overall architecture of cSamp. The central optimization engine computes and disseminates sampling manifests based on the traffic matrix and routing information continuously measured in the network. This engine also assigns an identifier to every OD-pair and propagates this information to the ingress routers. The ingress routers determine the OD-pair and mark packets with the identifier. Each router uses the OD-pair identifier and its sampling manifest to decide if it should record a specific flow. In order to handle traffic dynamics, the optimization engine recalculates the traffic matrix periodically based on the observed flow reports to generate and distribute new sampling manifests. Such a centralized approach is consistent with the operating model of modern ISPs, where operators push out router configuration files (e.g., routing tables, ACLs) and collect information from the routers.

To complete the description of the cSamp system, we describe the following mechanisms: 1) obtaining OD-pair information for packets; 2) responding to long- and short-term traffic dynamics; 3) managing memory resources on routers; 4) computing the sampling manifests efficiently; and 5) reacting to routing dynamics.

2.3.1 OD-pair Identification

Each router, on observing a packet, must identify the OD-pair to which the packet belongs. There are prior approaches to infer the OD-pair for a given packet based on the source and

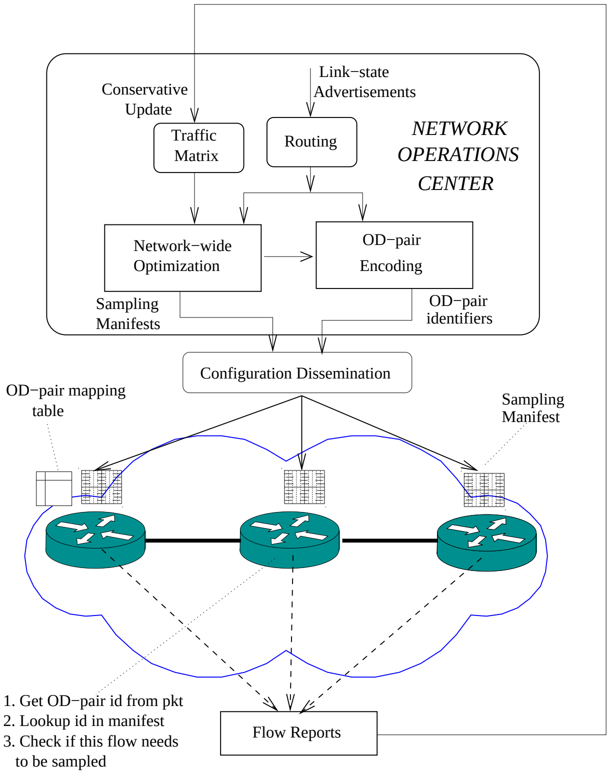


Figure 2.3: An overall view of the architecture of the cSamp system. The optimization engine uses up-to-date traffic and routing information to compute and disseminate sampling manifests to routers.

destination IP addresses and routing information [66]. However, such information may not be immediately discernible to interior routers from their routing tables due to prefix aggregation. Ingress routers are in a better position to identify the appropriate egress when a packet enters the network using such techniques. Thus the ingress routers mark each packet header with the OD-pair identifier. Interior routers can subsequently extract this information. In practice, the OD-pair identifier can either be added to the IP-header or to the MPLS label stack. Note that the multi-path extension (Section 2.2.5) does not impose additional work on the ingress routers for OD-pair identification. In both the single-path and multi-path cases, an ingress router only needs to determine the egress router and the identifier for the ingress-egress pair, and need not distinguish between the different paths for each ingress-egress pair.

The identifier can be added to the IP-id field in a manner similar to other proposals that rely on packet marking (e.g., [107, 143, 177]). This 16-bit field allows assigning a unique identifier to each OD-pair in a network with up to 256 border routers (and 65,536 OD-pairs), which suffices for medium-sized networks. For larger ISPs, we use an additional encoding step to assign identifiers to OD-pairs so that there are no conflicts in the assignments. For example, OD_i and $OD_{i'}$ can be assigned the same identifier if P_i and $P_{i'}$ do not traverse a common router (and the same interfaces on that router) or, if they do, the common router is not assigned logging responsibility for one of them. We formulate this notion of non-conflicting OD-pairs as a graph coloring problem, and run a greedy coloring algorithm on the resulting conflict graph. Using this extension, the approach scales to larger ISPs (e.g., needing fewer than 10 bits to encode all OD-pairs for a network with 300 border routers).

While the above approach to retrofit OD-pair identifiers within the IP header requires some work, it is easier to add the OD-pair identifier as a static label in the MPLS label stack. In this case, the space required to specify OD-pair identifiers is not a serious concern. In the next chapter, we relax this assumption and describe an alternative approach that does not require OD-pair identifiers.

2.3.2 Dealing with Traffic Dynamics

To ensure that the flow monitoring goals are achieved consistently over time, the optimization engine must be able to predict the traffic matrix to compute the sampling manifests. This prediction must take into account long-term variations in traffic matrices (e.g., diurnal trends), and also be able to respond to short-term dynamics (e.g., on the scale of a few minutes).

Long-term variations in traffic matrices typically arise from predictable time-of-day and day-of-week effects [140]. To handle these, we use historical traffic matrices as inputs to the optimization engine to compute the sampling strategy. For example, to compute the manifests for this week's Fri. 9am-10am period, we use the traffic matrix observed during the previous week's Fri. 9am-10am period.

The optimization engine also has to respond to less predictable short-term traffic variations. Using historical traffic matrices averaged over long periods (e.g., one week) runs the risk of *underfitting*; important structure present over shorter time scales is lost due to averaging. On the other hand, using historical traffic matrices over short periods (e.g., 5-minute intervals) may result in *overfitting*, unnecessarily incorporating details specific to the particular historical period in question.

To handle the long and short-term traffic dynamics, we take the following heuristic approach. Suppose we are interested in computing sampling manifests for every 5-minute interval for the Fri. 9am-10am period of the current week. To avoid overfitting, we do not use the traffic matrices observed during the corresponding 5-minute intervals that make up the previous week's Fri. 9am-10am period. Instead, we take the (hourly) traffic matrix for the previous week's Fri. 9am-10am period, divide it by 12 (the number of 5-minute segments per hour), and use the resulting traffic matrix T^{old} as input data for computing the manifests for the first 5-minute period. At the end of this period, we collect flow data from each router and obtain the traffic matrix T^{obs} from the collected flow reports. (If the fractional coverage for OD_i with the current sampling strategy is C_i and x_i sampled flows are reported, then $T_i^{obs} = \frac{x_i}{C_i}$, i.e., normalizing the number of sampled flows by the total flow sampling rate.)

Given the observed traffic matrix for the current measurement period T^{obs} and the historical traffic matrix T^{old}, a new traffic matrix is computed using a *conservative update* policy. The resulting traffic matrix T^{new} is used as the input for obtaining the manifests for the next 5-minute period.

The conservative update policy works as follows. First, we check if there are significant differences between the observed traffic matrix T^{obs} and the historical input data T^{old}. Let $\delta_i = \frac{|T_i^{obs} - T_i^{old}|}{T_i^{old}}$ denote the estimation error for OD_i. If δ_i exceeds a threshold Δ, then compute a new traffic matrix entry T_i^{new}, otherwise use T_i^{old}. If T_i^{obs} is greater than T_i^{old}, then set $T_i^{new} = T_i^{obs}$. If T_i^{obs} is smaller than T_i^{old}, we check the resource utilization of the routers currently responsible for monitoring OD_i. If all these routers have residual resources available, set $T_i^{new} = T_i^{obs}$; otherwise set $T_i^{new} = T_i^{old}$.

The rationale behind this conservative update heuristic is that if a router runs out of resources, it may result in underestimating the new traffic on OD-pairs for which it is responsible (i.e., T^{obs} is an under-estimate of the actual traffic matrix). Updating T^{new} with T^{obs} for such OD-pairs is likely to cause a recurrence of the same overflow condition in the next 5-minute period. Instead, we err on the side of overestimating the traffic for each OD-pair. This ensures that the information obtained for the next period is reliable and can help make a better decision when computing manifests for subsequent intervals.

The only caveat is that this policy may provide lower flow coverage since it overestimates the total traffic volume. Our evaluations with real traffic traces (Section 2.4.3) show that this performance penalty is low and the heuristic provides near-optimal traffic coverage.

2.3.3 Flow Records in SRAM

We assume that the flow table is maintained in (more expensive) SRAM. Thus, we need a compact representation of the flow record in memory, unlike Netflow [48] which maintains a 64-byte flow record in DRAM. We observe that the entire flow record (the IP 5-tuple, the OD-pair identifier, and counters) need not actually be maintained in SRAM; only the flow counters (for byte and packet counts) need to be in SRAM. Thus, we can offload most of the flow fields to DRAM and retain only those relevant to the online computation: a four byte flow-hash (for flowtable lookups) and 32-bit counters for packets and bytes, requiring only 12 bytes of SRAM per flow record. To further reduce the SRAM required, we can use techniques for maintaining counters using a combination of SRAM and DRAM [187]. We defer a discussion of handling router memory exhaustion to Section 2.5.

2.3.4 Computing the Optimal Solution

In order to respond in near-real time to network dynamics, computing and disseminating the sampling manifests should require at most a few seconds. Unfortunately, the simple two-step approach in Section 2.2.2 requires a few hundreds of seconds on large ISP topologies and thus does not scale, even with state of the art LP solvers like CPLEX. We discovered that the main bottleneck is the first step of solving the modified LP to find $OptMinFrac$.

To reduce the computation time we implement two optimizations. First, we use a binary search procedure to determine $OptMinFrac$. This was based on experimental evidence that solving the LP specified by $maxtotgivenfrac(\theta)$ for a given θ is faster than solving the LP to find $OptMinFrac$. Second, we use the insight that $maxtotgivenfrac(\theta)$ can be formulated as a special instance of a MAXFLOW problem [63]. These optimizations reduce the time needed to compute the optimal sampling strategy to at most eleven seconds even on large tier-1 ISPs with more than 300 routers.

Binary search: The main idea is to use binary search over the value of θ in the LP formulation $maxtotgivenfrac(\theta)$. The procedure (Figure 2.4) takes as input an error parameter ϵ and returns a feasible solution with a minimum fractional coverage θ^* with $OptMinFrac - \theta^* \leq \epsilon$. The search keeps track of θ_{lower}, the smallest feasible value known (initially set to zero), and θ_{upper}, the highest possible value (initially set to $\frac{\sum_j L_j}{\sum_i T_i}$). In each iteration, the lower and upper bounds are updated depending on whether the current value θ is feasible or not and the current value θ is updated to $\frac{\theta_{lower}+\theta_{upper}}{2}$. The search starts from $\theta = \theta_{upper}$, and stops if $\theta_{upper} - \theta_{lower} \leq \epsilon$, and returns $\theta^* = \theta_{lower}$ at this stopping point.

```
BinarySearch($\epsilon$, Solve)
    // $\epsilon$ is the additive approximation error
    // Solve solves the maxtotgivenfrac problem for a given $\theta$
    $\theta_{lower} \leftarrow 0$
    // The best possible solution is simply the ratio
    // of total resource available to total traffic
    $\theta_{upper} \leftarrow \frac{\sum_j L_j}{\sum_i T_i}$
    $currentgap \leftarrow \theta_{upper} - \theta_{lower}$
    $\theta_{current} \leftarrow \theta_{upper}$
    while ( $currentgap > \epsilon$ )
    do
        $\langle status, Solution \rangle \leftarrow$ Solve($\theta_{upper}$)
        if ($status =$ feasible)
        then
            $\theta_{lower} \leftarrow \theta_{current}$
        else
            $\theta_{upper} \leftarrow \theta_{current}$
        $\theta_{current} \leftarrow \frac{\theta_{upper} + \theta_{lower}}{2}$
        $currentgap \leftarrow \theta_{upper} - \theta_{lower}$
    Return $\langle \theta_{lower}, Solution \rangle$
```

Figure 2.4: Using the binary search optimization to find the optimal sampling strategy

Reformulation using MaxFlow: We formulate the LP $maxtotgivenfrac(\theta)$ as an equivalent MaxFlow problem. Specifically, we construct a variant of traditional MaxFlow problems that has additional lower-bound constraints on edge capacities. The intuition behind this optimization is that MaxFlow problems are typically more efficient to solve than general LPs.

We construct the following (directed) graph $G = \langle V, E \rangle$. The set of vertices in G is

$$V = \{source, sink\} \cup \{od_i\}_{1 \leq i \leq M} \cup \{r_j\}_{1 \leq j \leq N}$$

Each od_i in the above graph corresponds to OD-pair OD_i in the network and each r_j in the graph corresponds to router R_j in the network.

The set of edges is $E = E_1 \cup E_2 \cup E_3$, where

$$\begin{aligned} E_1 &= \{(source, od_i)\}_{1 \le i \le M} \\ E_2 &= \{(r_j, sink)\}_{1 \le j \le N} \\ E_3 &= \{(od_i, r_j)\}_{i,j: R_j \in P_i} \end{aligned}$$

Let $f(x, y)$ denote the flow on the edge $(x, y) \in E$, and let $UB(x, y)$ and $LB(x, y)$ denote the upper-bound and lower-bound on edge capacities in G. Our objective is to maximize the flow F from $source$ to $sink$ subject to the following constraints.

$$\forall x, \left(\sum_y f(x, y) - \sum_y f(y, x) \right) = \begin{cases} F & x = source \\ -F & x = sink \\ 0 & \text{otherwise} \end{cases}$$

We specify lower and upper bounds on the flow on each edge as:

$$\forall x, \forall y, LB(x, y) \le f(x, y) \le UB(x, y)$$

The upper-bounds on the edge capacities are: (i) the edges from the $source$ to od_i have a maximum capacity equal to T_i (the traffic for OD-pair OD_i), and (ii) the edges from each r_j to the $sink$ have a maximum capacity equal to L_j (resource available on each router R_j).

$$UB((x, y)) = \begin{cases} T_i & x = source, y = od_i \\ L_j & x = r_j, y = sink \\ \infty & \text{otherwise} \end{cases}$$

We introduce lower bounds only on the edges from the $source$ to each od_i, indicating that each OD_i should have a fractional flow coverage at least θ:

$$LB((x, y)) = \begin{cases} \theta \times T_i & x = source, y = od_i \\ 0 & \text{otherwise} \end{cases}$$

We use the binary search procedure discussed earlier, but use this MAXFLOW formulation to solve each iteration of the binary search instead of the LP formulation.

2.3.5 Handling Routing Changes

The cSamp system can receive real-time routing updates from a passive routing and topology monitor such as OSPF monitor [149]. Ideally, the optimization engine would recompute the sampling manifests for each routing update. However, recomputing and disseminating sampling manifests to all routers for each routing update is expensive. Instead, the optimization engine uses a snapshot of the routing and topology information at the beginning of every measurement interval to compute and disseminate manifests for the next interval. This ensures that all topology changes are handled within at most two measurement intervals.

To respond more quickly to routing changes, the optimization engine can *precompute* sampling manifests for different failure scenarios in a given measurement cycle. Thus, if a routing change occurs, an appropriate sampling manifest corresponding to this scenario is already available. This precomputation reduces the latency of adapting to a given routing change to less than one measurement interval. Since it takes only a few seconds (e.g., 7 seconds for 300 routers and 60,000 OD-pairs) to compute a manifest on one CPU (Section 2.4.1), we can precompute manifests for all single router/link failure scenarios with a moderate (4-5×) level of parallelism. While precomputing manifests for multiple failure scenarios is difficult, such scenarios are also relatively rare.

2.3.6 Prototype implementation

Optimization engine: Our implementation of the algorithms for computing sampling manifests (Section 2.3.4) consists of 1500 lines of C/C++ code using the CPLEX callable library. The implementation is optimized for repeated computations with small changes to the input parameters, in that it carries state from one solution over to the next. Solvers like CPLEX typically reach a solution more quickly when starting "close" to a solution than when starting from scratch. Moreover, the solutions that result tend to have fewer changes to the preceding solutions than would solutions computed from scratch, which enables reconfigured manifests to be deployed with fewer or smaller messages. We implement this optimization for both our binary search algorithm and when recomputing sampling manifests in response to traffic and routing dynamics.

Flow collection: We implemented a cSamp extension to the YAF flow collection tool [21]. Our choice was motivated by our familiarity with YAF, its simplicity of implementation, and because it is a reference implementation for the IETF IPFIX working group [10]. The extensions to YAF required 200 lines of additional code. The small code modification suggests that many current flow monitoring tools can be easily extended to realize the benefits of cSamp. In our implementation, we use the Bob hash function [4] recommended by Molina et al [121].

2.4 Evaluation

We divide our evaluation into three parts. First, we demonstrate that the centralized optimization engine and the individual flow collection processes in cSamp are scalable in Section 2.4.1. Second, we show the practical benefits that network operators can derive from cSamp in Section 2.4.2. Finally, in Section 2.4.3, we show that the system can effectively handle realistic traffic dynamics.

In our experiments, we compare the performance of different sampling algorithms at a PoP-level granularity, i.e., treating each PoP as a "router" in the network model. We use PoP-level network topologies from educational backbones (Internet2 and GÉANT) and tier-1 ISP backbone topologies inferred by Rocketfuel [157]. We construct OD-pairs by considering all possible pairs of PoPs and use shortest-path routing to compute the PoP-level path per OD-pair. To obtain the shortest paths, we use publicly available static IS-IS weights for Internet2 and GÉANT and inferred link weights [117] for Rocketfuel-based topologies.

Topology (AS#)	**PoPs**	**OD-pairs**	**Flows** $\times 10^6$	**Packets** $\times 10^6$
NTT (2914)	70	4900	51	204
Level3 (3356)	63	3969	46	196
Sprint (1239)	52	2704	37	148
Telstra (1221)	44	1936	32	128
Tiscali (3257)	41	1681	32	218
GÉANT	22	484	16	64
Internet2	11	121	8	32

Table 2.1: Parameters for the experiments

Due to the lack of publicly available traffic matrices and aggregate traffic estimates for commercial ISPs, we take the following approach. We use a baseline traffic volume of 8 million IP flows for Internet2 (per 5-minute interval).[3] For other topologies, we scale the total traffic by the number of PoPs in the topology (e.g., given that Internet2 has 11 PoPs, for Sprint with 52 PoPs the traffic is $\frac{52}{11} \times 8 = 37$ million flows). These values match reasonably well with traffic estimates reported for tier-1 ISPs. To model the

[3] The weekly aggregate traffic on Internet2 is roughly 175TB. Ignoring time-of-day effects, this translates into 0.08TB per 5-minute interval. Assuming an average flow size of 10KB, this translates into roughly 8 million flows.

structure of the traffic matrix, we first annotate PoP k with the population p_k of the city to which it is mapped. We then use a gravity model to obtain the traffic volume for each OD-pair [150, 139]. In particular, we assume that the total traffic between PoPs k and k' is proportional to $p_k \times p_{k'}$. We assume that flow size (number of packets) is Pareto-distributed, i.e., $\Pr(Flowsize > x \text{ packets}) = (\frac{c}{x})^\gamma, x \geq c$ with $\gamma = 1.8$ and $c = 4$ [170]. (We use these as representative values; our results are similar across a range of flow size parameters.) Table 2.1 summarizes our evaluation setup.

2.4.1 Microbenchmarks

In this section, we measure the performance of cSamp along two dimensions – the cost of computing sampling manifests and the router overhead.

AS	**PoP-level (secs)**		**Router-level (secs)**	
	Bin-LP	**Bin-MaxFlow**	**Bin-LP**	**Bin-MaxFlow**
NTT	0.53	0.16	44.5	10.9
Level3	0.27	0.10	24.6	7.1
Sprint	0.01	0.08	17.9	4.8
Telstra	0.09	0.03	9.6	2.2
Tiscali	0.11	0.03	9.4	2.2
GÉANT	0.03	0.01	2.3	0.3
Internet2	0.01	0.005	0.20	0.14

Table 2.2: Time (in seconds) to compute the optimal sampling manifest for both PoP- and router-level topologies. Bin-LP refers to the binary search procedure without the MaxFlow optimization.

Computing sampling manifests: Table 2.2 shows the time taken to compute the sampling manifests on an Intel Xeon 2.80 GHz CPU machine for different topologies. For every PoP-level topology we considered, our optimization framework generates sampling manifests within one second, even with the basic LP formulation. Using the MaxFlow formulation reduces this further. On the largest PoP-level topology, NTT, with 70 PoPs, it takes only 160 ms to compute the sampling manifests with this optimization.

We also consider augmented router-level topologies constructed from PoP-level topologies by assuming that each PoP has four edge routers and one core router, with router-level OD-pairs between every pair of edge routers. To obtain the router-level traffic matrix, we split the inter-PoP traffic uniformly across the router-level OD-pairs constituting each PoP-level OD-pair.

Even with $5\times$ as many routers and $16\times$ as many OD-pairs as the PoP-level topologies, the worst case computation time is less than 11 seconds with the MaxFlow optimization. These results show that cSamp can respond to network dynamics in near real-time, and that the optimization step is not a bottleneck.

Worst-case processing overhead: cSamp imposes extra processing overhead per router to look up the OD-pair identifier in a sampling manifest and to compute a hash over the packet header. To quantify this overhead, we compare the throughput (on multiple offline packet traces) of running YAF in full flow capture mode, and running YAF with cSamp configured to log every flow. Note that this configuration demonstrates the worst-case overhead because, in real deployments, a cSamp instance would need to compute hashes only for packets belonging to OD-pairs that have been assigned to it, and update flow counters only for the packets it has selected. Even with this worst-case configuration the overhead of cSamp is only 5% (not shown).

Network-wide evaluation using Emulab: We use Emulab [169] for a realistic network-wide evaluation of our prototype implementation. The test framework consists of support code that (a) sets up network topologies; (b) configures and runs YAF instances per "router"; (c) generates offline packet traces for a given traffic matrix; and (d) runs real-time tests using the `BitTwist`[4] packet replay engine with minor modifications. The only difference between the design in Section 2.2 and our Emulab setup is with respect to node configurations. In Section 2.2, sampling manifests are computed on a per-router basis, but YAF processes are instantiated on a per-interface basis. We map router-level manifests to interface-level manifests by assigning each router's responsibilities across its ingress interfaces. For example, if R_j is assigned the responsibility to log OD_i, then this responsibility is assigned to the YAF process instantiated on the ingress interface for P_i on R_j.

We configure cSamp in full-coverage mode, i.e., configured to capture all flows in the network. (In our formulation this means setting the router resources such that $OptMinFrac = 1$). We also consider the alternative full coverage solution where each ingress router is configured to capture all traffic on incoming interfaces. The metric we compare is the normalized throughput of each YAF instance running in the emulated network. Let the total number of packets sent through the interface (in a fixed interval of 300 seconds) on which the YAF process is instantiated be $pkts_{actual}$. Suppose the YAF instance was able to process only $pkts_{processed}$ packets in the same time interval. Then the normalized throughput

[4] `http://bittwist.sourceforge.net`

is defined as $\frac{pkts_{processed}}{pkts_{actual}}$. By definition, the normalized throughput can be at most 1.

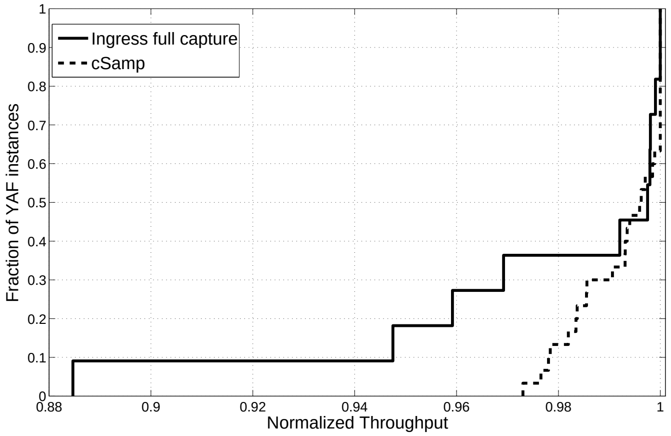


Figure 2.5: Comparing the CDF of normalized throughput per-interface across the entire network

Our test setup is unfair to cSamp for two reasons. First, with a PoP-level topology, every ingress router is also a core router. Thus, there are no interior routers on which the monitoring load can be distributed. Second, to emulate a router processing packets on each interface, we instantiate multiple YAF processes on a single-CPU Emulab `pc3000` node. In contrast, ingress flow capture needs exactly one process per Emulab node. In reality, this processing would be either parallelized in hardware (offloaded to individual linecards), or on multiple CPUs per YAF process even in software implementations, or across multiple routers in router-level topologies.

Figure 2.5 shows the distribution of the normalized throughput values of each YAF instance in the emulated network. Despite the disadvantageous setup, the normalized packet processing throughput of cSamp is higher. Given the 5% overhead due to hash computations mentioned before, this result might appear surprising. The better throughput of cSamp is due to two reasons. First, each per-interface YAF instance incurs per-packet flow processing overheads (look up flowtable, update counters, etc.) only for the subset of flows assigned to it. Second, we implement an optimization that first checks whether the OD-pair (identified from IP-id field) for the packet is present in its sampling manifest, and computes a hash only if there is an entry for this OD-pair. We also repeated the experiment by doubling the total traffic volume, i.e., using 16 million flows instead of 8 million flows. The difference between the normalized throughputs is similar in this case as well.

For example, the minimum throughput with ingress flow capture is only 85%, whereas for cSamp the minimum normalized throughput is 93% (not shown). These results show that by distributing responsibilities across the network, cSamp balances the monitoring load effectively.

2.4.2 Benefits of cSamp

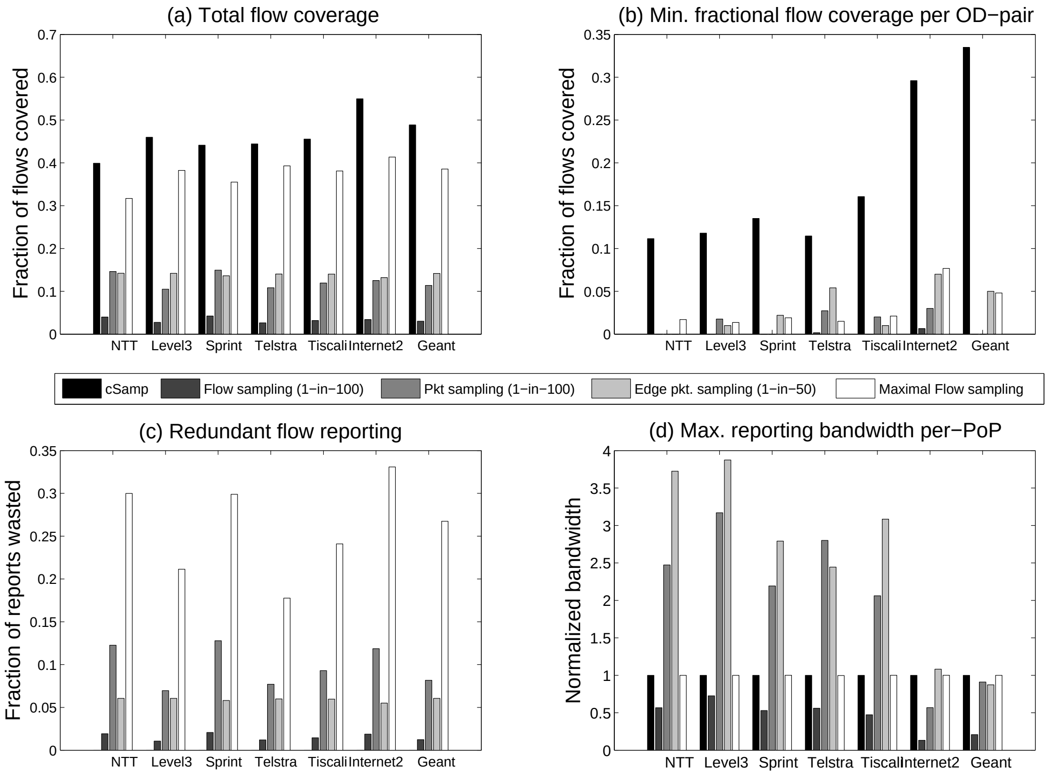


Figure 2.6: Comparing cSamp with packet sampling and hypothetical flow sampling approaches

It is difficult to scale our evaluations to larger topologies using Emulab. Therefore, we implemented a custom packet-level network simulator (roughly 2500 lines of C++)

to evaluate the performance of different sampling approaches. For all the sampling algorithms, the simulator uses the same network topology, OD traffic matrix, and IP flow-size distribution for consistent comparisons.

We consider two packet sampling alternatives: (i) uniform packet sampling with a sampling rate of 1-in-100 packets at all routers in the network, and (ii) uniform packet sampling at edge routers (this may reflect a feasible alternative for some ISPs [66]) with a packet sampling rate of 1-in-50 packets. We also consider two flow sampling variants: (iii) constant-rate flow sampling at all routers with a sampling rate of 1-in-100 flows, and (iv) maximal flow sampling in which the flow sampling rates are chosen such that each node maximally utilizes its available memory. In maximal flow sampling, the flow sampling rate for a router is $min(1, \frac{l}{t})$, where l is the number of flow records it is provisioned to hold and t is the total number of flows it observes. Both constant-rate and maximal flow sampling alternatives are hypothetical; there are no implementations of either available in routers today. We consider them along with cSamp to evaluate different intermediate solutions in the overall design space, with current packet sampling approaches at one end of the spectrum and cSamp at the other.

cSamp and the two flow sampling alternatives are constrained by the amount of SRAM on each router. We assume that each PoP in the network is provisioned to hold up to 400,000 flow records. Assuming roughly 5 routers per PoP, 10 interfaces per router, and 12 bytes per flow record, this requirement translates into $\frac{400{,}000 \times 12}{5*10} = 96$ KB SRAM per linecard, which is well within the 8 MB technology limit (in 2004) suggested by Varghese [167]. (The total SRAM per linecard is shared across multiple router functions, but it is reasonable to allocate 1% of the SRAM for flow monitoring.) Since packet sampling alternatives primarily operate in DRAM, we use the methodology suggested by Estan and Varghese [62] and impose no memory restrictions on the routers. By assuming that packet sampling operates under no memory constraints, we provide it the best possible flow coverage (i.e., we underestimate the benefits of cSamp).

Coverage benefits: Figure 2.6(a) compares the total flow coverage obtained with different sampling schemes for the various PoP-level topologies (Table 2.1). The total flow coverage of cSamp is 1.8-$3.3\times$ that of the uniform packet sampling approaches for all the topologies considered. Doubling the sampling rate for edge-based uniform packet sampling only marginally improves flow coverage over all-router uniform packet sampling. Among the two flow sampling alternatives, constant rate flow sampling uses the available memory resources inefficiently, and the flow coverage is 9-$16\times$ less than cSamp. Maximal flow sampling saturates the memory resources and is the closest in performance. Even in this case, cSamp provides 14-32% better flow coverage. While this represents only a modest gain over maximal flow sampling, Figures 2.6(b) and 2.6(c) show that maximal flow sampling suffers from poor minimum fractional coverage and increases the amount of redundancy in flow reporting.

Figure 2.6(b) compares the minimum fractional coverage per OD-pair. cSamp significantly outperforms all alternatives, including maximal flow sampling. This result shows a key strength of cSamp to achieve network-wide flow coverage objectives, which other alternatives fail to provide. In addition, the different topologies vary significantly in the minimum fractional coverage, in comparison to the total coverage. For example, the minimum fractional coverage for Internet2 and GÉANT is significantly higher than other ASes even though the traffic volumes in our simulations are scaled linearly with the number of PoPs. We attribute this to the unusually large diagonal and near-diagonal elements in a traffic matrix. For example, in the case of Telstra, the bias in the population distribution across PoPs is such that the top few densely populated PoPs (Sydney, Melbourne, and Los Angeles) account for more than 60% of the total traffic in the gravity-model based traffic matrix.

Reporting benefits: In Figure 2.6(c), we show the ratio of the number of *duplicate flow records* reported to the total number of distinct flow reports reported. The absence of cSamp in Figure 2.6(c) is because of the assignment of non-overlapping hash-ranges to avoid duplicate monitoring. Constant rate flow sampling has little duplication, but it provides very low flow coverage. Uniform packet sampling can result in up to 14% duplicate reports. Edge-based packet sampling can alleviate this waste to some extent by avoiding redundant reporting from transit routers. Maximal flow sampling incurs the largest amount of duplicate flow reports (as high as 33%).

Figure 2.6(d) shows the *maximum reporting bandwidth* across all PoPs. We normalize the reporting bandwidth by the bandwidth required for cSamp. The reporting bandwidth for cSamp and flow sampling is bounded by the amount of memory that the routers are provisioned with; memory relates directly to the number of flow-records that a router needs to export. The normalized load for uniform packet sampling can be as high as four. Thus cSamp has the added benefit of avoiding reporting hotspots unlike traditional packet sampling approaches.

Summary of benefits: cSamp outperforms traditional packet sampling approaches on all four metrics. Compared to constant rate flow sampling, cSamp is more efficient at using the available resources. While maximal flow sampling can provide reasonable total flow coverage, it has poor performance with respect to the minimum fractional flow coverage and duplicated flow reports. Also, as network operators provision routers to obtain greater flow coverage, this bandwidth overhead due to duplicate flow reports will increase.

2.4.3 Robustness Properties

To evaluate the robustness of our approach to realistic traffic changes, we consider a two-week snapshot (Dec 1–14, 2006) of (packet sampled) flow data from Internet2. We map each flow entry to the corresponding network ingress and egress points using the technique outlined by Feldmann et al. [66].[5] We assume that there are no routing changes in the network, and that the sampled flow records represent the actual traffic in the network. (Since cSamp does not suffer from flow size biases there is no need to renormalize the flow sizes by the packet sampling rate.) For this evaluation, we scale down the per-PoP memory to 50,000 flow records. (Due to packet sampling, the dataset contains fewer unique flows than the estimate in Table 2.1.)

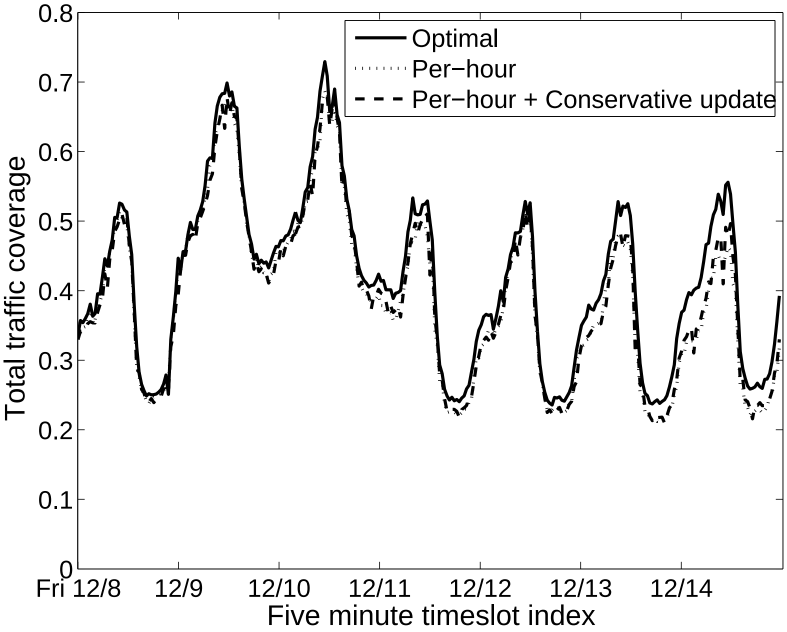


Figure 2.7: Comparing total traffic coverage with the conservative update heuristic vs. the optimal solution

Figure 2.7 compares the total flow coverage using our approach for handling traffic dynamics (Section 2.3.2) with the optimal total flow coverage (i.e., if we use the actual traffic matrix instead of the estimated traffic matrix to compute manifests). As expected,

[5] Since IP-addresses are anonymized by zero-ing out the last 11 bits, there is some ambiguity in egress resolution. However, this does not introduce a significant bias as less than 3% of the flows are affected.

the optimal flow coverage exhibits time-of-day and day-of-week effects. For example, during the weekend, the coverage is around 70% while on the weekdays the coverage is typically in the 20-50% range. The result confirms that relying on traffic matrices that are based on hourly averages from the previous week gives near-optimal total flow coverage and represents a time scale of practical interest that avoids both overfitting and underfitting (Section 2.3.2). Using more coarse-grained historical information (e.g., daily or weekly averages) gives sub-optimal coverage (not shown). Figure 2.7 also shows that even though the conservative update heuristic (Section 2.3.2) overestimates the traffic matrix, the performance penalty arising from this overestimation is negligible.

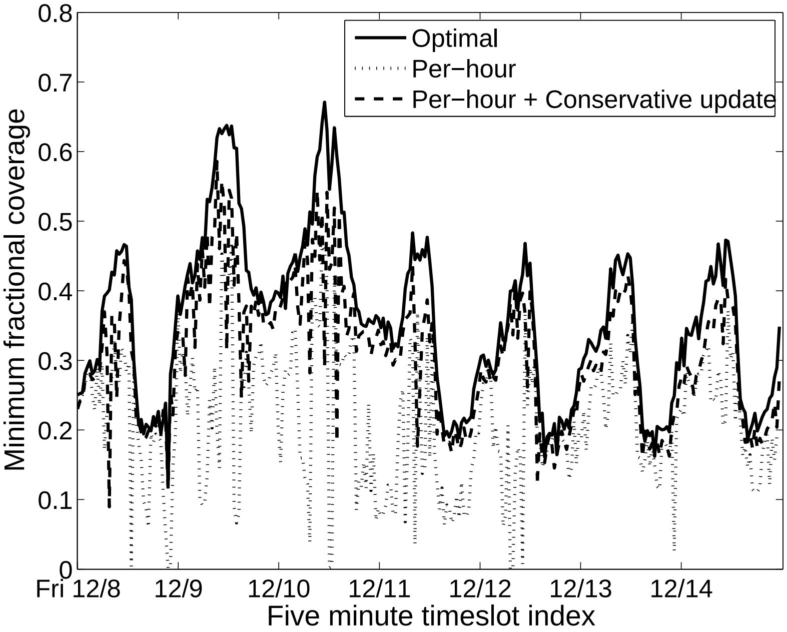


Figure 2.8: Comparing the minimum fractional coverage with the conservative update heuristic vs. the optimal solution

Figure 2.8 shows that using the per-hour historical estimates alone performs poorly compared to the optimal minimum fractional coverage. This is primarily because of short-term variations that the historical traffic matrices cannot account for. The conservative update heuristic significantly improves the performance in this case and achieves near-optimal performance. These results demonstrate that our approach of using per-hour historical traffic matrices combined with a conservative update heuristic is robust to realistic traffic dynamics.

2.5 Discussion

Router memory exhaustion: Despite factoring in the router memory constraints into the optimization framework, a router's flow memory might be exhausted due to traffic dynamics. In our current prototype, we choose not to evict flow records already in the flow memory, but instead stop creating new flow records until the end of the measurement cycle. The conservative update heuristic (Section 2.3.2) will ensure that the traffic demands for the particular OD-pairs that caused the discrepancy are updated appropriately in the next measurement cycle.

In general, however, more sophisticated eviction strategies might be required to prevent unfairness within a given measurement cycle under adversarial traffic conditions. For example, one such strategy could be to allocate the available flow memory across all OD-pairs in proportion to their hash ranges and evict flows only from those OD-pairs that exceed their allotted share. While this approach appears plausible at first glance, it has the side effect that traffic matrices will not be updated properly to reflect traffic dynamics. Thus, it is important to jointly devise the eviction and the traffic matrix update strategies to prevent short-term unfairness, handle potential adversarial traffic conditions, and minimize the error in estimating traffic matrices. We intend to pursue such strategies as part of future work.

Transient conditions inducing loss of flow coverage or duplication: A loss in flow coverage can occur if a router that has been assigned a hash range for an OD-pair no longer sees any traffic for that OD-pair due to a routing change. Routing changes will not cause any duplication if the OD-pair identifiers are globally unique. However, if we encode OD-pair identifiers without unique assignments (see Section 2.3.1), then routing changes could result in duplication due to OD-pair identifier aliasing. Also, due to differences in the time for new configurations to be disseminated to different routers, there is a small amount of time during which routers may be in inconsistent sampling configurations resulting in some duplication or loss.

Applications of cSamp: cSamp provides an efficient flow monitoring infrastructure that can aid and enable many new traffic monitoring applications (e.g., [49, 86, 99, 145, 173]). As an example application that can benefit from better flow coverage, we explored the possibility of uncovering botnet-like communication structure in the network [135]. We use flow-level data from Internet2 and inject 1,000 synthetically crafted single-packet flows into the original trace, simulating botnet command-and-control traffic. cSamp uncovers 12× (on average) more botnet flows compared to uniform packet sampling. We also confirmed that cSamp provides similar or better fidelity compared to uniform packet sampling for traditional traffic engineering applications such as traffic matrix estimation.

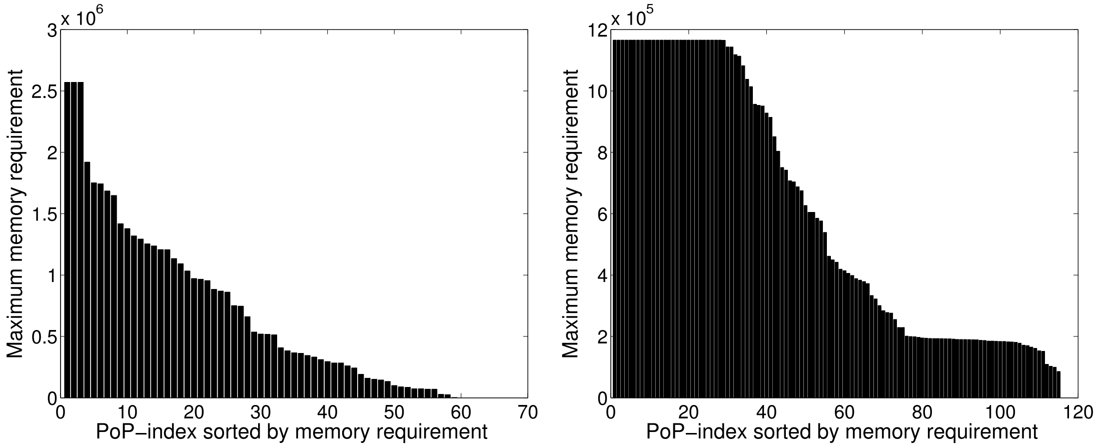
Network Provisioning: An alternative version of the network-wide formulation (Section 2.2.2) can be posed as a capacity provisioning problem; i.e., how should a network operator invest resources at routers (e.g., memory) to achieve a given target traffic coverage? To discuss such a "what-if" scenario, we use the notation and formulation from Section 2.2.2 and let θ_i denote the targeted fraction of traffic on OD-pair i to be monitored; that is,

$$\forall i,\ Coverage_i \geq \theta_i$$

The monitoring load L_j on router j is given by

$$\forall j,\ L_j = \sum_i d_{ij} \times T_i$$

and translates directly into the memory and reporting bandwidth that need to be provisioned on the router. It also reflects the cost incurred by the operators (e.g., memory upgrades on router hardware). We consider the following objective: minimizing the maximum load on any single router in the network.



(a) AS 3356

(b) AS 7018

Figure 2.9: Distribution of memory requirement across PoPs

Across the different PoP-level topologies we find that even with a target flow coverage of 90%, the maximum memory required per PoP is of the order of a 1-3 million traffic records. Assuming a 32-byte flow record, this translates into a maximum memory requirement of 90MB per-PoP, which is larger than the memory capacities on routers today, but not technologically inconceivable. This is promising in view of certain applications for which near-complete traffic coverage is desirable (e.g., forensic applications [173]). Figure 2.9 shows the distribution of the per-PoP memory requirement (in terms of number of flow records). We observe that the number of nodes that need very high provisioning is small. This is consistent with the observations in Section 2.4.2 regarding the structure of the underlying traffic matrix – dominant PoPs that carry a significant fraction of the traffic naturally demand better provisioning than smaller PoPs.

Minimizing Reconfigurations: An aspect of robustness that has not been addressed in this chapter concerns the number of reconfigurations under traffic dynamics. To reduce management complexity, network operators may prefer sampling manifests that are stable over time or require only a handful of reconfigurations in response to some of the typical events they expect. Here, a reconfiguration refers to either (i) a non-zero d_{ij} value becoming zero in the new sampling strategy recomputed after the traffic change, or (ii) a d_{ij} entry that was previously zero becoming non-zero in the new sampling strategy. As a preliminary exploration, we augmented the objective function with a reconfiguration cost term. The reconfiguration cost penalizes feasible sampling strategies that, while optimal otherwise, require a large number of reconfigurations when compared to the sampling strategy currently in use. Figure 2.10 shows the results of this preliminary exploration using data from Internet2. (We only show the results for day2 from week2; results for other days were similar). We see that the new sampling manifests are relatively stable throughout the 24-hour period and require only a small number of reconfigurations (less than 5% of the entries on average). Moreover, this added robustness feature is achieved with negligible loss in total flow coverage and minimum fractional coverage (0.5% and 3% respectively). These preliminary results are similar to prior work on configuring link weights in the context of intra-domain routing [32, 69]. One direction of future work is exploring this connection and developing strategies that are explicitly designed to have as few reconfigurations as possible.

2.6 Chapter Summary

Flow-level monitoring is an integral part of the suite of network management applications used by network operators today. Existing solutions, however, focus on incrementally improving single-router sampling algorithms and fail to meet the increasing demands for fine-grained flow-level measurements. To meet these growing demands, we argue the need for a system-wide rather than router-centric approach for flow monitoring.

We presented cSamp, a system that takes a network-wide approach to flow moni-

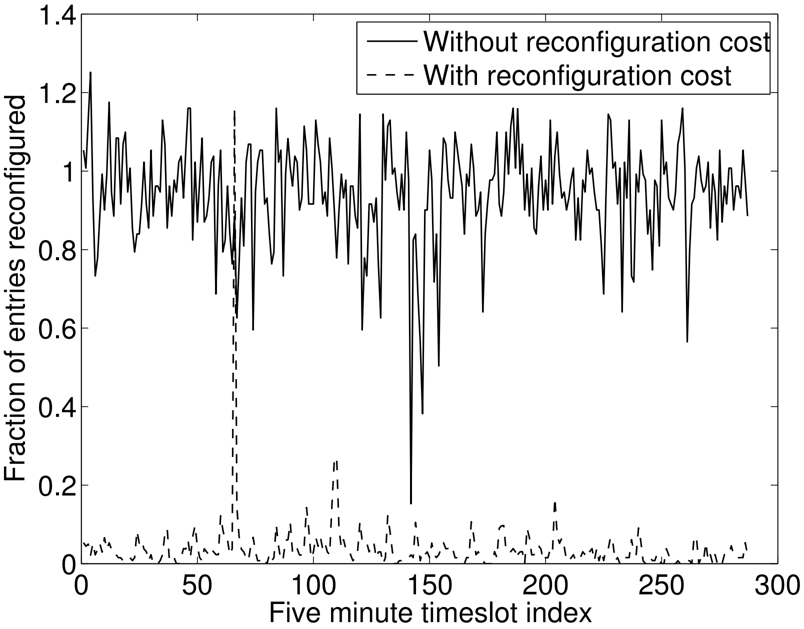


Figure 2.10: Effect of introducing reconfiguration cost to the formulation

toring. Compared to current solutions, cSamp provides higher flow coverage, achieves fine-grained network-wide flow coverage goals, efficiently leverages available monitoring capacity and minimizes redundant measurements, and naturally load balances responsibilities to avoid hotspots. We also demonstrated that our system is practical: it scales to large tier-1 backbone networks, it is robust to realistic network dynamics, and it provides a flexible framework to accommodate complex policies and objectives.

Chapter 3

Coordinated Sampling sans Origin-Destination Identifiers

In order to simplify the underlying algorithmic formulations, cSamp assumes that each router on receiving a packet can immediately ascertain the Origin-Destination (OD) pair for the packet, specified by the ingress and egress routers. However, due to prefix-aggregation and multi-exit peers, interior routers in the network cannot identify the OD-pair given just the source and destination IP addresses. Thus, cSamp imposes two requirements: (i) modifications to packet headers to carry OD-pair identifiers, and (ii) upgrades to border routers to compute the OD-pair identifiers [66] for each packet. Both modifications present significant deployment barriers for many ISPs. Thus, while cSamp is an elegant architecture that has the potential to improve flow monitoring, it does not have an immediate deployment path for ISPs today.

To address this impediment, in this chapter, we reformulate the problem of implementing a cSamp-like architecture when OD-pair identifiers are not available. The goal of such an architecture, to which we refer as cSamp-T[1], is to realize the benefits of cSamp and at the same time be immediately deployable. An immediate consequence of this reformulation is that the known algorithms [147] for efficiently maximizing either the total flow coverage or minimum fractional coverage across all OD-pairs, no longer apply. In fact, we show that these problems are NP-hard. Consequently, a central challenge is to develop algorithms for efficiently computing sampling strategies so as to optimize these measures, either exactly or approximately.

In this chapter, we present substantial progress toward meeting this challenge. For the

[1] cSamp-T denotes cSamp minus Tags for OD-pairs

measure of total flow coverage (total number of unique flows logged), we notice that the objective function is *submodular*. This is important because even though it is hard to find an exact optimal solution, we can implement efficient greedy algorithms with good approximation guarantees that leverage this submodularity property. We borrow and extend results from a rich theory of optimizing submodular functions subject to budget constraints (e.g., [71, 114, 91, 93]) to this specific application. We show that on realistic topologies, this approach yields near optimal total flow coverage.

The minimum fractional coverage objective (i.e., the minimum across all OD-pairs of the fraction of flows logged per OD-pair) is not submodular, however, and so does not inherit these approximation guarantees with a greedy approach [93]. Moreover, on realistic topologies the greedy approach performs poorly. So, in this case we turn to examining the additional resources needed in order to obtain good performance. We consider two practical scenarios for ISPs to alleviate this concern: (a) augmenting targeted routers with more memory resources and (b) incremental deployment of cSamp by upgrading a small subset of border routers with the functionality to compute OD-pair identifiers and add them into packet headers. Our results in this direction are promising: we show that a few such router upgrades can significantly boost the minimum fractional coverage obtained in realistic topologies.

cSamp-T thus makes cSamp-like solutions more immediately deployable by relaxing the dependence on the OD-pair identifiers. Further, it provides an incremental deployment path for ISPs to transition their flow monitoring infrastructures to cSamp, while in the interim partial deployment phase it provides performance comparable to cSamp. We also believe that many of the specific algorithmic techniques and heuristic extensions we develop here (e.g., applying results from the theory of submodular set maximization, intelligent resource provisioning, hybrid cSamp/cSamp-T deployment) can be more broadly applied to other aspects of network management and measurement.

3.1 Background and Motivation

Assumptions in cSamp: There are three main assumptions: (i) a centralized module for assigning router responsibilities that has access to routing and traffic matrices, (ii) routers implement hash-based flow sampling, and (iii) routers obtain OD-pair information from packet headers.

The first two assumptions are feasible within current technological and operational realities. First, centralization is viable if the router configurations are generated in a reasonable amount of time (say at most 1-2 minutes). Further, recent trends show that ISPs increasingly favor centralization of the network management functions [37, 73] and that routing and traffic matrices are typically already available [66, 184]. The second assumption that routers support hash-based flow sampling is also feasible within capabilities available today. The requirements on such hash functions are quite simple [153, 54] (e.g., no strong cryptographic guarantees) and thus they are amenable to fast hardware implementations [136]. Further, routers already implement hardware hash functions for other tasks. Flow sampling requires flow table lookups for each packet; the flow table, therefore, needs to be implemented in fast SRAM. Prior work has shown that maintaining such counters is feasible [62, 89]. For simplicity, cSamp assumes that the flow counters are maintained in SRAM and the amount of SRAM is the resource constraint that determines the number of flows a router can log.

The assumption that routers can obtain OD-pair identifiers simplifies cSamp's design and makes the optimization problem theoretically tractable. Specifically, Eq (2.2) implicitly assumes that the hash-ranges assigned to different routers for the same OD-pair are non-overlapping. Thus, the coverage of each OD-pair is simply the sum of the fractional coverages of the routers on the path. If OD-pair identifiers were not available, this would no longer hold. As we argue next, for many ISPs this assumption is not practical.

Challenges in OD-pair identification:

Given no additional information, a router needs to determine the ingress and egress routers (the OD-pair) for a given packet using only the packet's source and destination IP addresses and its local routing table. The feasibility of doing this depends on whether the ISP uses IP-based or MPLS-based forwarding. While IP forwarding is destination-based, MPLS can also take into account source information. However, we are unaware of deployments configured in this way, and we have confirmed that a large tier-one ISP's deployment of MPLS, for example, does not [29]. As such, here we restrict our attention to destination-based MPLS forwarding, which we believe to be the norm. Table 3.1 summarizes the feasibility of resolving the ingress and egress in these two scenarios.

Information to	Routing/Forwarding	
Resolve	IP (dest-based)	MPLS (dest-based)
Ingress	Difficult	Difficult
Egress	With some ambiguity	Possible

Table 3.1: Feasibility of resolving ingress and egress information using packet headers and local routing tables.

In both cases, resolving the ingress is nearly impossible. For example, in the case of traffic entering from a multi-exit peer (i.e., a neighboring AS with which an ISP peers at multiple peering points), source IP address and routing table information cannot determine the ingress from which the packet arrived. With MPLS, the egress can be resolved exactly; with IP the egress can be resolved within some ambiguity. Further, in IP forwarding, ingress/egress resolution may be additionally difficult due to prefix aggregation.

Due to the above challenges, cSamp assumes that ingress routers explicitly add OD-pair identifiers to packet headers. However, this leads to a practical deployment hurdle–it imposes additional processing on ingress routers to resolve/add the egress information and requires modifications to packet headers to carry OD-pair identifiers.

3.2 cSamp-T: Problem Statement

The above challenges in OD-pair identification bring us to the motivating question for our work: Can we implement a cSamp-like approach without requiring OD-pair identifiers? Intuitively, we want to specify each router's sampling manifest at a *coarser granularity* relying only on *local information* rather than the global OD-pair identifiers, while still achieving the coverage guarantees of cSamp. We call this new approach cSamp-T.

cSamp-T eliminates the need for ISPs to (a) upgrade border routers with additional intelligence for OD-pair identification, (b) modify packet headers to accommodate these identifiers, and (c) overhaul their routing infrastructures. Thus, cSamp-T makes the benefits of cSamp-like solutions available to network operators without incurring the overhead for OD-pair identification that cSamp imposes.

High-level approach: The key requirement in the cSamp-T approach is to only use *local* information at each router to specify the router's sampling responsibilities. The coverage of each OD-pair is obtained by "stitching" together the coverages provided by each router on the path.

Consider the example shown in Figure 3.1 with 2 ingresses, 2 egresses, and 4 OD-pairs P1–P4. The top-half shows a cSamp configuration; OD-pair identifiers are available and each router's responsibilities are in terms of hash-ranges per OD-pair and for each OD-pair the ranges on the routers on its path are non-overlapping.

The bottom-half of Figure 3.1 shows a scenario where routers cannot obtain OD-pairs. The sampling manifests are specified based on just local information; each router is assigned a *hash-range per router 3-tuple* consisting of the previous hop, current router, and

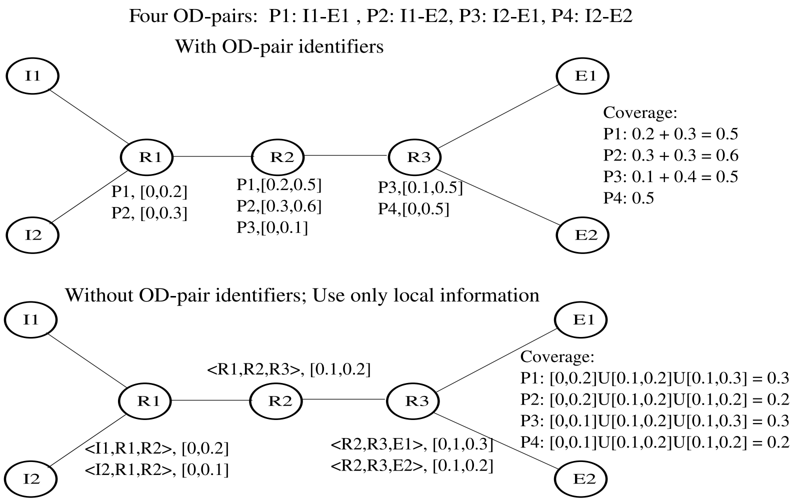


Figure 3.1: Example topology showing the intuition behind the cSamp-T approach

the next hop. Note that for each packet, a router can ascertain the previous hop and next hop just based on local information (e.g., the interface the packet arrives on and the next hop router determined by the routing table). The coverage for each OD-pair will then be the *union* of the ranges assigned to its constituent path-segments (the 3-tuples on each path in this example).

This example demonstrates two key differences between cSamp and cSamp-T. First, the sampling responsibilities are specified using locally available information rather than global OD-pair identifiers. Second, the coverage for each OD-pair is no longer simply the sum of the coverage of each router on the path; it is the union of the ranges assigned to the routers on the path.

Now, how do we assign sampling responsibilities in cSamp-T to maximize specific flow coverage objectives while operating within each router's resource constraints? The following sections present a formal framework to address this.

Problem Formulation for cSamp-T: We borrow two assumptions from cSamp: (a) sampling responsibilities are generated at a centralized module with access to routing and traffic matrices and (b) routers implement hash-based flow sampling using SRAM counters and the amount of SRAM is the main constraint on the number of flows a router can log. As discussed earlier, both are reasonable assumptions. Next, we discuss how the a centralized module can assign sampling responsibilities without OD-pair identifiers.

We first define the notion of a *SamplingSpec* to capture the granularity at which each router's sampling decisions are made. For the current discussion, the SamplingSpecs are three-tuples of router identifiers $\langle R_{j_1}, R_{j_2}, R_{j_3} \rangle$ that appear contiguously on some path in the network, and so in particular R_{j_1} and R_{j_3} are neighbors of R_{j_2}. Let a_k denote a generic SamplingSpec in our system.

The notation $a_k \in P_i$ captures the idea of a SamplingSpec being on the path P_i for OD_i.[2] For example, if the path P_i uses routers $\cdots, R_{j_1}, R_{j_2}, R_{j_3}, \cdots$ in that order, then the SamplingSpec $a = \langle R_{j_1}, R_{j_2}, R_{j_3} \rangle \in P_i$. This is a natural extension similar to the notion of a router R_j being on path P_i. We use $t_k = \sum_{i:a_k \in P_i} T_i$ to denote the total traffic that traverses a_k. Our framework maps SamplingSpecs to routers in a many-to-one fashion; we denote the set of SamplingSpecs assigned to R_j by $R_j.\mathsf{specs}$. In this way, R_j is assigned sampling responsibilities corresponding to all $a_k \in R_j.\mathsf{specs}$. In this chapter, if $a_k = \langle R_{j_1}, R_{j_2}, R_{j_3} \rangle$, then $a_k \in R_{j_2}.\mathsf{specs}$.

From the above discussion, it is clear that if $R_j.\mathsf{specs} \ni a_k$, then R_j is in a position to log (some or all) of the traffic on paths $P_i \ni a_k$. But which fraction should it log? To this end, if the entire traffic corresponding to a_k is mapped to points in the unit interval $[0, 1]$ (say, by hashing) then the router will be responsible for some subset of $[0, 1]$. In particular, we discretize $[0, 1]$ into $\frac{1}{\delta}$ equal-sized intervals of length δ $h_l = [(l-1)\delta, l\delta]$, and assign to a_k some of these δ-intervals.

We formalize this by creating a set of *SamplingAtom*s. A SamplingAtom is a pair $\langle a_k, h \rangle$, where a_k is a SamplingSpec and $h \subseteq [0, 1]$ is a "hash-range"—a subset of the unit interval of length δ. For any SamplingAtom, $g_{kl} = \langle a_k, h_l \rangle$, if $a_k \in R_j.\mathsf{specs}$, then router R_j will log the flows that traverse a_k such that the hash of the flow falls in h_l. We use $h(g_{kl})$ as a shortcut for the hash-range associated with g_{kl}.

Example: Figure 3.2 illustrates the above definitions with an example. $R3$ has three SamplingSpecs in the forward direction (and three similar SamplingSpecs in the reverse direction): $\langle R1, R3, R4 \rangle$, $\langle R1, R3, R2 \rangle$ and $\langle R2, R3, R4 \rangle$. $R3$ is assigned three SamplingAtoms, two for $\langle R1, R3, R4 \rangle$, one for $\langle R2, R3, R4 \rangle$, and none for $\langle R1, R3, R2 \rangle$. Say $\delta = 0.25$. Consider paths of the form $\{.., R1, R3, R4, ..\}$ (there may be many such paths). $R3$ will log all flows along these paths whose hashes fall either in the range $[0, 0.25]$ or $[0.75, 1]$, and flows along paths of the form $\{.., R2, R3, R4, ..\}$ such that the hash of the flow falls in the range $[0, 0.25]$.

[2] Since this notion of "on-path"-ness is quite general, our approach works for multi-path routing as well.

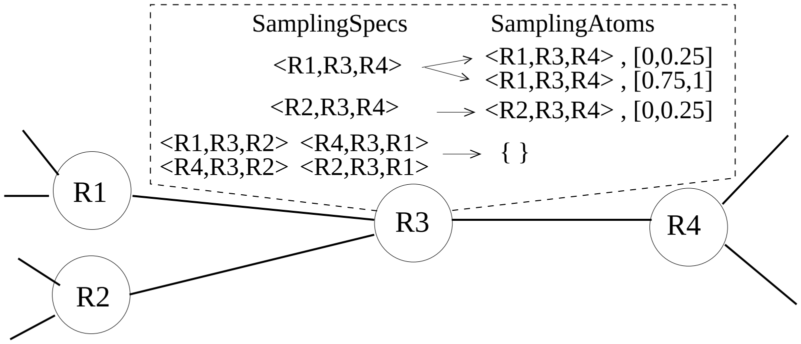


Figure 3.2: Example to illustrate the definitions showing the SamplingSpecs and assigned SamplingAtoms at router R3.

Notation	Explanation
M	Number of OD-pairs
N	Number of routers
OD_i	OD-pair i
C_i	Fraction of flows on OD-pair i covered
R_j	Router j
L_j	Available resources on R_j
$Load_j$	Total monitoring load on R_j
a_k	SamplingSpec k
$R_j.\mathsf{specs}$	set of SamplingSpecs on R_j
t_k	Total traffic traversing SamplingSpec a_k
g_{kl}	SamplingAtom l on a_k
$\widehat{g_{kl}}$	an assigned or selected SamplingAtom
$h(g_{kl})$	hash-range $\subseteq [0,1]$ in SamplingAtom g_{kl}

Table 3.2: Notation in the problem statement

Measures of Goodness: Given a set of assigned SamplingAtoms, $\{\widehat{g_{kl}}\}$, the *fractional coverage* for OD_i is as follows. The coverage due to one particular SamplingSpec $a_k \in P_i$ is $\cup_l \, h(\widehat{g_{kl}}) \subseteq [0,1]$, and hence the total coverage is

$$\text{coverage } C_i = \left|\bigcup_{a_k \in P_i} \bigcup_l \, h(\widehat{g_{kl}})\right| \tag{3.1}$$

Here, given an interval $S \subseteq [0,1]$, we use $|S|$ to denote the fraction of the unit interval covered by this subset. Note that the coverage for a path is the *union* of the assigned hash-ranges across all the constituent SamplingSpecs – if the *same* hash-range is assigned to several SamplingSpecs along a path, then the same set of flows gets sampled and we do not get any extra coverage.

The *monitoring load* on a router is given by summing, over all SamplingSpecs $a_k \in R_j.\mathsf{specs}$, the portion of the traffic through a_k that R_j logs:

$$Load_j = \sum_{a_k \in R_j.\mathsf{specs}} t_k \times |\bigcup_l \, h(\widehat{g_{kl}})| \tag{3.2}$$

Given the C_is for the various OD-pairs, the specific functions we consider are the *total traffic coverage* $f_{tot} = \sum_i T_i C_i$, and the *minimum fractional coverage* $f_{min} = \min_i C_i$. Formally, the goal of our algorithms is to obtain the set of assigned SamplingAtoms $\{\widehat{g_{kl}}\}$ such that we maximize either f_{tot} or f_{min}, while operating within the router resource constraints (i.e., $Load_j \leq L_j$ for all j). We choose these specific objective functions because of their use in cSamp [147]; our framework can accommodate a wider range of objective functions specified as convex combinations of the C_i values.

The maximization problem: We can rewrite the above maximization problems as follows. Consider a "ground set" $\mathcal{V}$ which contains as its elements all possible SamplingAtoms: i.e., $\mathcal{V} = \{\langle a_k, h_l \rangle$ *for all possible SamplingSpecs* a_k *and all* $\frac{1}{\delta}$ *hash-ranges* $h_l\}$. Suppose a subset $S \subseteq \mathcal{V}$ of these SamplingAtoms are chosen and assigned to their corresponding routers. These give us the fractional coverages defined by Eq (3.1) and router loads given by Eq (3.2). Now, f_{tot} or f_{min} can be viewed as functions from subsets of $\mathcal{V}$ to the reals. The problem is to select the *optimal* $S^* \subseteq \mathcal{V}$, satisfying $Load_j \leq L_j$, that maximizes f_{tot} or f_{min}.

Exact Solutions are Hard: Finding the optimal S^* to maximize f_{tot} or f_{min} subject to the load constraints on routers is NP-hard. The next section demonstrates the hardness via a reduction from the 3-SAT problem. Moreover, it is infeasible for practical system sizes. Specifically, we cast the problem into an integer linear programming (ILP) formulation by assigning 0-1 indicator variables for each g_{kl} to denote whether it is "assigned" or not. Even on the Internet2 topology with just 11 routers, the commercial solver `CPLEX` did not converge to a solution after a day. Because of this intractability of solving the problem exactly, we resort to greedy approximations. However, as we will see, our algorithms yield results that compare favorably to the original cSamp performance.

3.3 NP-hardness

First, we show that the decision version of the f_{tot} cSamp-T problem with $\delta = 1$ is NP-hard via a reduction from 3-SAT. Then, we extend the result and show the $\delta < 1$ case is at least as hard as the $\delta = 1$ case.

Hardness for $\delta = 1$: Let the variables in the 3-SAT problem be denoted by $x_1, \ldots, x_N$ and the clauses denoted by $C_1, \ldots, C_M$. Given an instance of a 3-SAT problem, we construct a cSamp-T problem as follows.

The set of "routers" in cSamp-T is $X \cup T \cup F \cup D$, where $X = \{X_1, \ldots, X_N\}$, $T = \{T_1, \ldots, T_N\}$, $F = \{F_1, \ldots, F_N\}$, and $D = \{D_1, \ldots, D_N\}$. Edges in the graph are $\{\langle T_j, X_j \rangle\} \cup \{\langle F_j, X_j \rangle\} \cup \{\langle X_j, D_j \rangle\} \cup \{\langle D_j, T_{j'} \}|j' > j\} \cup \{\langle D_j, F_{j'} \}|j' > j\}$.

Each SamplingSpec a_k can be one of the following: $\langle T_j, X_j, D_j \rangle$, $\langle F_j, X_j, D_j \rangle$, $\langle X_j, D_j, T_{j'} \rangle$, and $\langle X_j, D_j, F_{j'} \rangle$. There is exactly one SamplingAtom g_{k1} for each a_k and is equal to $\langle a_k, [0, 1] \rangle$. The budget constraints for D, F, and T nodes is zero. The only non-zero budgets are on the X nodes and $Budget(X_j)$ is equal to $\max($*#clauses with* x_j, *#clauses with* $\overline{x_j})$.

For each clause, we construct a OD-pair/path P_i as follows. Without loss of generality, let us assume that the clauses appear in sorted order of the variable indices. If the literal x_j appears in the clause, there is a sequence of vertices of the form T_j, X_j, D_j in the path. If the literal $\overline{x_j}$ appears in the clause, there is a sequence of vertices of the form F_j, X_j, D_j in the path. P_i has edges from D_j to the adjacent (in sorted order of indices) variable's $T_{j'}$ or $F_{j'}$ depending on whether $x_{j'}$ appears in positive or negative form in the clause. Each path has unit traffic, i.e. $\forall i$, $T_i = 1$.

Example: If $C_i = (x_j \vee \overline{x_k} \vee x_l)$, we create a path $P_i = (T_j, X_j, D_j, F_k, D_k, T_l, X_l)$ as shown in Fig. 3.3.

Claim: The decision problem of checking if $f_{tot} = M$ on the above cSamp-T problem is equivalent to solving the 3-SAT instance.

By construction, the only non-trivial SamplingAtoms are of the form $\langle \langle T_j, X_j, D_j \rangle, [0, 1] \rangle$

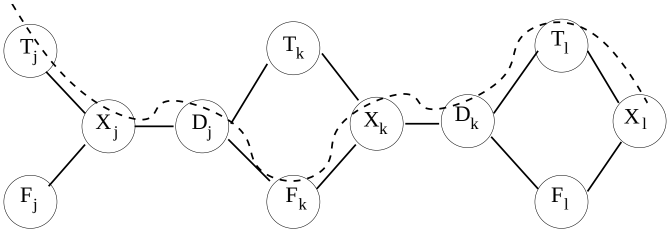


Figure 3.3: Example showing the path corresponding to the clause $C_i = (x_j \vee \overline{x_k} \vee x_l)$

or $\langle\langle F_j, X_j, D_j\rangle, [0, 1]\rangle$. Note that they specify all-or-nothing responsibilities. Due to the way the budgets are defined, for each X_j exactly one of $\langle T_j, X_j, D_j, [0, 1]\rangle$ or $\langle F_j, X_j, D_j, [0, 1]\rangle$ is "active"—in effect this corresponds to setting the variable x_j to be true or false. Hence, P_i has unit coverage in the solution of the cSamp-T instance if and only if there is at least one satisfied literal in clause C_i. Thus, checking if there is a satisfying assignment or not for the 3-SAT formula is equivalent to checking if the coverage $f_{tot} = M$ or $f_{tot} < M$. (In fact, it is also equivalent to checking if $f_{min} = 1$ or $f_{min} = 0$.) This proves the hardness for both cSamp-T problems of maximizing f_{tot} and f_{min} with $\delta = 1$.

Hardness with finer discretization: Given integer $d \geq 1$, the hardness for the $\delta = 1/d < 1$ case follows from a reduction from the $\delta = 1$ problem. Indeed, given an instance of the cSamp-T decision problem of deciding if $f_{tot} = M$ with $\delta = 1$, we construct the following instance with $\delta = 1/d$: we create $d - 1$ "dummy" vertices $V_1, \ldots, V_{d-1}$, and prepend these vertices to all paths P_i. We set the budgets on the dummy vertices to be $(1/d) \times M$. For every non-dummy vertex in the $\delta = 1$ problem, we scale the budgets by a factor $1/d$. By construction, $f_{tot} = M$ on the $\delta = 1/d$ problem if and only if $f_{tot} = M$ on the $\delta = 1$ problem; an analogous result holds for f_{min}. Thus, the $\delta = 1/d$ problems are at least as hard as the $\delta = 1$ problems.

3.4 Submodularity and Algorithms

Overview and Intuition: In the previous section, we saw that obtaining exact solutions for maximizing the total coverage or the minimum fractional coverage in the cSamp-T framework is hard. Fortunately, as we will see in the next sections, there are efficient practical algorithms to obtain the sampling strategies in cSamp-T. The key insight is that the coverage functions have a natural "submodularity" property (defined next) which allows us to apply powerful results from the theory of maximizing submodular set functions to our context. This is particularly promising, since submodularity implies that the greedy algorithm yields a constant-factor approximation [71].

More specifically, the coverage functions are “submodular” and the memory constraints at each router are “knapsack” constraints; our problem is then equivalent to the problem of maximizing submodular functions subject to knapsack constraints. We give theoretical bounds (Section 3.5) and also show that the greedy algorithms work very well in practice. We also give results for maximizing f_{min} using algorithms for max-min submodular maximization [93].

3.4.1 Submodularity

Definition: A function $F : 2^{\mathcal{V}} \rightarrow \Re$ mapping subsets of a ground set $\mathcal{V}$ to the reals is *submodular* if for all sets $S \subseteq S' \subseteq \mathcal{V}$, all elements $s \in \mathcal{V}$,

$$F(S \cup \{s\}) - F(S) \geq F(S' \cup \{s\}) - F(S')$$

i.e., *the marginal benefit obtained from adding s to a larger set is smaller* [71]. This captures the intuitive property of diminishing returns. The function F is *monotone (non-decreasing)* if $\forall S \subseteq S', F(S) \leq F(S')$.

Submodular set maximization: The goal is to pick a subset $S \subset \mathcal{V}$ maximizing $F(S)$; what makes this problem hard is that we also have a “budget” constraint of the form $c(S) \leq B$; i.e., given “costs” $c(s)$ for all $s \in \mathcal{V}$, the total cost $c(S) := \sum_{s \in S} c(s)$ of elements picked in set S cannot exceed the “budget” B. This submodular maximization problem is NP-hard [71], but good approximation guarantees are known. In particular, the algorithm specified in Figure 3.4 either greedily picks elements that give the greatest marginal benefit and do not violate the budget constraints, or greedily picks the elements that give the maximum marginal benefit *per unit element-cost* (depending on whether $cbflag$ is true or false), as long as the budget is not violated. It is well-known that the better of these two algorithms is a constant factor approximation algorithm [172].

3.4.2 Application to cSamp-T

It is easy to check the coverages C_i viewed as a functions from $\mathcal{V} = SamplingAtoms \rightarrow \Re$ are monotone submodular functions, and hence so is their weighted sum $f_{tot} = \sum T_i C_i$.

Budget constraints in cSamp-T: The budget constraints in cSamp-T come from the

```
SUBMODULARGREEDY($F, \mathcal{V}, cbflag, B$)
    // $F : 2^{\mathcal{V}} \rightarrow \Re$ submodular, $B$ is total budget
    // if $cbflag$ is true use benefit/cost instead of benefit
    $S \leftarrow \emptyset$
    while ($\exists s \in \mathcal{V} \setminus S : c(S \cup \{s\}) \leq B$) do
        for $s \in \mathcal{V} \setminus S$ do
            $norm \leftarrow ((cbflag = \textbf{true})\ ?\ c(s)\ :\ 1)$
            $\delta_s \leftarrow \frac{F(S \cup \{s\}) - F(S)}{norm}$
        $s^* \leftarrow \textbf{argmax}_{s \in \mathcal{V} \setminus S} \delta_s$
        $S \leftarrow S \cup \{s^*\}$
    return $\langle S, F(S) \rangle$
```

Figure 3.4: Basic greedy algorithm for maximizing a submodular function

bounds on router load. To model router load, we need a knapsack constraint $Load_j \leq L_j$ for each router R_j. A naive approach is to consider the cSamp-T problem as a submodular set maximization problem with multiple knapsack constraints. This naive approach yields a $O(N)$ approximation, where N is the number of routers. This is clearly undesirable, especially for large networks. Specifically, since each $SamplingAtom$ contributes to the load on exactly one router, this results in a collection of *non-overlapping* knapsack constraints. We call the resulting problem *submodular function maximization subject to partition-knapsack constraints*. (Each "partition" corresponds to a different router, and the "knapsack" comes from the load constraint for that router). In Section 3.5 we show that a modified greedy algorithm—an extension of one from Figure 3.4—gives a constant-factor approximation.

Maximizing f_{tot}: To match the theoretical guarantees [172] from Section 3.5, we run two separate invocations of the greedy algorithm—with and without the benefit-cost flag set to true, and return the solution with better performance. In practice, both have similar performance (Section 3.7.1).

Maximizing f_{min}: To maximize f_{min}, we need to go from one submodular function F to many submodular functions $F_1, F_2, \ldots, F_M$—in our case, these are the fractional coverages $C_1, \ldots, C_M$. The problem is now to pick $S \subseteq \mathcal{V}$ to (approximately) maximize $F^{\min}(S) = \min_i F_i(S)$, the *minimum* value across these different functions. This new

```
GreedyMaxMin($F_1, \ldots, F_M, \epsilon, \mathcal{V}, B, \gamma$)
    // Maximize $\mathbf{min}_i\{F_i\}$
    // $\forall i,\ F_i : 2^{\mathcal{V}} \rightarrow [0,1]$ is submodular
    $\tau_{lower} \leftarrow 0$; $\tau_{upper} \leftarrow 1$
    while $(\tau_{upper} - \tau_{lower} > \epsilon)$ do
        $\tau_{current} \leftarrow \frac{\tau_{upper} + \tau_{lower}}{2}$
        // Define the modified objective function
        $\forall i, \hat{F}_i \equiv \mathbf{min}(F_i, \tau_{current})$; $\hat{F} \equiv \sum_i \hat{F}_i$
        // Run greedy without budget constraints
        $B_{used} \leftarrow$ SubmodularGreedy($\hat{F}, \mathcal{V}, \mathbf{true}, \infty$)
        // Compare resource usage
        if MaxUsage($B_{used}, B$) $> \gamma$ then
            // $\tau_{current}$ is infeasible, reduce upper bound
            $\tau_{upper} \leftarrow \tau_{current}$
        else
            // $\tau_{current}$ is feasible, increase lower bound
            $\tau_{lower} \leftarrow \tau_{current}$
    Return $\tau_{lower}$
```

Figure 3.5: Maximizing the minimum of a set of submodular functions with resource augmentation

```
cSamp-T_Router($pkt, Manifest$)
    // $Manifest = \{\widehat{g_{kl}} = \langle a, h \rangle\}$
    $a \leftarrow$ GetSamplingSpec($pkt$)
    // $Ranges$ is a set of hash-range blocks
    $Ranges \leftarrow$ GetRanges($a$)
    // Hash returns a value in $[0, 1]$
    $h_{pkt} \leftarrow$ Hash(FlowHeader($pkt$))
    // Log if the hash value falls in one of the ranges
    if $h_{pkt} \in Ranges$ then
        Create an entry in $Flowtable$ if none exists
        Update byte and packet counters for the entry
```

Figure 3.6: Implementing cSamp-T on router R_j

function $F^{\min}$ is no longer submodular; indeed, obtaining any non-trivial approximation guarantee for this max-min optimization problem is NP-hard [93]. However, we can give an algorithm to maximize $F^{\min}$ when we are allowed to exceed the budget constraint by some factor. Formally, if S^* is an optimal set satisfying budget constraints, the algorithm in Figure 3.5 finds a set S with $F^{\min}(S) \geq F^{\min}(S^*) - \epsilon$ but which exceeds the budget constraints by a factor of γ, where $\gamma = O\big(\log(\frac{1}{\epsilon}\sum_{v \in \mathcal{V}} F_i(v))\big)$ [93].

The key idea is this: the modified objective function $\hat{F}_\tau = \sum_i^M \min(F_i, \tau)$ is submodular. For any τ, $\hat{F}_\tau$ has the property that its maximum value is $M \times \tau$ and at this maximum value $\forall i, F_i \geq \tau$. Running the greedy algorithm assuming no resource constraints always gives a set such that the actual resource usage at router R_j is at most $\gamma \times Load_j$. Notice that this holds for all τ, and in particular, for the optimal value $\tau^* = F^{\min}(S^*)$. Since the optimal τ^* is not known, the algorithm in Figure 3.5 uses binary search over τ.

Router algorithm: Given a solution to the problem of maximizing f_{tot} or f_{min}, Figure 3.6 shows each router's sampling algorithm. Note that the router no longer requires the OD-pair information for a packet; it only requires the coarser SamplingSpec information which can be immediately discerned using only the packet headers and other local information (e.g., what interface the packet arrives/leaves on). We allow for the $Ranges$ for each SamplingSpec to be a set of non-contiguous hash ranges; thus, the router samples the packet if the hash value falls in *any* of the ranges.

```
SubmodularGreedyLazy($F, \mathcal{V}, \mathit{cbflag}, B$)
    // $F : 2^{\mathcal{V}} \rightarrow \Re$ submodular, $B$ is total budget
    // if $\mathit{cbflag}$ is true use benefit-cost instead of benefit
    $S \leftarrow \emptyset$; $\delta_s \leftarrow \infty$ for all $s \in \mathcal{V}$
    while ($\exists s \in \mathcal{V} \setminus S : c(S \cup \{s\}) \leq B$) do
        $\forall s \in \mathcal{V} \setminus S, \mathit{active}_s \leftarrow \mathbf{false}$
        $\mathit{flag} \leftarrow \mathbf{true}$
        while $\mathit{flag}$ do
            $s^* \leftarrow \mathbf{argmax}_{s \in \mathcal{V} \setminus S} \delta_s$
            if $\mathit{active}_s$ then
                $S \leftarrow S \cup \{s^*\}$
                $\mathit{flag} \leftarrow \mathbf{false}$
            else
                $\mathit{norm} \leftarrow ((\mathit{cbflag} = \mathbf{true})\ ?\ c(s)\ :\ 1)$
                $\delta_{s^*} \leftarrow \frac{F(S \cup \{s^*\}) - F(S)}{\mathit{norm}}$
                $\mathit{active}_{s^*} \leftarrow \mathbf{true}$
    Return $\langle S, F(S) \rangle$
```

Figure 3.7: Greedy algorithm with lazy execution to reduce computation time

3.4.3 Practical Issues

Reducing computation time: The computation time of the algorithm of Figure 3.4 can be reduced by using the insight that for each element $s \in \mathcal{V}$, the marginal benefit obtained by picking s decreases monotonically across iterations of the greedy algorithm [114, 72]. Thus, we can use a *lazy evaluation* algorithm [114, 72]. The main intuition behind lazy evaluation (Figure 3.7) is that not all δ_s values need to be recomputed in Figure 3.4; only a smaller subset of that are likely to affect the choice of s^* need to be computed. We omit further details of this algorithm for brevity and refer the reader to the references [114, 72]. We can replace all instances of the procedure call SUBMODULARGREEDY with the lazy evaluation version in Figure 3.7. Section 3.7.2 shows that this reduces the computation time by more than an order of magnitude.

For very large topologies (>200 nodes), we use two additional optimizations: (1) In each greedy iteration, we evaluate the next k best choices in *parallel* using the OpenMP library [15]; (2) We use the cSamp solution for the minimum fractional coverage as the starting upper bound and avoid unnecessary iterations for the binary search in Figure 3.5.

Generalizing SamplingSpecs: We assumed that the SamplingSpecs are defined at the granularity of router three-tuples. Note, however, that the greedy algorithms and the per-router sampling algorithm are generic as they do not depend on SamplingSpecs being router three-tuples. Thus, we can generalize the algorithms and results to different notions of a SamplingSpec. For example, the SamplingSpecs can be router identifiers (in which case the router applies the same sampling decisions to every path passing through it), or router two-tuples (previous hop and current router), or incorporate IP-prefix information as well.

Practical issues in discretization: Section 3.2 defined discretization intervals δ such that $g_{kl} = \langle a_k, [(l-1)\delta, l\delta] \rangle$, for values $l \in \{1, \ldots, \frac{1}{\delta}\}$. There are two practical issues to note here. First, we can make the width δ arbitrarily small; there is a tradeoff between (potentially) better coverage vs. the time to compute the solution. In our evaluations, we fix $\delta = 0.02$ since we find that it works well in practice. Secondly, instead of considering $\frac{1}{\delta}$ disjoint intervals, we can also consider the $\frac{1}{\delta}^2$ hash-ranges of the form $[m\delta, (m+n)\delta]$ to make assignments as contiguous as possible. This increases the computation time quadratically without providing any additional coverage benefits. In practice, we avoid this and instead run a simple merge procedure (Section 3.7.3) to compress the sampling manifests.

3.5 Algorithmic Guarantees

Suppose we are given a monotone submodular function $F : U \to \Re$ with a partition $U = U_1 \uplus U_2 \uplus \ldots \uplus U_k$. The goal is to pick a set $S \subseteq U$ such that $|S \cap U_i| \leq 1$ and the value $F(S)$ is maximized. (In other words, we have a partition matroid on U and want to maximize F subject to S being independent in this matroid.) If we greedily pick elements $e_i \in U_i$ such that e_i is an element that α-approximately maximizes ($\alpha \leq 1$) the marginal benefit $F(\{e_1, e_2, \ldots, e_{i-1}, e_i\}) - F(\{e_1, e_2, \ldots, e_{i-1}\})$, then the benefit $F(\{e_1, \ldots, e_k\})$ is at least $\frac{\alpha}{2+\alpha}$ of the optimal benefit possible [44].

A different setting is when $F : U \to \Re$ is monotone submodular, we have a "budget" B, and each $e \in U$ has "size" c_e: the goal is to pick $S \subseteq U$ with $c(S) := \sum_{e \in S} c_e \leq B$. Consider two greedy algorithms: (a) the "cost/benefit" algorithm greedily keeps picking an element e which maximizes $\frac{\text{increase in } F}{c_e}$ and does not violate the budget, and (b) the "benefit" algorithm greedily keeps picking element e which maximizes the increase in F and does not violate the budget. One can show that the better of these two algorithms gets benefit at least 0.35 times the best possible [172]. In fact, an algorithm based on partial enumeration [161] gets an optimal $(1 - e^{-1})$-approximation.

We can combine these ideas to solve the problem of "submodular maximization subject to partition-knapsack constraints". Formally, we are given a monotone submodular function $F : \mathcal{V} \to \Re$, where there is a partition $\mathcal{V} = \mathcal{V}_1 \uplus \mathcal{V}_2 \uplus \ldots \uplus \mathcal{V}_k$. Each element $e \in \mathcal{V}$ has a size c_e, and each part $\mathcal{V}_i$ has a budget B_i: we want to pick a set $S \subseteq \mathcal{V}$ such that if $S_i = S \cap \mathcal{V}_i$, then the knapsack constraint $\sum_{e \in S_i} c_e \leq B_i$ is satisfied. For this problem, we can combine the two ideas above: imagine each valid knapsack of the elements in $\mathcal{V}_i$ to be a distinct element of the abstract set U_i, and $U = \uplus U_i$. Then considering the parts $\mathcal{V}_i$ one-by-one, and running the better of the benefit or cost-benefit algorithms on each part, results in the following result:

Theorem 1 *The simple greedy algorithm described above is a* $\frac{0.35}{2+0.35} \geq 0.148$*-approximation for the problem of submodular maximization subject to partition-knapsack constraints. Using a knapsack algorithm based on partial enumeration, we can get a* $\frac{e-1}{3e-1} \approx 0.406$*-approximation.*

As always, note that the results are *worst-case guarantees*: often these greedy algorithms for submodular maximization perform much better in practice.

The idea can be extended to the max-min problem. The algorithm for the max-min problem (subject to a cardinality constraint) from Krause et al. [93] uses an $(1 - e^{-1}) \approx 0.632$-approximation algorithm for submodular maximization only in a black-box fashion. Hence we can replace that algorithm by the above algorithm for submodular maximization subject to partition-knapsack constraints to get a bicriteria algorithm for the max-min problem that achieves optimal benefit, but exceeds each budget by a factor $O\big(\log(\sum_{e\in\mathcal{V}} F_i(v))\big)$—the fact that we are using an approximation guarantee of 0.148 instead of 0.632 only changes the constants in the big-oh.

3.6 Heuristic Extensions

While the theoretical guarantees for f_{tot} are encouraging, achieving good performance for f_{min} is less promising. The theoretical results suggests that the resource augmentation γ required to obtain any non-trivial guarantee is quite high.

In this section, we consider three practical extensions to improve the performance for f_{min}. The first extension uses a targeted provisioning heuristic to use fewer resources in aggregate. The second extension evaluates an incremental deployment scenario where a small subset of ingress routers can be upgraded to add OD-pair identifiers. We present these in the specific context of the f_{min} objective. However, these two techniques we develop for targeted provisioning and partial marking can be more generally applied to other network-wide objectives where the greedy algorithm performs poorly. We also consider an alternative submodular objective function for getting better performance for f_{min}.

3.6.1 Intelligent Provisioning

$$\text{Maximize } \mathbf{min}_i\ C_i, \text{ subject to}$$

$$\forall j, \textstyle\sum_{k:a_k\in R_j.\mathsf{specs}} u_k \times t_k \leq L_j \tag{3.3}$$

$$\textstyle\sum_j L_j \leq Budget \tag{3.4}$$

$$\forall j, LB_j \leq L_j \leq UB_j \tag{3.5}$$

$$\forall i, C_i = \textstyle\sum_{k:a_k\in P_i} u_k \tag{3.6}$$

$$\forall k, u_k \geq 0 \tag{3.7}$$

$$\forall i, C_i \leq 1 \tag{3.8}$$

The theoretical bounds from the previous section assume that each router in the network is uniformly given γ times more resources. In practice, this may be quite excessive since it might be very expensive to add γ times more SRAM capacity to each router. An interesting question is whether it is possible get better performance if we can add more memory on routers intelligently – instead of upgrading all routers, we seek to augment a smaller subset of routers and still get similar performance. The rationale behind the approach is that it may suffice to upgrade a small number of heavily loaded routers.

Problem $provisioning$**:** To address this question, we consider the above provisioning problem. The network operator specifies a total budget of memory resources to be distributed across different routers (e.g., defined by a total monetary budget and the cost of SRAM). Each router R_j has a lower bound (LB_j) for the default memory configuration and a technology upper bound (UB_j) on the maximum amount of memory that can be provisioned. (There are natural technological limits on the amount of fast SRAM that can be added to linecards [167].) The inputs to the problem are the total memory budget $Budget$, LB_j, and UB_j. The output is the specific allocation of resources to routers to optimize f_{min}.

However, it is difficult to model the coverage C_i of each OD-pair provided by the greedy algorithm under a given set of resources. Thus, we make a simplifying assumption that the hash ranges (represented by the variables u) allocated across the different SamplingSpecs on a given path are mutually non-overlapping. This allows us to model C_i as simply the sum of the ranges u_k in (3.6). Under this assumption, the resource provisioning problem can be solved as a linear program $provisioning$. While this is not optimal compared to faithfully modeling the C_i as the union of the ranges, this is a reasonable assumption since our goal is to obtain general guidelines for resource provisioning. As we will see in Section 3.7.4, this heuristic works well in practice.

There are two steps to the intelligent provisioning heuristic. The first step solves the LP $provisioning$. Next, given the resource allocation output by $provisioning$, we run the greedy algorithm in Figure 3.5 with $\gamma = 1$ to ensure that we are strictly within the resource constraints.

Adding a variance term to the objective: In practice, we find that it is useful to add a variance term to the objective function. We modify the above objective function $\min_i C_i$ to be $\{\min_i C_i\} - g(\{L_j^2\})$, where g is a function of the second-moments of the L values. The negative term denotes that our intent is to ***minimize*** the variance across the L values (with appropriate normalization to ensure that the variance term and the coverage term do not have wildly different magnitudes). Among the different configurations that maximize $\min_i C_i$, the goal is to pick the configuration that distributes the resources most uniformly across the routers. This offsets two potential undesirable effects. First, the LP solver may not necessarily use all the available resources to achieve the optimal minimum fractional coverage. Second, the LP solution may result in a skewed resource allocation which may be undesirable for the greedy algorithm and less robust to changes in traffic or routing inputs. The variance term forces the optimization solver (now a quadratic program instead of a LP) to use up the available resources efficiently and also reduces the skew. While this works well for most common cases, it may not prevent skewed allocations when $\beta >> \gamma$.

3.6.2 Partial OD-pair identification

Next, we consider a scenario in which a network operator can choose to upgrade some border routers. For example, this can be achieved using a software update to the router or by adding a simple two-port middlebox (using a software switch running on commodity hardware [122] or using FPGA [82]) that processes each packet, modifies the header, and forwards them to the router. These few upgraded nodes (routers or router plus middlebox) then have the capabilities to identify the OD-pairs and add the identifiers to packet headers. We assume that all routers run both cSamp and cSamp-T sampling algorithms – i.e., a router logs a flow if the hash of the flow falls in a hash-range corresponding *either* to the OD-pair or the SamplingSpec for the packet.

Problem $enabledODs(\theta, \mathcal{P}_e)$**:**

$$\text{Minimize } \sum_j L_j, \text{ subject to}$$

$$\forall j, \textstyle\sum_{i \in \mathcal{P}_e : R_j \in P_i} (d_{ij} \times T_i) \leq L_j \tag{3.9}$$

$$\forall i \in \mathcal{P}_e, C_i = \textstyle\sum_{j : R_j \in P_i} d_{ij} \tag{3.10}$$

$$\forall i \in \mathcal{P}_e, \ \forall j, \ d_{ij} \geq 0 \tag{3.11}$$

$$\forall i \in \mathcal{P}_e, \theta \leq C_i \leq 1 \tag{3.12}$$

Let $\mathcal{P}_e$ denote the set of "enabled" OD-pairs whose packets carry OD-pair identifiers and let $\mathcal{P}$ denote the set of all OD-pairs. We compute the maximum minimum fractional coverage using a binary search over τ. The key difference between the new algorithm and Figure 3.5 is that each iteration of the binary search has two logical steps. In the first step, we solve a cSamp-style linear program over the enabled OD-pairs. In the second step, we define the capped functions $\hat{C}_i(\tau) = \min_i(C_i, \tau)$ for the non-enabled OD-pairs and use the greedy algorithm to maximize $\hat{F} = \sum_i \hat{C}_i$.

In each iteration, for the current value $\tau_{current}$, the first step involves solving the LP $enabledODs$. The input to the LP is the set of enabled OD-pairs $\mathcal{P}_e$ and the target fractional coverage $\theta = \tau_{current}$. The objective of the LP is to minimize the total amount of resources used across the different routers to ensure that each $OD_i \in \mathcal{P}_e$ gets coverage at least

$\theta = \tau_{current}$. Solving the LP returns the resources allotted to each router or returns an infeasible status if there is no feasible solution.

If the LP is infeasible, then we directly proceed to the next iteration of the binary search. If the LP is feasible, then we obtain the new budget per router by subtracting the resources used in the LP stage from the original budget per router. Next, we run the greedy algorithm with the reduced budget and modified objective specified over the non-enabled OD-pairs. By construction, the maximum value $\hat{F}$ can take is $(M - |\mathcal{P}_e|) \times \tau_{current}$ where M is the total number of OD-pairs and $|\mathcal{P}_e|$ is the number of enabled OD-pairs. This maximum value is achieved if and only if each of the non-enabled OD-pairs (i.e., in the set $\mathcal{P} \setminus \mathcal{P}_e$) achieves a fractional coverage equal to $\tau_{current}$. If the greedy algorithm achieves this objective value, then $\tau_{current}$ is feasible and we try a higher value in the next iteration; else we try a lower value in the next iteration.

3.6.3 Using the α-fairness function

The idea of α-fairness has been used in the congestion control literature (e.g., [120]) to generalize the notion of max-min fair allocation. Given items x_i, and a total resource C we want to allocate the total resource to the items in a "fair" manner. The α-fairness function is defined as $\sum_i U(x_i)$, where $U(x) = \frac{x^{1-\alpha}}{1-\alpha}$. The parameter α can take values in $[0, \infty)$, and the values $\alpha = 0$, $\alpha = 1$,[3] and $\alpha \rightarrow \infty$ correspond to achieving maximum throughput, proportional fairness, and max-min fairness respectively.

In our problem setting, each x_i corresponds to the submodular function $F_i = C_i$. It is easy to check that $\sum_i U(C_i)$ is submodular; thus, we can use SUBMODULARGREEDY with α set to some large value. To avoid numerical instabilities, we use $\alpha = 100$ and also add a small additive constant to each C_i at the beginning since the function $U(x)$ is undefined when $x = 0$. Note that unlike the above heuristics, using the α-fair function is tightly coupled to maximizing the minimum fractional coverage.

3.7 Evaluation

Evaluation Setup: We compare the performance of cSamp and cSamp-T at a PoP-level granularity, i.e., treating each PoP as a "router" in the network model. Our evaluation setup (Table 2.1) consists of several PoP-level network topologies from educational backbones

[3] At $\alpha = 1$, the function is defined as $U(x) = \log(x)$.

and tier-1 ISP backbones inferred by Rocketfuel [157]. We use shortest-path routing to construct paths between every OD-pair. The traffic matrix is modeled using a gravity model based on city populations [150]. We assume that each PoP is provisioned to log up to $L = 400,000$ flow records.[4] For cSamp-T, we discretize the hash-range in increments of $\delta = 0.02$.

3.7.1 Coverage and Overlap

Total flow coverage: We are interested in two aspects: (a) the granularity of SamplingSpecs and (b) is there a significant difference in performance between the benefit or benefit-cost tradeoff versions of the greedy algorithm.

We consider three granularities of SamplingSpecs: router, router 3-tuple, and router 3-tuple augmented with egress information. Note that the first two SamplingSpecs can always be inferred from just local information but there may be some residual ambiguity in resolving the egress (Table 3.1). We use the tuple+egress as a hypothetical solution to see the gap between it and other solutions. We also compare these to cSamp and a maximal uncoordinated flow sampling solution. Recall that in maximal flow sampling, the flow sampling rate for a router is $\min(1, \frac{l}{t})$, where l is the number of flow records it is provisioned to hold and t is the total number of flows it observes; each node maximally utilizes the available resources.

Figure 3.8 shows that using 3-tuple SamplingSpecs provides significant improvement (25-30%) over the router-level case. cSamp-T (3-tuple+egress) is closest to cSamp, but the gap between the 3-tuple and egress-added cases is small. cSamp-T with the tuple formulation is closest to cSamp.

The theoretical guarantee for total flow coverage depends on running the two greedy algorithms: with and without the cost-benefit flag. We want to understand if there is a clear difference in performance between the two configurations. Figure 3.9 shows that both configurations have very similar performance and that the algorithm with the cost-benefit flag $cbflag = \mathbf{false}$ is slightly better.

Minimum fractional coverage: We saw in Section 3.4 that it is impossible to maximize f_{min} using a greedy algorithm without resource augmentation. Thus, we evaluate the performance as a function of the resource augmentation factor γ where each router can log $\gamma \times 400,000$ flow records. Here, we only consider the router and tuple granularities.

[4] Assuming 12 bytes per flow record [147], this translates into $400,000 \times 12 = 4.8$ MB of SRAM per PoP, which is well within the 8 MB technology limit per linecard suggested by Varghese [167].

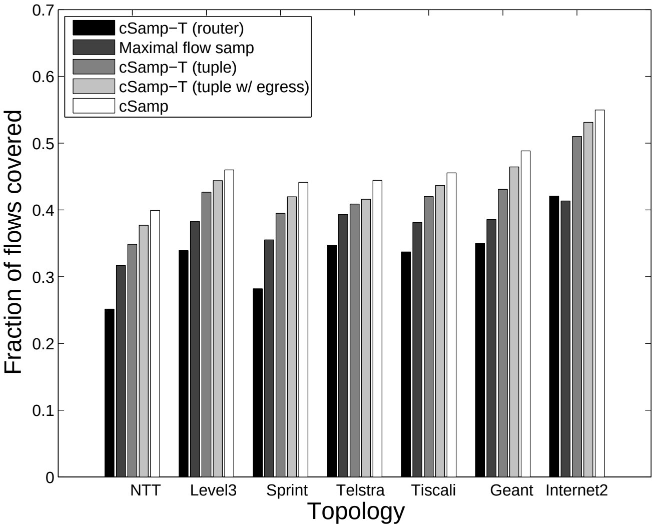


Figure 3.8: Total flow coverage

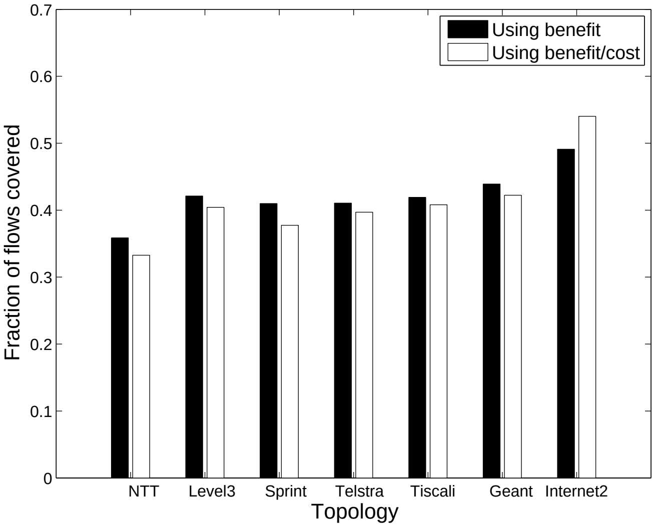


Figure 3.9: Benefit vs. benefit-cost versions

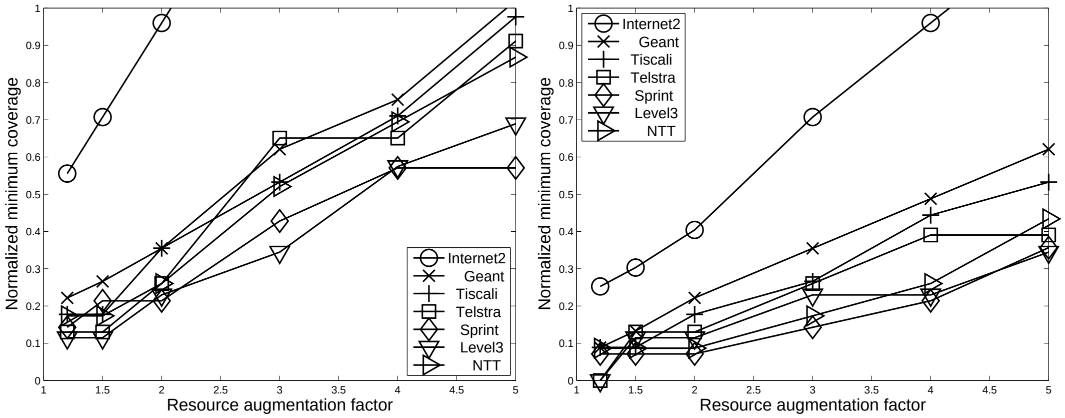
The tuple+egress was almost identical to the tuple case; we do not show this for brevity. In Figure 3.10, we normalize the minimum fractional coverage by the optimal value achieved by cSamp at the baseline provisioning (i.e., cSamp at $\gamma = 1$). For example, if the greedy algorithm returned a value of 0.2 at $\gamma = 3$ and the solution for cSamp has value 0.4 at

$\gamma = 1$, the normalized y-axis value corresponding to $\gamma = 3$ is $\frac{0.2}{0.4} = 0.5$.

With $\gamma \geq 4$, cSamp-T has performance comparable ($\geq$ 50%) to cSamp for all topologies. Also, the difference between the router and tuple formulations becomes even more pronounced with the minimum fractional coverage result – there is a significant advantage to be gained in using more fine-grained SamplingSpecs. With router-level SamplingSpecs, even at $\gamma = 5$, four out of the seven topologies only reach 40% of cSamp's performance. For the same $\gamma = 5$, with tuple-level SamplingSpecs, five out seven topologies achieve at least 90% of cSamp's performance.

Figure 3.11 shows the corresponding result when we use the α-fairness objective function with the tuple formulation. We see that this function gives slightly better performance compared to the capped-minfrac technique used above.

The γ at which cSamp has good performance is much better than the theoretical bound in Section 3.4. In Section 3.7.4, we show that targeted provisioning reduces this even further.



(a) Tuple

(b) Router

Figure 3.10: Normalized minimum fractional coverage achieved by cSamp-T as a function of the resource augmentation factor

These results show that 3-tuple SamplingSpecs perform much better than router SamplingSpecs, and are very close to the tuple+egress case. Thus, we focus on 3-tuples for the rest of the evaluation.

Performance gap between cSamp and cSamp-T: The approximation guarantees compare the performance of the greedy algorithms with the optimal solution for the cSamp-T

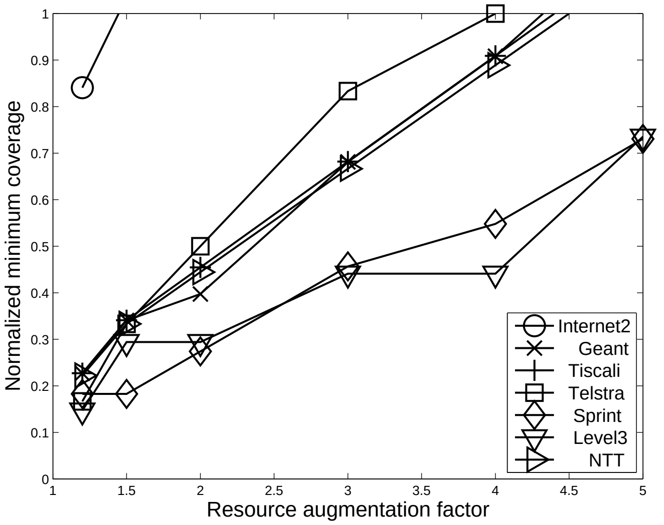
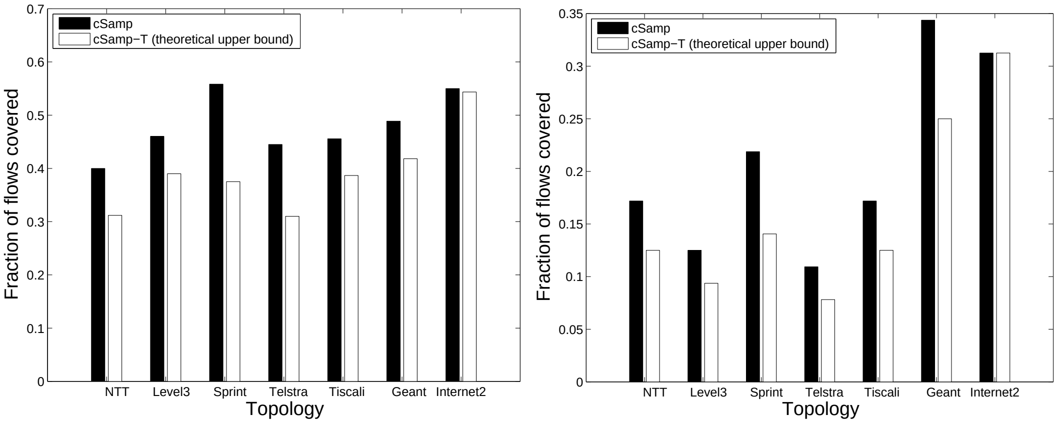


Figure 3.11: Normalized minimum fractional coverage using the α-fair function with the tuple formulation



(a) Total coverage

(b) Minimum coverage

Figure 3.12: Performance gap between cSamp and theoretical upper-bound for cSamp-T

problem. A related question is the gap between the optimal solutions for cSamp-T and cSamp. It is hard to reason about the optimal cSamp-T solution. Instead, we compare the theoretical upper bound for the cSamp-T problem by considering a relaxed LP-version of the problem (similar to the *provisioning* problem in Section 3.6). Figures 3.12(a) and 3.12(b) show that this performance gap for the total flow coverage and the minimum coverage respectively using a router 3-tuple granularity for cSamp-T. The figure shows that the upper bound on cSamp-T performance can be up to 30% lower than cSamp. Comparing this with Figure 3.8, we also see that the greedy algorithm is very close to the theoretical upper bound for cSamp-T in case of the total flow coverage.

Duplicated flow reports: A secondary objective of cSamp is to minimize the total amount of duplicated flow reports. This reduces the data management overhead in processing and eliminating duplicated flow measurements. Figure 3.13 shows the ratio of duplicated flow reports to the number of unique flow reports comparing cSamp-T (at the tuple granularity) and maximal flow sampling. Compared to maximal flow sampling, cSamp-T has 2-3× fewer duplicated flow reports. Compared to cSamp (zero duplicated reports) this is not ideal; however, this performance penalty is unavoidable since cSamp-T operates at a much coarser granularity.

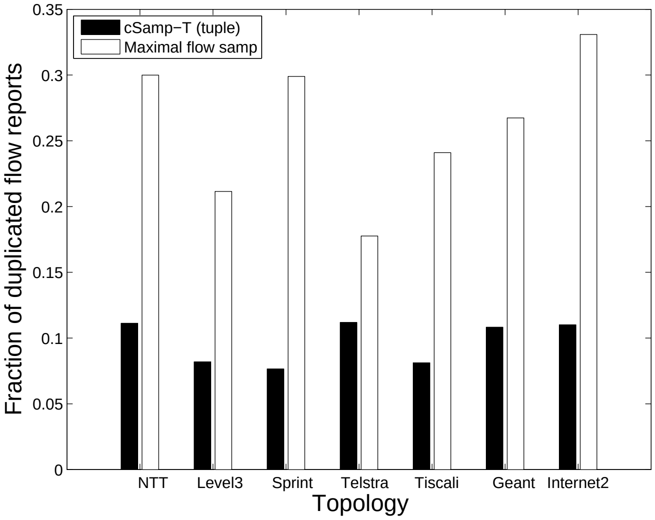


Figure 3.13: Ratio of duplicated flow reports to the number of unique flow reports

3.7.2 Algorithm Running Time

In order for cSamp-T to be reasonably responsive to network dynamics, we want the time to compute sampling manifests to be within few tens of seconds. (A typical measurement epoch spans a few minutes; we expect that manifests are recomputed across epochs, not within epochs.) Table 3.3 shows the computation times using the “vanilla” greedy and lazy evaluation algorithms. Lazy evaluation provides more than an order of magnitude reduction in the total computation time. The reduction is even more significant for the minimum fractional coverage since it involves multiple invocations of the greedy subroutine during the binary search. With this reduction, cSamp-T scales to larger topologies.

Topology	**Total coverage (sec)**		**Min. Fractional (sec)**	
	Naive	**Lazy**	**Naive**	**Lazy**
NTT	207.12	4.15	39632	154.1
Level3	205.36	3.30	48269	84.3
Sprint	75.30	2.21	14211	71.6
Telstra	50.53	1.65	6997	45.0
Tiscali	35.18	1.16	8518	33.7
GÉANT	3.06	0.28	542	7.6
Internet2	0.22	0.05	38.4	1.9

Table 3.3: Time to compute sampling strategy comparing the vanilla greedy algorithm with the lazy evaluation optimization

Next, we evaluate how the algorithms scale to very large router-level topologies. We generate router-level topologies by treating each PoP as a “core” router and add 4 edge routers to each of these core routers. As described earlier, we use two extra optimizations: parallel execution within each greedy iteration and tighter upper bounds for the binary search. Table 3.4 show that even for these very large topologies, the compute times are within reasonable bounds and can be further reduced by increasing the degree of parallelization.

Topology	**# Routers**	**Total Cov. (sec)**	**Min. Frac. (sec)**
NTT	350	345.9	994.7
Level3	315	224.1	540.2
Sprint	260	174.0	554.6
Telstra	220	180.7	267.6
Tiscali	205	77.0	327.4

Table 3.4: Compute times for large router-level topologies with 4 threads in parallel

3.7.3 Size of Sampling Manifests

Compared to cSamp, cSamp-T increases the size of the sampling manifests. This is because, unlike cSamp, the hash-ranges assigned for each SamplingSpec are no longer contiguous blocks. To reduce the size of the manifests, we implement a simple compression heuristic to merge hash-ranges after the greedy algorithm computes the manifests. This looks for maximally contiguous hash ranges in the original sampling manifest and merges them into a single hash range.

We evaluate the overhead of disseminating manifests in Table 3.5. First, the merge algorithm reduces the manifest sizes roughly 10×. Second, we notice that the total bandwidth overhead of disseminating the manifests is not large – 25KB in the worst case after the merge routine. Finally, on a per-router basis, the worst case size of the manifest is around 3KB which is quite low.

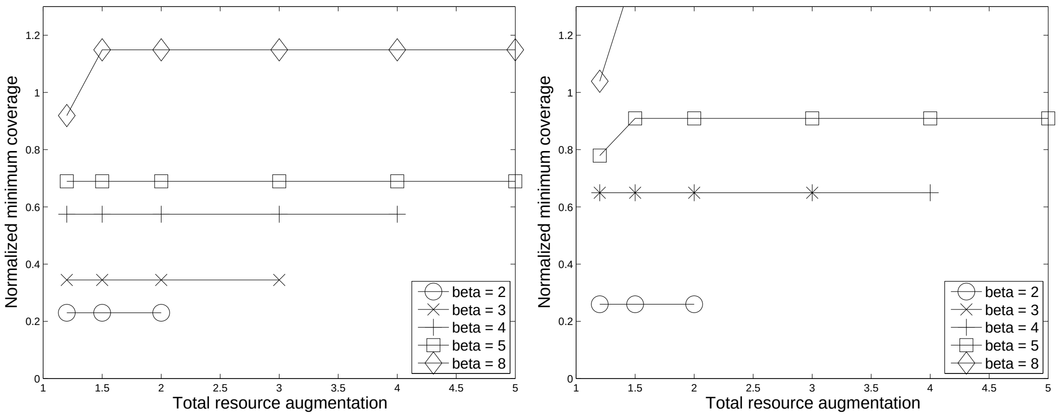
Topology	**Total (KB)**		**Max. per PoP (KB)**	
	Naive	**Merged**	**Naive**	**Merged**
NTT	178.5	16.3	5.6	1.0
Level3	341.9	25.2	34.1	3.3
Sprint	140.9	13.0	10.3	0.6
Telstra	112.3	7.2	3.3	0.5
Tiscali	110.9	12.6	9.8	0.6
GÉANT	45.5	6.5	5.6	0.6
Internet2	14.5	5.0	4.5	0.7

Table 3.5: Size of the sampling manifests (in kilobytes of text configuration files) with cSamp-T

3.7.4 Intelligent Resource Provisioning

As a specific scenario, we set $LB_j = L = 400,000$ for all j. We model the total budget as $Budget = \gamma \times N \times L$ (N is the number of PoPs) and the technology limit as $\beta \times L$. We vary γ and β and for each pair of values. Figures 3.14(a) and 3.14(b) show the result for two of the topologies, Level3 (AS3356) and Telstra (AS1221) respectively. We chose these topologies because the greedy algorithm performed poorly with respect to cSamp in Figure 3.10. An interesting result is that the curve levels off as a function of γ; i.e., there is not much to be gained with increasing the total budget. However, there is significant improvement by increasing β, the technology upper bound. In fact, even with a moderate increase $\gamma = 1.2$, we see that the performance gets within 80% of the cSamp performance.

Since β is more crucial to the overall performance than γ, for the remaining topologies we fix $\gamma = 1.5$ and analyze the normalized minimum fractional coverage as a function of β in Figures 3.15 and 3.16. With $\beta = 5$, all the topologies achieve at least 60% of the



(a) Level3 (b) Telstra

Figure 3.14: Understanding the impact of total resource augmentation (γ) and technology upper bound (β) in the resource allocation formulation.

ideal cSamp performance. Similar to the previous results, the α-fair shows slightly better performance. Contrasting this result with Figures 3.10 and 3.11, the main difference is that we do not require all PoPs to be augmented with five times as many resources – the total resource budget is less than $1.5\times$.

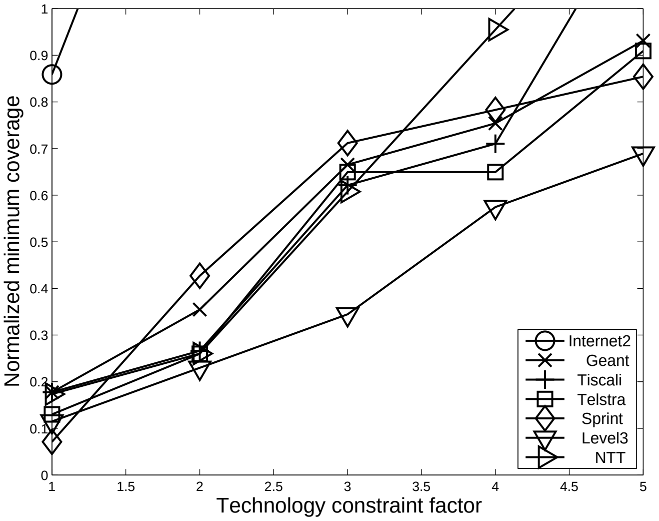


Figure 3.15: Intelligent resource allocation with $\gamma = 1.5$ and varying β

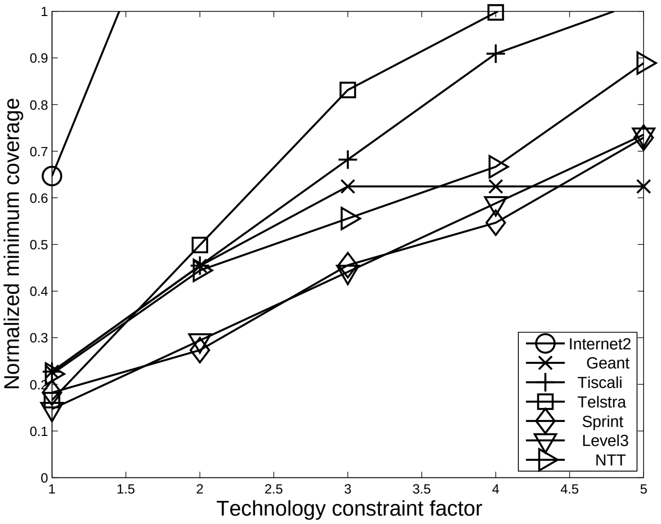
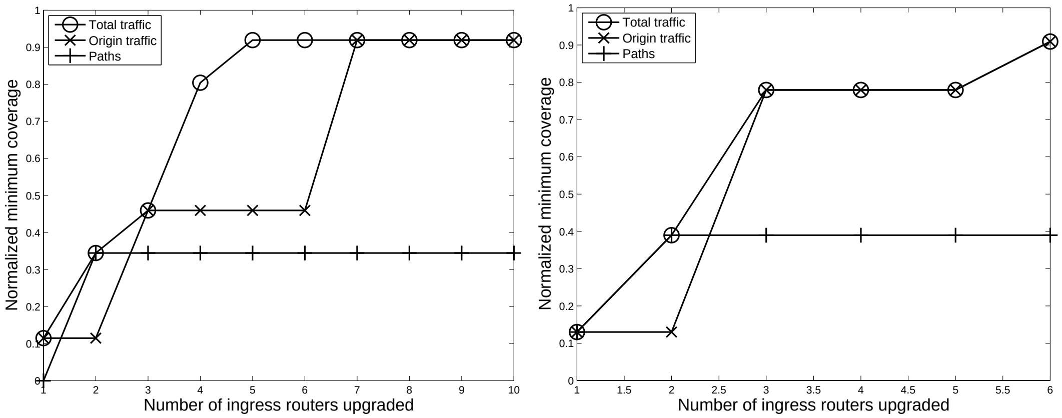


Figure 3.16: Intelligent resource allocation with $\gamma = 1.5$ with the α-fair function



(a) Level3

(b) Telstra

Figure 3.17: Performance of cSamp-T with partial OD-pair identification. Alternatively, this can be viewed as incremental deployment of cSamp via cSamp-T.

3.7.5 Partial OD-pair Identification

We try three strategies for selecting the enabled OD-pairs $\mathcal{P}_e$: upgrading the top-k PoPs that (a) observe the maximum amount of traffic, (b) lie on most number of routing paths, or

AS	**Greedy-Minfrac**		**Greedy-Total**
	NoHybrid	**Hybrid**	
NTT	0.13	0.58	0.58
Level3	0.10	0.60	0.60
Sprint	0.22	0.61	0.64
Telstra	0.13	0.59	0.62
Tiscali	0.23	0.60	0.63
GÉANT	0.35	0.63	0.68
Internet2	0.60	0.71	0.78

Table 3.6: Comparing the performance of the hybrid maximization to the greedy algorithm for maximizing the total flow coverage alone

(c) originate the most traffic. Here, upgrading implies that we enable OD-pair identifiers on all OD-pairs having one of these top-k PoPs as origins. For each k, we run the two-step procedure from Section 3.6.2 for all values in $1, \ldots, k$ and pick the configuration with the highest f_{min}.

Figures 3.17(a) and 3.17(b) show the normalized minimum fractional coverage for the Level3 and Telstra topologies as a function of k (number of top-k PoPs). First, we observe that enabling even on a small number (around 8%) significantly improves the performance. Second, enabling identifiers on routers that observe the most traffic performs much better than the other two strategies.

3.7.6 Hybrid Coverage Objective

cSamp maximizes the total flow coverage subject to achieving the highest possible minimum fractional coverage across OD pairs. So far, in cSamp-T we considered these two objectives separately. A natural question is if there is an effective algorithm for maximizing the hybrid objective, i.e., maximize total coverage subject to achieving the maximum minimum fractional coverage. It is relatively simple to extend the algorithm in Figure 3.5 to achieve this – first run the greedy algorithm to optimize the capped minimum fractional objective ($\hat{F}$) and then modify the objective function to optimize the total coverage if $\tau_{current}$ is feasible.

To evaluate this hybrid approach, we consider the resource configuration obtained using the targeted provisioning approach with $\gamma = 1.5$ and $\beta = 5$. Table 3.6 compares the total coverage obtained with three strategies: maximizing the minimum fractional coverage, maximizing the total flow coverage, and the above two-step heuristic. Not surprisingly, we find that maximizing the minimum fractional coverage alone does not work well for the total coverage. This is because the greedy algorithm terminates when it has achieved the targeted coverage for all OD-pairs even if it has additional resources that can be used to boost the total coverage. The table also shows that total coverage obtained by the hybrid approach is very close to that of the greedy algorithm for maximizing the total coverage alone. While it is hard to provide theoretical guarantees for the hybrid objective, Table 3.6 shows that our approach works very well in practice.

3.8 Discussion

More fine-grained local information: Our current choice of SamplingSpecs is topology-driven; we model the granularity of sampling manifests in terms of path-segments (e.g., router or router 3-tuple). One direction of future work is to expand the scope to include prefix and routing table information. For example, it might be possible to approximately estimate the OD-pair information given the source and destination address of a packet and the available routing table information or alternatively providing additional information (e.g., distributing IP-prefix to ingress-egress maps to routers [35]). This creates the possibility of a cSamp-T formulation with more fine-grained information to bring the performance closer to cSamp.

Sensitivity of router upgrades: Section 3.6 suggests two heuristics for upgrading routers either with additional memory or the ability to insert OD-pair identifiers in packet headers. The provisioning and partial marking formulations, as presented, assume static routing and a static traffic matrix. Real-world routing and traffic matrices typically have some dominant structural patterns that are invariant to localized dynamics. Thus, we can apply these formulations and perform upgrades after extracting these dominant patterns. Evaluating the sensitivity of the performance improvements to traffic or routing dynamics and designing upgrade strategies robust to dynamics are topics of future work.

3.9 Related Work

Theory of submodularity: Submodular set-functions have long been studied as discrete analogs of convex functions: in particular, maximizing a submodular function subject to side constraints has a rich history; see, e.g., [44, 172, 161] and the references therein.

Greedy algorithms for monitor placement: Prior work has applied greedy algorithms for monitor placement to cover all routing paths using as few monitors as possible [47, 159]. The authors show that such a formulation is NP-hard and propose greedy approximation algorithms. There are also extensions to these problems to incorporate packet sampling [159, 45]. However, these do not satisfy flow coverage objectives, and in fact by relying on packet sampling, they can result in a large amount of redundant flow measurements. cSamp-T provides more fine-grained flow coverage objectives and reduces duplicated flow reports.

Sensor network monitoring: There has been recent work applying the theory of maximizing submodular set cover functions in the context of maximizing information obtained from multiple sensors [74, 92]. The objective of selecting observations against a set of adversarial objectives [93] is similar to the notion of maximizing the minimum fractional coverage objective. Krause and Guestrin [91] provide a good survey of known results and applications of these ideas.

3.10 Chapter Summary

cSamp is a promising architecture to meet the demand for fine-grained flow monitoring capabilities in ISPs. However, ISPs cannot realize the benefits of cSamp in practice because of its reliance on OD-pair identifiers; it requires changes to packet headers, imposes additional overhead at ingress routers, and may require ISPs to overhaul their routing infrastructure.

This chapter described cSamp-T, a framework that provides benefits comparable to cSamp, in which the sampling decisions at routers are based only on local information, and do not rely on global OD-pair identifiers. Obtaining exact solutions to maximize the total flow coverage (f_{tot}) and minimum fractional coverage (f_{min}) is NP-hard. We achieve near-optimal performance for f_{tot} by leveraging its submodularity. For f_{min}, getting good performance without resource augmentation is provably hard. However, targeted provisioning achieves near-ideal performance with low overhead. Alternatively, upgrading a small number of border routers to provide OD-pair information also yields good results.

cSamp-T thus makes the benefits of coordinated network-wide monitoring solutions like cSamp more immediately available to ISPs and also provides an incremental deployment path for ISPs to transition to cSamp.

Chapter 4

Revisiting the Case for A Minimalist Flow Monitoring Architecture

Flow monitoring supports critical network management tasks such as traffic engineering [66], anomaly detection [99, 100], accounting [55, 62], identifying and analyzing end-user applications [49, 86], understanding traffic structure [174], detecting worms, scans, and botnet activities [176, 168, 134], and forensic analysis [173]. These require high-fidelity estimates of traffic metrics relevant to each application.

High traffic rates exceed the monitoring capabilities of routers,[1] and since traffic is scaling at least as fast as routers' capabilities, some form of sampling or data reduction is necessary in commodity solutions. (There are high-end solutions for full packet capture [12]. These are expensive and require specialized instrumentation.) The de-facto standard is NetFlow [48, 11] which uses packet sampling. Each packet is sampled with some probability and the selected packets are aggregated into flows[2]. NetFlow-style monitoring is sufficient for coarse applications such as traffic volume estimation, but several studies have shown the inadequacy of packet sampling for many of the fine-grained monitoring applications mentioned earlier (e.g., see [118, 79, 56, 96, 40, 134, 62]).

Consequently, several research efforts have focused on application-specific monitoring techniques. This is exemplified by the proliferation of data streaming algorithms for computing the flow size distribution [96], entropy [102], superspreader detection [168], degree histogram estimation [176], change detection [94], and so on.

[1] Our arguments apply to non-router monitors as well. For simplicity, we use the term router as it represents operational realities.

[2] A flow is a sequence of packets with the same IP 5-tuple $\langle$ srcip, dstip, srcport, dstport, protocol $\rangle$.

While this body of work has made valuable algorithmic contributions, this shift to application-specific approaches is undesirable for two reasons:

- First, this increases the implementation complexity and resource requirements of routers.
- Second, the set of applications is a moving target as normal and anomalous traffic patterns change. This requires router vendors and network managers to commit to a fixed set of application-level metrics without knowing if these will meet future requirements.

We reflect on these trends and ask a fundamental question in this chapter:
Is such complexity and early commitment necessary?
Are there simpler alternatives that can provide the requisite fidelity and generality?

Approach and Intuition: We revisit the case for a *minimalist* approach that retains the simplicity of NetFlow, where routers only need to support a *few* monitoring primitives, but still provide coverage over a wide spectrum of applications.

To understand how we can achieve this, we can think of each monitoring application as being composed of two logical phases: (1) a *collection* phase that needs to operate at line rates and (2) an *estimation* phase to compute different traffic metrics that need not strictly work at line rates. Application-specific alternatives tightly couple these two components, only retaining counters and statistics relevant to a specific application context (Figure 4.1). In contrast, we can envision a minimalist approach that *decouples* the collection and estimation phases as much as possible.

A key question is whether such an approach can provide estimation accuracy comparable to application-specific alternatives. One rationale to suggest that it can, is that the primary bottleneck for monitoring is keeping counters in fast memory (SRAM). Instead of splitting the available memory across different applications, we can aggregate it, and run a few simple primitives with high-enough sampling rates to obtain accurate estimates of traffic metrics for a wide spectrum of applications. In other words, when we look at each application in isolation, application-specific strategies are appealing. However, when we consider a portfolio of applications in aggregate, a minimalist approach might be a better alternative.

Contributions and Implications: Our goal is not to design an optimal minimalist approach. Rather, our objective is to establish a *feasible* instance.

We present a practical minimalist approach in Section 4.3 that combines sample-and-hold [62], flow sampling [79], and cSamp [147]. Our choice of these specific primitives

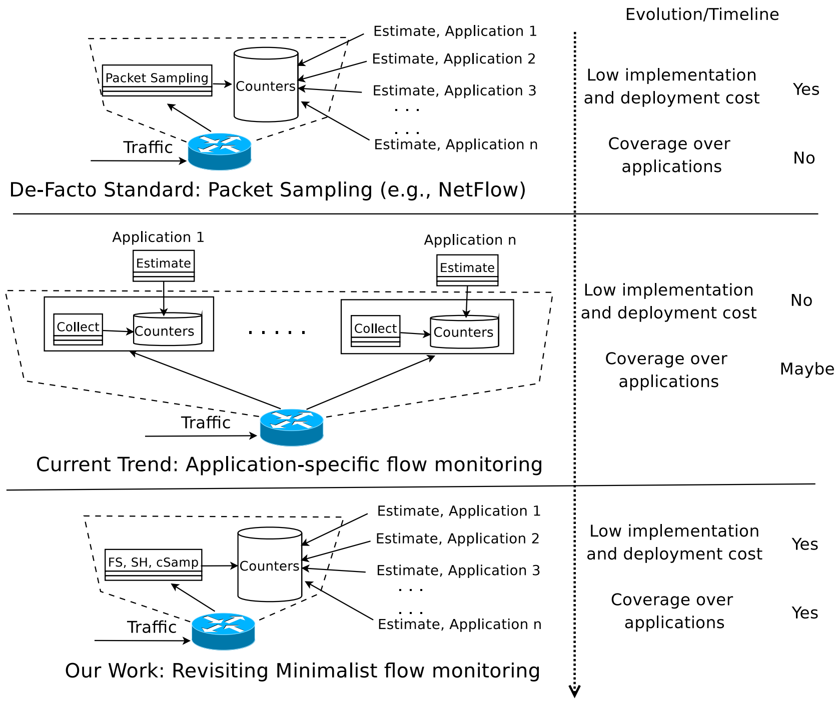

Evolution/Timeline
Packet Sampling
Counters
Estimate, Application 1
Estimate, Application 2
Estimate, Application 3
Estimate, Application n
Traffic
De-Facto Standard: Packet Sampling (e.g., NetFlow)
Low implementation and deployment cost
Yes
Coverage over applications
No
Application 1
Application n
Estimate
Collect
Counters
Traffic
Current Trend: Application-specific flow monitoring
Low implementation and deployment cost
No
Coverage over applications
Maybe
FS, SH, cSamp
Counters
Estimate, Application 1
Estimate, Application 2
Estimate, Application 3
Estimate, Application n
Traffic
Our Work: Revisiting Minimalist flow monitoring
Low implementation and deployment cost
Yes
Coverage over applications
Yes

Figure 4.1: A minimalist approach runs a few collection algorithms. Applications can use the collected data later (possibly offline). NetFlow/packet sampling is a minimalist approach, but it is not well-suited for many applications. An application-specific architecture implements many focused algorithms. These work well for the specific applications, but increase complexity and are not robust to changing demands. We demonstrate a minimalist alternative that performs favorably w.r.t application-specific approaches over a wide spectrum of applications.

is guided by the understanding that monitoring applications fall into two broad classes that analyze (1) *volume structure* (e.g., traffic engineering) or (2) *communication structure* (e.g., security applications). Flow sampling is ideally suited for the latter class [79, 118, 115] and sample-and-hold for the former [62]. cSamp provides a framework to efficiently leverage router resources to meet network-wide monitoring goals.

We use trace-driven analysis to evaluate this design against several application-specific approaches (Section 4.5): detecting heavy hitters [62], superspreaders [168], and large traffic changes [94]; computing entropy [102] and the outdegree histogram [176]; and estimating the flow size distribution [96]. When our approach has the same total memory resources as that used by the different application-specific algorithms in aggregate, it provides comparable or better estimation accuracy across the entire spectrum of applications. Moreover, by delaying the binding to specific applications, it enables computation of not-yet-conceived measures that will be interesting in the future.

This chapter shows the promise of a minimalist approach even with a simple combination of existing techniques. We believe that this has significant implications for router vendors, network operators, and measurement researchers. First, it can reduce router complexity without compromising a vendor's ability to satisfy its customers' demands. Second, it helps insulate network deployments efforts from the changing needs of monitoring applications. Finally, we hope that our work encourages future research in developing better minimalist primitives and estimation algorithms, and in understanding their fidelity for different applications.

4.1 Background and Related Work

Packet sampling: Router vendors today use uniform packet sampling [48]: a router selects a subset of packets, and aggregates the sampled packets into flow reports. However, packet sampling has inherent limitations. There are known biases toward sampling larger flows (e.g., [79, 96, 118]) and several studies have questioned its fidelity for many management applications (e.g., see [118, 79, 56, 96, 40, 134, 62]).

Application-specific approaches: The limitations of packet sampling have motivated many application-specific data streaming algorithms [28, 123]. The high-level approach is to use a small number of SRAM counters pertinent to each application and then estimate the relevant traffic statistics from these counters. These include algorithms for estimating the flow size distribution [96, 138], identifying heavy hitters [62, 90], entropy estimation [102, 75, 24], superspreader detection [168], degree histogram estimation [176], and change detection [94, 144]. However, these approaches are tightly coupled to the specific applications and report summary statistics pertinent to each application. Thus, it is difficult to estimate or extrapolate other measures of interest from these reports. Therefore, these lack the generality to serve as minimalist primitives.

Some data structures (e.g., count-min sketches [50]) provide more generality. However, these have two limitations. First, they are designed primarily for coarse *volume queries* and thus less suited for more fine-grained tasks like entropy estimation and superspreader detection. Second, sketches operate with a specific "flowkey" defined over one or more fields of the IP 5-tuple (srcip, dstip, srcport, dstport, protocol). Each flowkey of interest requires a separate instance of the data structure. However, it is often necessary to analyze combinations of two or more fields for diagnostic purposes (e.g., investigating anomalies). A separate instance for each required combination incurs high overhead. Furthermore, this needs advance knowledge of which flowkeys will be useful, which may not be known until after the operator begins to investigate specific events.

Selective sampling: Some approaches assign different sampling rates for different classes of packets [98, 134]. Others only log flows with pre-specified patterns (e.g., [180, 9, 14, 116, 41]). While these approaches provide some flexibility, they need to know the specific classes and sampling rates to meet the applications' requirements. In contrast, we envision a minimalist approach that is agnostic to the specific types of analyses that may be performed.

Network-wide measurements: Many studies have stressed the importance of network-wide measurements to meet operational requirements as applications and attacks become massively distributed [66, 99, 100]. For example, understanding peer-to-peer traffic [49], detecting botnets [134] and hit-list worms [115], understanding DDoS attacks [145], and network forensics [173] inherently require a network-wide view aggregated from multiple vantage points. In this respect, recent proposals show the benefits of moving beyond router-centric solutions to network-wide monitoring solutions [45, 147].

4.2 Design Considerations

Given this background, we synthesize key requirements for a flow monitoring architecture and derive guiding principles for a minimalist approach, echoing the charter of the IETF PSAMP working group [16].

4.2.1 Requirements

Low router complexity: Given the hardware and development costs involved in modern router design, we want to keep router implementations as simple as possible.

Generality across applications: The monitoring infrastructure should cover a wide spectrum of applications and ideally be robust to future application needs.

Enable diagnostics: The monitoring architecture should support diagnostic "drill-down" tasks; e.g., by providing the capability to give different views into traffic structure.

Provide network-wide views: The monitoring architecture should provide network-wide capabilities as these are increasingly crucial for several aspects of network management and traffic analysis as discussed earlier.

4.2.2 Design Principles

A few, simple, and generic primitives: A natural way to reduce router complexity is to have a few primitives that are easy to implement but powerful enough to support many management tasks.

Decouple collection and computation: Now, how can we provide generality and support diagnostics with a few monitoring primitives? We believe that this best achieved by *decoupling* the collection and computation involved in monitoring tasks. Note that this is already implicit in network operations today: routers export NetFlow reports to a (logically) central collector and operators analyze this data. We retain this operational model; routers run some collection algorithms and export the collected flow reports. Once we have the flow-level reports, we can compute any traffic metric of interest and provide different views required for further diagnosis.

Network-wide resource management: To provide network-wide capabilities, we need a framework that assigns monitoring responsibilities across routers to satisfy network-wide monitoring goals. At the same time, this framework should be *resource-aware*; i.e., respect the resource constraints (e.g., memory) of routers.

4.2.3 Challenges

Given the above considerations, two questions remain:

1. **Concrete Design:** What primitives should be implemented on routers to support a range of applications? How should monitoring responsibilities be assigned to meet network-wide measurement goals?

2. **Performance:** Does the intuitive appeal of a minimalist approach translate into quantitative benefits for a wide spectrum of applications?

In addressing these challenges, our goal is not to look for an “optimal” minimalist approach. (In fact, it is not clear if we can formally reason about optimality without committing to a fixed set of applications.) Rather, we want to look for a *feasible* instance that covers a broad spectrum of applications. We present one such proposal in the next section.

4.3 Architecture: Components and Combination

The first challenge is to choose a small set of generic collection primitives that runs on each router and to design a framework to manage them intelligently across a network of routers. Our specific proposal combines three ideas: flow sampling [79] and sample-and-hold [62] for single router sampling algorithms, and cSamp [147] for network-wide management. Keys et al. designed a system for providing traffic summaries and detecting "resource hogs" [88] using a combination of flow sampling and sample-and-hold, similar to our approach. We extend their work in two significant ways. First, we show how to combine these primitives with the network-wide capabilities of cSamp [147] in contrast to the single-vantage-point view in their work. Second, we look beyond simple traffic summaries and demonstrate that this combination can support a much wider range of applications.

4.3.1 Router Primitives

Choice of primitives: Flow monitoring applications can be divided into two broad classes: (1) those that require an understanding of *volume structure*; e.g., heavy-hitter detection and traffic engineering that require an understanding of the number of packets/bytes per-port or per-src and (2) those that depend on the *communication structure*; e.g., security applications and anomaly detection application that require an understanding of "who-talks-to-whom". Our choice of primitives is guided by these two broad classes. Flow sampling is well suited for security and anomaly detection applications that analyze communication structure [79, 118, 115]. Similarly, sample-and-hold is well suited for traffic engineering and accounting applications that analyze volume structure [62]. Thus, these two primitives effectively complement each other in their capabilities.

For the following discussion, a flow refers to the IP 5-tuple.[3] We use flow sampling and sample-and-hold at this 5-tuple granularity. The rationale is to collect flows at the most general definition possible. The collected flows can be sliced-and-diced after the fact by projecting from this general definition to more specific definitions (e.g., per destination port, per source address).

[3] $\langle srcaddr, dstaddr, srcport, dstport, protocol\rangle$

Sample-and-Hold (SH): Sample-and-hold (SH) [62] keeps near-exact counts of “heavy hitters”–flows with high packet counts. SH works as follows. For each packet, the router checks if it is tracking this packet’s *flowkey*, defined over one or more fields of the IP 5-tuple. If yes, the router updates that counter. If not, the flowkey for this packet is selected with probability p, and the router keeps an exact count for this selected flowkey subsequently. Since this requires per-packet counter updates, the counters are kept in SRAM [62].

To configure SH, we specify the flowkey(s) (e.g., srcport, srcaddr, or 5-tuple), the anticipated total number of packets for a specific time interval ($numpkts$), and the number of flows that can be logged (L) depending on the SRAM constraint. The probability p is set to $\frac{L}{numpkts}$.[4] In our design, we use one instance of SH and configure it to operate at the 5-tuple granularity.

Hash-based flow sampling (FS): Flow sampling (FS) picks flows rather than packets at random [79]. One way to implement FS is as follows. Each router has a *sampling manifest* — a table of one or more hash ranges indexed using a key derived from each packet header. On receiving a packet, the router computes the hash of the packet’s 5-tuple (i.e., the flowkey). Next, it selects the appropriate hash range from the manifest and selects the flow if the hash falls within this range. The hash is used as an index into a table of flows and it updates the byte and packet counters for the flow. The hash function maps the input 5-tuple uniformly into the interval $[0, 1]$. Thus, the size of each hash range determines the flow sampling rate for each category of flows in the manifest.

Similar to SH, FS requires per-packet table lookups; the flow table must therefore be implemented in SRAM. It is possible to add a packet sampling stage to make DRAM implementations possible [89]. For simplicity, we assume that the counters are stored in SRAM.

4.3.2 Resource Management

Having chosen FS and SH as our minimalist primitives, we address the following question. Given a fixed amount of SRAM available for monitoring on each router, how should we divide it between these primitives?

Combining FS-SH on a single router: Consider a single router with a fixed amount of

[4] To track heavy hitters who contribute more than a fraction $\frac{1}{x}$ to the total volume, p is set to $\frac{O \times x}{numpkts}$, where O is an oversampling factor [62]. Our configuration can be viewed as determining x and O from the memory budget L.

SRAM that can hold L flow counters. A simple way to split L is to give a fraction f to FS and the remaining $1 - f$ to SH. We show in Section 4.5 that $f \approx 0.8$ is a good choice.

Network-wide case: The above split works for the single router case. Next, we see how we can manage the monitoring resources across a network of routers. Network-wide management tasks are typically specified in terms of Origin-Destination pairs, specified by an ingress and egress router (or PoP). OD-pairs are convenient abstractions that naturally fit many of the objectives (e.g., traffic engineering) and constraints (e.g., routing paths, traffic matrix) in network management. A natural extension of the single router hybrid primitive to the network-wide case is to consider the resource split per OD-pair [45, 147].

Here, we observe a key difference between FS and SH. It is possible to coordinate FS instances by assigning non-overlapping responsibilities across routers on a path [147]. However, because SH logs heavy hitters, the same set of heavy hitters will be reported across routers on a path. Thus, replicating SH across routers on a path duplicates measurements and wastes router resources.

To address this issue, we make a distinction between ingress and non-ingress routers. Ingresses implement both FS and SH, sharing the aggregate memory as in the single router case. At each such ingress router, the SH resources are split between the OD-pairs originating at the ingress, in proportion to the anticipated number of packets per OD-pair. Non-ingress routers only implement FS. In order to distribute FS responsibilities across the network, we use cSamp from Chapter 2.

Example configuration: Figure 4.2 shows how the different components are combined in the network-wide case. There are three OD-pairs P1, P2, and P3 originating at the left-most router. We envision a configuration module at the network operations center which disseminates configurations to the routers. This module takes into account the prevailing network conditions, policies, router constraints, and the flow monitoring objectives to generate the FS and SH configurations for each router. In the example, the ingress router is assigned SH responsibilities for P1, P2, and P3. The non-ingress routers are not assigned any SH responsibilities for these OD-pairs. (The other edge routers could be assigned SH responsibilities for OD-pairs for which they are the origin, but these are not shown.) The FS responsibilities are generated using cSamp. Each router is only assigned FS responsibilities for the paths of OD-pairs it lies on and these are specified as non-overlapping hash ranges per OD-pair.

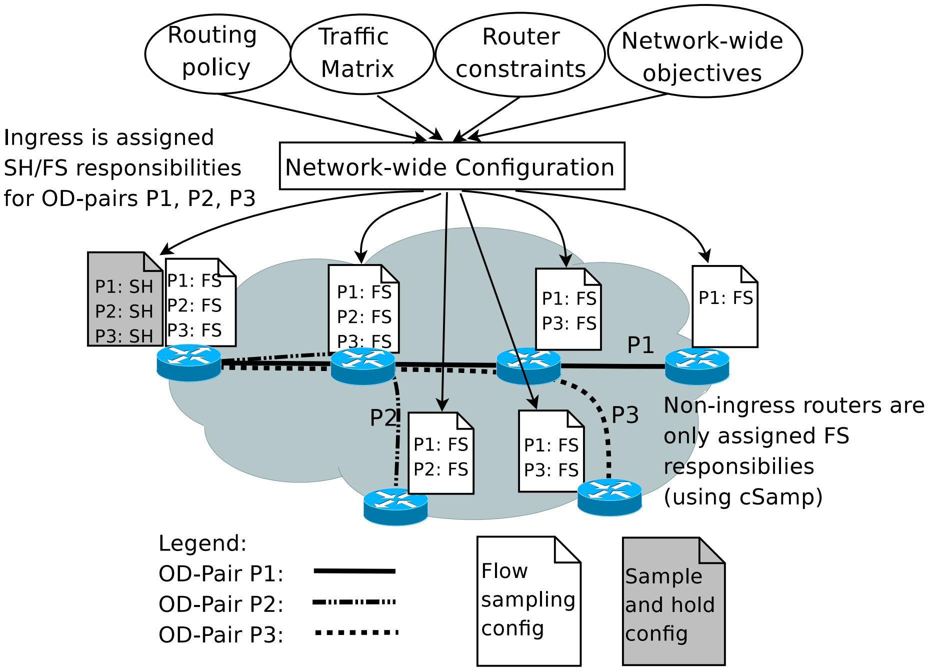


Figure 4.2: Overview of our network-wide approach

4.4 Evaluation Methodology

Our goal is to compare the minimalist design from the previous section against an application-specific architecture when both approaches are given the same total resource budget. In order to do so, we need to specify the different applications of interest, the corresponding application-specific algorithms, and the configurations for determining the resources provisioned for each algorithm.

First, we describe the different applications, the corresponding data streaming algorithms, and accuracy metrics in Section 4.4.1. Then, in Section 4.4.2, we describe how we normalize the resource usage of the minimalist and application-specific algorithms. We explain our assumptions and justify why these are conservative in that we underestimate the performance of an equivalently provisioned minimalist approach. Finally, in Section 4.4.3, we describe the configuration parameters for the different algorithms and the estimation phase for the minimalist approach in Section 4.4.4.

4.4.1 Applications and Accuracy Metrics

We pick a set of diverse monitoring applications that span the spectrum of traffic engineering, security, and anomaly detection tasks of interest to network operators. Table 4.1 summarizes the applications and the corresponding application-specific algorithms, accuracy metrics, and configuration parameters. The table also shows the default parameters we use in each case.

Flow size distribution (FSD) estimation: Let F denote the total number of flows in a traffic stream and F_l be the number of flows of size l (pkts per flow). The FSD estimation problem is to determine $\forall l = 1 \ldots z, \phi_l = \frac{F_l}{F}$, where z is the largest flow size. Understanding the FSD is useful for many management tasks such as estimating gains from caches, configuring flow-switched networks, attack detection, and traffic matrix estimation [56, 96]. We use the data streaming and expectation-maximization algorithm proposed by Kumar et al. [96].

The accuracy metric for FSD estimation is the weighted mean relative difference (*WMRD*) between the true FSD F_l and the estimated FSD $\hat{F}_l$ [96]. The WMRD is defined as $\frac{\sum_l |F_l - \hat{F}_l|}{\sum_l \frac{F_l + \hat{F}_l}{2}}$.

Heavy-hitter detection: The goal here is to identify the top k items (e.g., srcaddr, srcport) with the most traffic volume. These are used by operators to understand application patterns and resource hogs, as well as for traffic engineering and accounting.

We use the SH algorithm [62] described earlier. We configure it to run with six instances, one each for the following flowkeys: source port, destination port, source address, destination address, 5-tuple, and source-destination address pairs. The accuracy metric is the *top-k detection rate* – the set intersection between the exact top-k and estimated top-k heavy hitters. Our minimalist approach also uses SH; the main difference is that we use only one instance of SH that runs at the 5-tuple granularity and use offline projections to the other flowkeys.

Entropy estimation: The entropy of traffic distributions (e.g., distribution of pkts per dstport) is useful for anomaly detection [100] and traffic classification [174]. In particular, entropy-based analysis captures fine-grained properties that cannot be obtained with just volume-based analysis. The entropy of a random variable X is $H(X) = -\sum_{i=1}^{N} Pr(x_i) \log_2 (Pr(x_i))$, where $x_1, \ldots, x_N$ is the range of values for X, and $Pr(x_i)$ is the probability that X takes the value x_i. It is useful to normalize the entropy between zero and one as $H_{norm}(X) = \frac{H(X)}{\log_2(N_0)}$, where N_0 is the number of distinct x_i values observed in a given measurement epoch [100].

Application	Accuracy/Error Metric	Algorithm	Parameters (defaults)
FSD estimation (5-tuple)	WMRD	[96]	fsd (0.7)
Heavy hitter detection (5-tuple,sip,dip, sport,dport sip-dip)	Top-k detection rate	[62]	hh, k $(0.3, 50)$
Entropy estimation (5-tuple,sip, dip,sport,dport)	Relative Error	[102]	ϵ, δ $(0.5, 0.5)$
Superspreader detection	Detection accuracy	[168]	K, b, δ $(100, 4, 0.5)$
Change detection (sip,dip)	falsepos + falseneg	[94]	h, k, θ $(10, 1024, 0.05)$
Deg. histogram estimation	JS-divergence	[176]	–

Table 4.1: Summary of applications, accuracy metrics, algorithms, and default parameters. The parentheses in the first column specify the flowkey(s) for the application (e.g., FSD uses 5-tuple; heavy-hitter has six flowkeys). fsd and hh are expressed as a fraction of the number of distinct IP flows per epoch. ϵ, δ denote error tolerances. K, b means that any IP contacting $\geq K$ distinct IPs is a superspreader and any IP contacting $\leq \frac{K}{b}$ distinct destinations is a false positive. h is the number of hash functions and k is the number of counters per hash function in the sketch data structure and θ is the change detection threshold.

We use the data streaming algorithm proposed by Lall et al. [102]. We consider five distributions: 5-tuple, src port, dst port, src address, and dst address. The accuracy metric is *relative error* – if the actual value is H_{norm} and the estimated value is $\hat{H}_{norm}$, the relative error is $\frac{|H_{norm} - \hat{H}_{norm}|}{H_{norm}}$.

Superspreader detection: Security applications like scan, worm, and botnet detection need to detect "superspreaders" – source IPs that contact a large number of distinct destination IPs. Note that this is different from heavy-hitter detection; we want to find sources talking to many *unique* destinations rather than sources generating a large volume of traffic.

We use the one-level superspreader detection algorithm proposed by Venkataraman et al. [168]. The algorithm has three parameters K, b, and δ; the goal is to detect all hosts that contact $\geq K$ distinct destinations with probability $\geq 1 - \delta$, and guarantee that a source that contacts $\leq \frac{K}{b}$ distinct destinations is reported with probability $\leq \delta$. The accuracy metric is the *detection accuracy*: the number of true superspreaders detected. (For brevity, we do not report the false positive rate since it was zero for the minimalist and application-specific approach in almost all cases.)

Change detection: Change detection is used to detect DDoS attacks, flash crowds, and worms [94]. At a high-level, the goal is to detect IP addresses or ports whose behavior deviates significantly from some expected behavior based on a history-based forecast model. The problem can be formally described as follows.

Suppose we bin a traffic stream into measurement epochs ($t = 1, 2, \ldots$). Let $I_t = \alpha_1, \alpha_2, \ldots$ be the input traffic stream for epoch t. Each packet α_i is associated with a flowkey a_i and a count c_i (e.g., #bytes or just 1 if we are counting packets). $Obs_a(t) = \sum_{i:a_i=a} c_i$ denotes the aggregate count for flowkey a in epoch t. Let $Fcast_a(t)$ denote the forecast value (e.g., using exponentially weighted moving average, EWMA) for item a in epoch t. The forecast error for a then is $Err_a(t) = Obs_a(t) - Fcast_a(t)$. $F2Err_t = \sum_a Err_a(t)^2$ is the second moment of the forecast errors. The goal is to detect all as with $Err_a(t) \geq \theta \times \sqrt{F2Err_t}$, where θ is a user-defined threshold. We define the *change detection accuracy* as the sum of the false positive (flowkeys whose volume did not change significantly but were incorrectly reported) and false negative rates (flowkeys that changed but were not reported).

We use the sketch-based change detection algorithm proposed by Krishnamurthy et al. [94] as sketches have a natural "linearity" property that makes them well-suited for change detection. We use an EWMA model $Fcast(t) = \alpha Obs(t) + (1 - \alpha) Fcast(t-1)$, with $\alpha = 0.9$. Note that since we are only interested in the relative performance of the minimalist vs. sketch-based approaches, the specific forecast model we use is not important.

We consider two instances to identify changes in (1) the number of packets per source address and (2) the number of packets per destination address.

Degree histogram estimation: The outdegree d of a source IP is the number of distinct IPs it contacts in a measurement epoch. We construct the degree histogram as follows. For bucket i, let m_i denote the number of sources with outdegree d such that $2^i \leq d \leq 2^{i+1}-1$. The goal is to estimate these m_i values. A specific application is to detect botnets involved in coordinated scans [176] by detecting changes in the outdegree histogram. The outdegree distribution is independently useful for understanding traffic structure. We use the sampling algorithm proposed by Gao et al. [176]. Given the exact distribution $\{m_1, m_2, \ldots\}$ and an estimated distribution $\{\hat{m}_1, \hat{m}_2, \ldots\}$, we use the *Jensen-Shannon (JS) divergence* between the two distributions as the accuracy metric.[5]

4.4.2 Assumptions and Justification

In order to compare the minimalist and application-specific approaches, we need to normalize their total resource footprints. We discuss our assumptions along three dimensions: hardware implementation, processing requirements, and memory use. We justify why our specific assumptions are *conservative* in that they underestimate the performance of our minimalist approach.

Hardware feasibility: We assume that both the application-specific algorithms and the minimalist primitives have feasible implementations that can operate at line rates. Some application-specific algorithms require a simple array of counters (e.g., [94, 176]), while others (e.g., [62, 102, 168]) and the minimalist primitives FS, SH [79, 62] involve key-value data structures. Previous work has demonstrated that it is possible to efficiently implement such key-value data structures in routers [78, 141, 111]. Also, discussions with a popular router vendor suggested that supporting FS, SH, and cSamp like primitives is within the capabilities of today's routers.

Processing requirements: There are two processing components: online collection and offline computation. By construction, the online collection overhead of the minimalist approach is lower. In the application-specific architecture, each packet requires as many counter updates as the number of application instances. (Further, each different flowkey

[5] Gao et al. [176] use the Kullback-Leibler (KL) divergence. However, it is not always well-defined. The JS divergence is based on KL divergence, but is always well-defined.

for the heavy-hitter, entropy, and change detection requires separate updates.) With the minimalist approach, each packet requires only two updates, one for FS and one for SH.[6]

We currently run estimation algorithms on the collected flow data without further sampling. Thus, the offline processing overhead of the minimalist approach could be higher because the application-specific schemes only need to process compact summaries. We believe that offline processing costs are not a serious issue, given the costs/capabilities of commodity hardware today. We do note that our estimation procedure can be augmented with additional directed sampling, if necessary, to reduce the offline overhead.

Memory consumption: Note that for FS and SH, the flow record (the IP 5-tuple and other meta-data) need not be maintained in SRAM (these can be offloaded to DRAM); only the counters (byte/packet counts) need to be in SRAM [111].

We assume a 4× overhead for maintaining flow counters as key-value pairs in SRAM for the minimalist approach as compared to a corresponding counter used by the application-specific approaches. We justify why this 4× factor is *conservative*.

1. Some application-specific algorithms we consider also require key-value counters; we conservatively assume that these incur no overhead compared to an array of counters. That is, if each entry in a counter array is 2 bytes, we assume that it takes 8 bytes to store one key-value pair for the minimalist primitives but only 2 bytes to store one key-value pair for the application-specific algorithms.

2. Suppose each counter for the application-specific algorithms is 2 bytes [187]. We ran experiments with a sparse hash data structure and found that it can store 10^6 flow counters in 8 MB, i.e., 8 bytes per counter. In other words, a *commodity, software only* implementation has just $\frac{8}{2} = 4\times$ overhead.

3. With smarter hardware for storing flow counters such as counter braids [111], the overhead will be even lower. For example, maintaining 1 million flow counters using counter braids only requires 1.4 MB of memory, i.e., an effective overhead $\frac{1.4}{2} << 4\times$.

Summarizing the above discussion, we see that

1. The hardware requirements of our primitives are similar to the application-specific algorithms.

[6] One caveat is that FS, SH, and the different application-specific approaches require per-packet processing unlike packet sampling. Again, our discussions with the router vendor suggested that this is feasible.

2. The online processing overhead of the minimalist approach is strictly lower.
3. The minimalist primitives have at most a 4× memory overhead.

Thus, for the rest of the chapter, we only consider the conservative 4× memory overhead to generate a equivalent resource configuration for the minimalist approach.

4.4.3 Configuring the Different Algorithms

Application-specific case: To configure the different algorithms, we follow the guidelines and recommended parameters from the literature:

1. The FSD estimation algorithm uses an array of $fsd \times F$ counters, where F is the number of distinct flows in a measurement interval. Following the guidelines of Kumar et al. [96], we set $fsd = 0.7$.
2. We configure the heavy-hitter detection algorithm with $hh \times F$ counters with $hh = 0.3$, divide these equally among the six instances, and focus on the top-50 detection rate.
3. The entropy estimation algorithm is an (ϵ, δ) approximation, i.e., the relative error is at most ϵ with probability at least $1 - \delta$. The number of counters it uses increases as we require tighter guarantees (lower ϵ and δ). However, Lall et al. [102] show that in practice it works well even with loose bounds. Thus, we set $\epsilon = \delta = 0.5$.
4. For superspreader detection, we set $K = 100$ and $b = 4$. Again, since loose bounds work well in practice, we set $\delta = 0.5$.
5. The sketch data structure has three parameters: h, the number of hash functions; k, the size of the counter array per hash function; and the detection threshold θ. Following Krishnamurthy et al. [94], we set $h = 10$, $k = 1024$, and $\theta = 0.05$.
6. For degree histogram estimation, we use the same configuration as Gao et al [176].

Minimalist case: The minimalist approach has two configuration parameters: the number of flow records it can collect (L) and, for ingress routers, the FS-SH split (f). To determine L, we measure the aggregate number of counters used by the different application-specific algorithms and scale it *down* by a factor of 4 as discussed earlier. We set $f = 0.8$, giving 80% of the resources to FS on each ingress router.

4.4.4 Estimation Phase in minimalist Approach

The estimation phase for the minimalist approach is conceptually simple. Since we have the actual flow records (i.e., the 5-tuples along with the packet counts), we can run exact estimation algorithms. For example, we can compute the flow size distribution of the reported flows and use that as our estimate of the true flow size distribution. Similarly, we can compute the observed (normalized) entropy of different flowkey combinations from the reported flows and use it as the estimate of the true (normalized) entropy.

The only issue is in combining the flow reports from the FS and SH components for the different estimation tasks. We use the following heuristic. First, we take the union of the flow records reported by SH (after normalizing packet counts by the sampling rate [62]) and the flow records reported by FS.[7] We compute the FSD, entropy, and detect heavy hitters or changes per-source (or destination) on this merged set of flow records. Second, we (logically) retain the set of flow records reported only by FS. We use this set for detecting superspreaders and computing the degree histogram.

Note that the minimalist approach exports the actual flow records. Thus, it is possible to run any estimation procedure on these flow records to compute any application metric, even unforeseen ones.

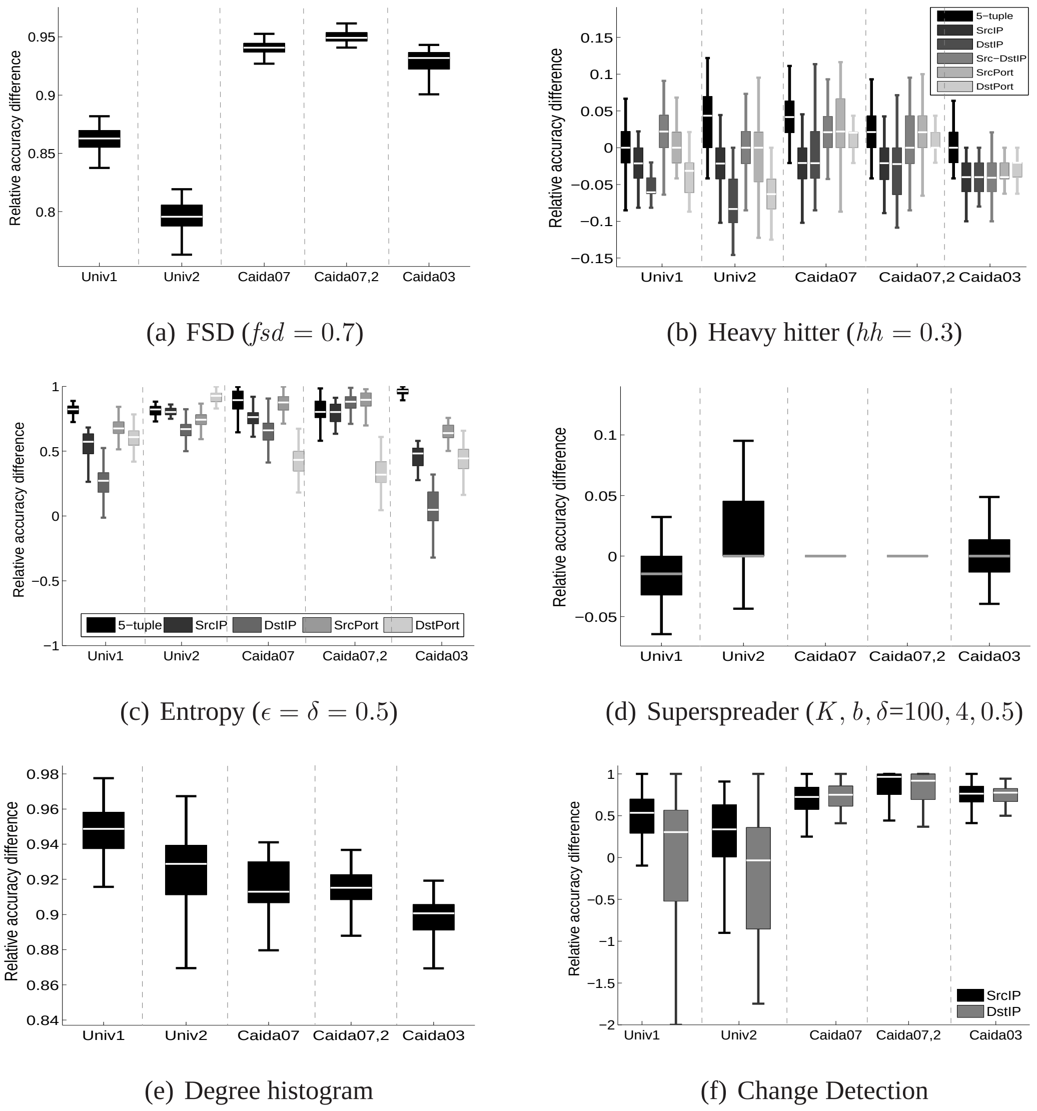
4.5 Trace-Driven Evaluation

Trace	Description	Avg # pkts (millions)	Avg # flows (thousands)
Caida 2003	OC-48, large ISP	6	400
Univ-2	UNC, 2003	2.5	91
Univ-1	USC, 2004	1.6	93
Caida 2007-2	OC-12	1.3	45
Caida 2007-1	OC-12	0.7	30

Table 4.2: Traces used in the single router experiments; averages are over 5-minute epochs

In this section, we compare the minimalist approach against the different application-

[7] If the same flow is reported by both FS and SH, we use the FS record because the packet count in FS is exact.



(a) FSD ($fsd = 0.7$)

(b) Heavy hitter ($hh = 0.3$)

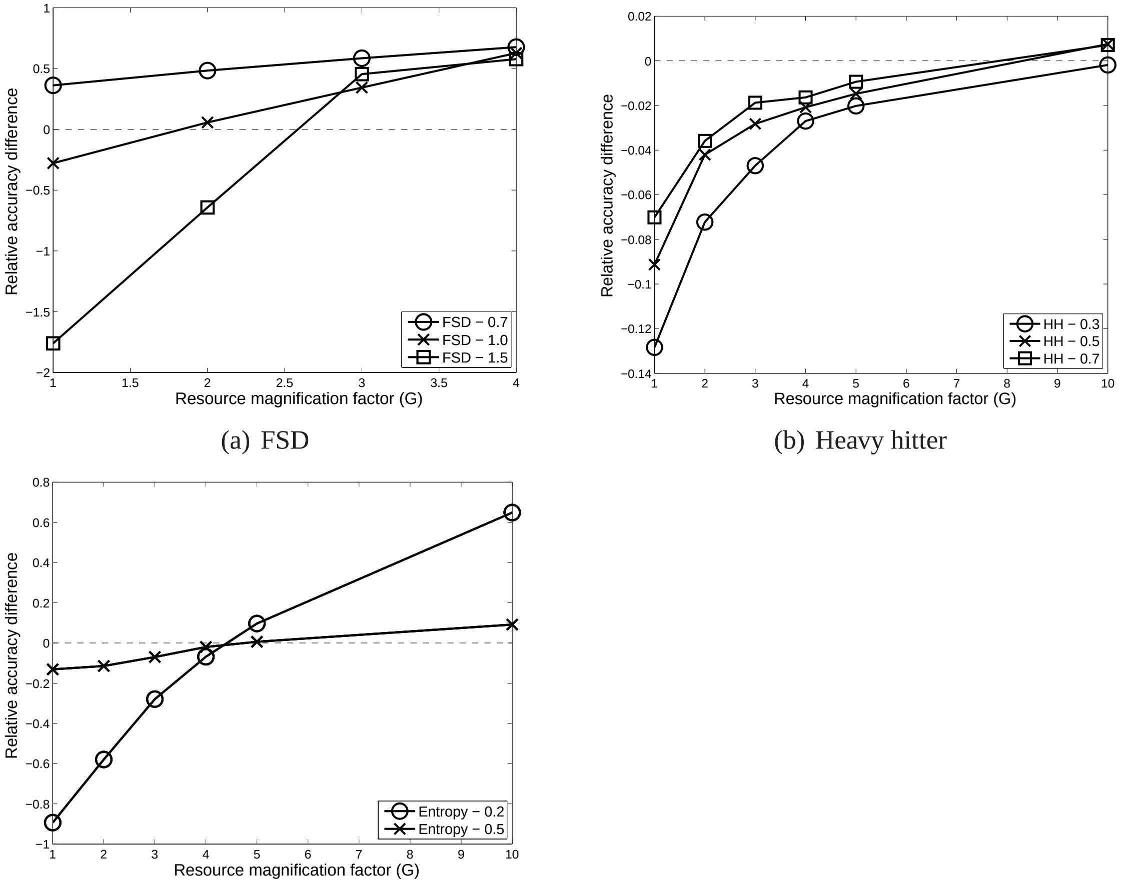
(c) Entropy ($\epsilon = \delta = 0.5$)

(d) Superspreader (K, b, δ=$100, 4, 0.5$)

(e) Degree histogram

(f) Change Detection

Figure 4.3: Each result shows a box-whiskers plot with the median, 25%ile, 75%ile, and extreme values. A positive value on the y-axis means that the accuracy of the minimalist approach was better; a negative value indicates otherwise. For most applications, the minimalist approach outperforms the application-specific alternatives. In the cases where the performance is worse, it is only worse by a small relative margin.



(a) FSD

(b) Heavy hitter

(c) Entropy

Figure 4.4: Exploring the sensitivity of applications in isolation. The zero line represents the point at which the minimalist approach starts to outperform the application-specific approach. The resource magnification factor captures the sharing effect of aggregating resources across applications.

specific algorithms using packet and flow-level traces collected from different settings. We start with a single router evaluation and then proceed to a network-wide evaluation.

4.5.1 Single Router Case

Using trace-driven evaluations, we answer the following questions:

- How does the accuracy of the minimalist approach compare with the application-specific approaches when configured with the aggregate memory used by the application-specific algorithms?
- How sensitive is individual application performance to the amount of memory available to the minimalist approach?
- How does the success of the minimalist approach depend on the set of application-specific algorithms that are implemented on the router (we call this an *application portfolio*)? That is, when does it make sense to adopt a minimalist approach instead of implementing each application-specific alternative?
- How should we split resources between FS and SH?

Table 4.2 summarizes the five different one-hour packet header traces (binned into 5-minute epochs) used for the single-router evaluation.

Accuracy: minimalist vs. Application-specific

We use the default parameters from Table 4.1 and run the minimalist approach configured with the total normalized memory used by the six algorithms. Then we compute the relative accuracy difference for each application defined as follows: Let $Acc_{specific}$ denote the accuracy of the application-specific algorithm and let $Acc_{minimalist}$ denote the accuracy of the minimalist approach for that application. The *relative accuracy difference* is $\frac{Acc_{minimalist} - Acc_{specific}}{Acc_{specific}}$. By construction, a positive value indicates that the accuracy of the minimalist approach is better; a negative value indicates otherwise.[8]

All the algorithms are inherently randomized; we present the results over five independent runs with different seeds. Figure 4.3 shows the relative accuracy difference using a

[8] Some metrics denote “error” while others denote “accuracy”. For error metrics (FSD, entropy, degree histogram, change detection) the relative accuracy as defined is negative when the minimalist approach performs better. For ease of presentation, we reverse the sign of the numerator in these cases.

box-and-whiskers plot for the different traces. Each box shows the 25%ile, median, and 75%ile values.[9]

The result shows that the median value of this metric is positive in most cases; i.e., the minimalist approach outperforms the application-specific alternative in most applications. Further, even the 25%ile is positive in many cases; i.e., the minimalist approach consistently outperforms the application-specific approaches. Only in heavy-hitter detection (Figure 4.3(b)) does the minimalist approach perform worse; even then the median accuracy gap is at most 0.08. This answers the second challenge from Section 4.2: *The minimalist approach provisioned with the total resources used by the six applications performs better than or comparable to the application-specific approaches.*

We now proceed to answer to two natural questions: (a) what if we consider each application class in isolation and (b) what types of application portfolios does the minimalist approach perform favorably in. For brevity, we only present the results from the Caida 2003 trace.

Application Sensitivity

In the following experiments, we try 2-3 configurations for each application-specific algorithm. For each configuration, we consider a minimalist approach provisioned with G times as much memory (before the $4\times$ normalization) as that used by the algorithm *in isolation*.

As before, we focus on the relative accuracy difference between the minimalist and application-specific approach. Figure 4.4(a) plots the relative accuracy difference between the minimalist approach and the FSD estimation algorithm. We show three different configurations with the FSD algorithm using $fsd = 0.7$, 1, and 1.5. For some configurations (e.g., $fsd = 1.5$, $G = 1$), the minimalist approach performs worse. The large negative values of the metric is an artifact of the low WMRD values at these points. Since we normalize the difference by the WMRD of the application-specific case, the gap gets magnified. The absolute accuracy of the FSD algorithm improves (i.e., the WMRD goes down) as it is provisioned with more resources (not shown). For example, for the configuration $fsd = 1.5$ and $G = 1$, the WMRD for the FSD EM algorithm was 0.02 and the WMRD for the minimalist approach 0.05. Both values are small for many practical purposes [96].

Figure 4.4(b) shows similar results for heavy-hitter detection, with hh set to $0.3, 0.5,$ and 0.7. For clarity, we average the relative accuracy difference across the six heavy-

[9] The whiskers extend to the most extreme data points not considered outliers. By default, this corresponds to a length of 1.5× the difference between the 25%ile and 75%ile values.

hitter instances. The minimalist approach is worse than the application-specific approach. However, as G increases, the accuracy gap closes significantly. One reason for the poor accuracy is that we configure the SH algorithm in the minimalist approach to operate at the 5-tuple granularity and then subsequently project results to other dimensions. In fact, the minimalist approach performs better if we only consider the 5-tuple granularity; it does worse for the other flowkeys due to some loss of accuracy in the projection phase (Figure 4.3(b)). We could also configure the SH algorithm in the minimalist approach to operate at multiple flowkeys. We tradeoff a small reduction in accuracy for a significant reduction in online processing overhead complexity since we only need to run one instance of the SH algorithm instead of six instances.

Entropy estimation (Figure 4.4(c)) with $\epsilon = \delta$ set to 0.2 and 0.5 and superspreader detection (not shown) show similar trends. If we consider each application in isolation, the minimalist approach performs worse. But, the gap closes as G increases and the minimalist approach eventually outperforms the application-specific algorithm.

Sensitivity to Application Portfolio

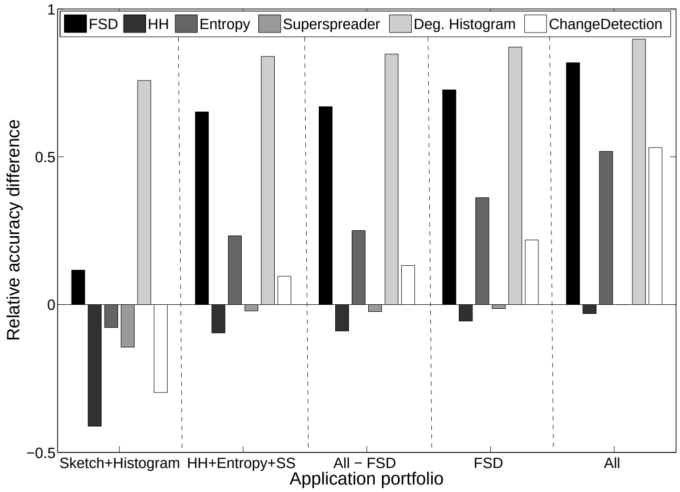


Figure 4.5: Effect of application portfolio on the relative accuracy difference. The portfolios are in increasing order of memory usage from left to right.

Next, we evaluate the effect of varying the application portfolio. That is, we consider the case where the router only implements a subset of the six applications described earlier. For a fixed portfolio, we use the default configurations from Table 4.1 and run the minimalist approach configured with the aggregate resources contributed only by the applications within this portfolio. The relative accuracies are computed with respect to the default configurations for the different applications (even for those not in the portfolio). For example, the configuration labeled "Sketch + Histogram" uses resources only from sketch-based change detection and degree histogram estimation (the most lightweight applications). At the other extreme, the configuration labeled "All" uses the aggregate resources (as in Figure 4.3).

Figure 4.5 shows the portfolios in increasing order of memory usage. For clarity, we show averages across the different flowkeys for heavy-hitter detection, entropy estimation, and change detection. We observe two effects. First, for larger application portfolios (i.e., as the requirements of management applications increase), there is a clear win for the minimalist approach. Second, if there are some resource-intensive applications (e.g., FSD estimation), then it is better to adopt a minimalist approach because it benefits all potential applications.

Split between FS and SH

So far, we fixed the FS-SH split to be $f = 0.8$. Figure 4.6 shows the effect of varying f. The x-axis is f, the fraction of resources allocated to FS. For most applications, increasing f improves the accuracy of the minimalist approach, but there is a diminishing returns effect. For heavy-hitter detection, as expected, giving more resources to SH helps, but the improvement is fairly gradual. In light of this, the 80-20 split is a reasonable tradeoff across the different application classes.

4.5.2 Network-wide Evaluation

Dataset and Setup: We use a one-hour snapshot of flow data collected across eleven routers from the Internet2 backbone. There are roughly 1.4 million distinct flows and 9.5 million packets in aggregate per 5-minute interval. We map each flow entry to the corresponding network ingress and egress points [66]. Unlike the packet traces used earlier, these are flow records with sampled packet counts (with $p = 0.01$). We assume that the sampled flow records represent the actual traffic and use the sampled counts as the actual packet counts. Also, IP-addresses in the dataset are anonymized by zero-ing out the last 11 bits. We treat each anonymized IP as a unique IP. Thus, the entropy and outdegree

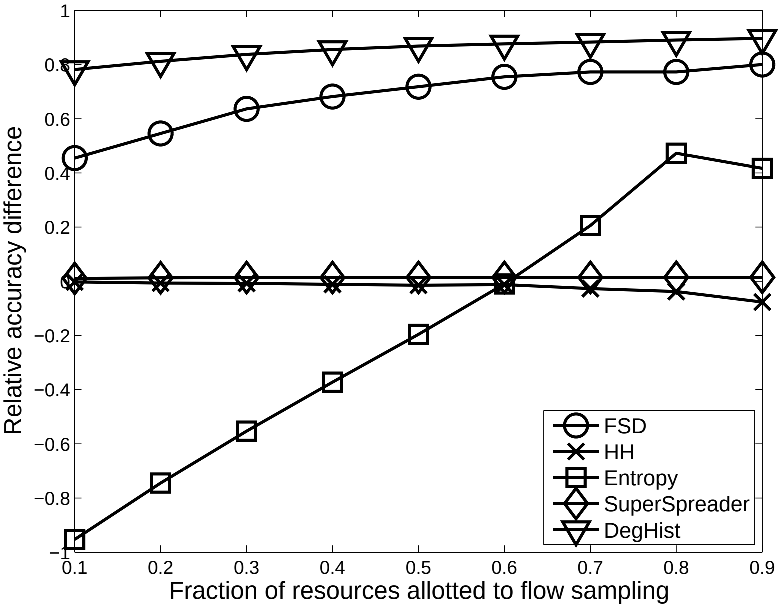


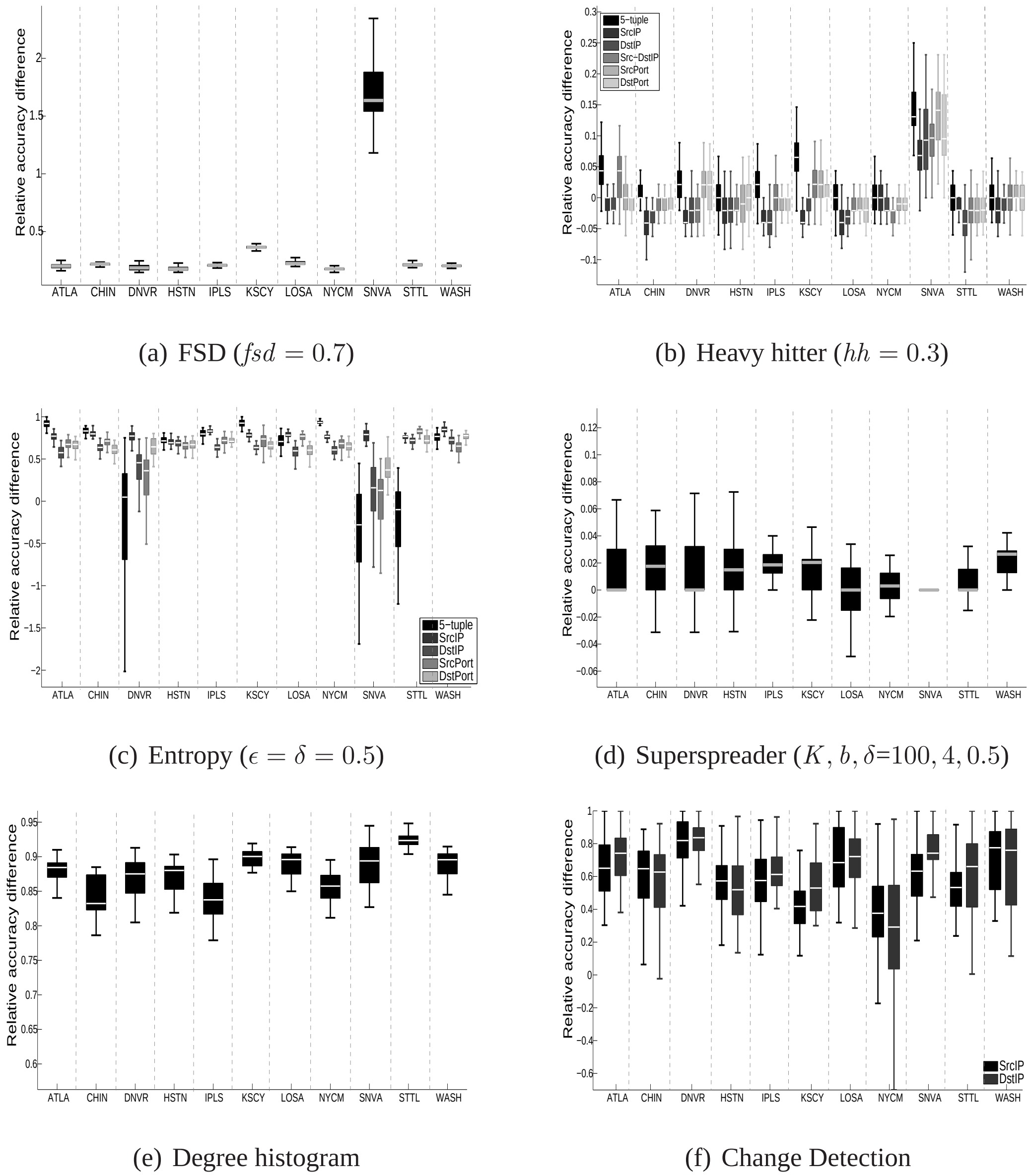
Figure 4.6: Varying the split between FS and SH

measures are computed at this granularity. Since we are only interested in the *relative* performance, this dataset is still valuable for understanding network-wide effects.

In a network-wide setting, operators often want to compute the different traffic metrics such as FSD, entropy, heavy hitters etc., over multiple *spatial views* [86, 174, 180, 100]. For example, we might want to understand traffic patterns on a per-ingress basis, or a per OD-pair basis, or over the entire network.

As such, we configure the application-specific algorithms on a per-ingress basis. That is, at each node, we run these algorithms only on packets originating from this node and ignore transit/terminating traffic. (In this topology, each node is an ingress for some traffic and there are no pure transit nodes that do not originate any traffic.) For example, the FSD algorithm at ATLA estimates the FSD for the traffic originating at ATLA and the superspreader algorithm at ATLA tracks only the source IPs that originate traffic at ATLA.

From this configuration, we obtain the total memory usage at each node. The coordinated minimalist approach from Section 4.3 operates on a per OD-pair granularity using this equivalent per-router memory (after scaling it down by the 4× normalization factor). Given the flow records for each OD-pair, we estimate the traffic metrics over three spatial views: per-ingress, per-OD, and network-wide.



(a) FSD ($fsd = 0.7$)

(b) Heavy hitter ($hh = 0.3$)

(c) Entropy ($\epsilon = \delta = 0.5$)

(d) Superspreader (K, b, δ=$100, 4, 0.5$)

(e) Degree histogram

(f) Change Detection

Figure 4.7: Result showing the relative accuracy difference between the coordinated minimalist approach and the application-specific algorithms per ingress router. A positive value indicates that the accuracy of the minimalist approach was better; a negative value indicates otherwise.

Per-ingress results: Figure 4.7 shows, for each ingress, the relative accuracy difference between the coordinated minimalist approach and the application-specific algorithms configured per ingress. Recall that a positive value indicates that the accuracy of the minimalist approach was better; a negative value indicates otherwise. As with the single router evaluation, we see that the minimalist approach outperforms the application-specific algorithms, except in heavy-hitter detection. (SNVA looks different from the others in the magnitude of the relative accuracy metric, but not in the qualitative sense that the minimalist approach is still better. While we have not been able to conclusively explain this observation, we noticed that the traffic volumes for SNVA were an order of magnitude lower than the rest. We suspect that this as a potential cause for the anomalous behavior.)

One potential concern is the high variability in the relative accuracy in some cases (e.g., DNVR and SNVA in Figure 4.7(c)). In each of these cases, we analyzed the raw accuracy values and found that the variability in fact comes from the application-specific case. That is, the accuracy of the minimalist approach has low variance, but the application-specific case has high variance.[10]

Network-wide result: Next, we consider the application metrics on a *network-wide* basis. As a point of comparison, we consider an *uncoordinated minimalist* approach. Here, each node has the same resources as the coordinated case, but independently runs FS and SH on the traffic it sees.

Given the per-ingress results for the application-specific algorithms obtained earlier, we compute network-wide estimates by merging the reports from each ingress after appropriately normalizing the per-ingress statistics. (We can do this because the per-ingress setup implicitly partitions the network-wide traffic into non-overlapping subsets. Thus, the summaries reported by the different ingresses for each application were generated over disjoint traffic subsets.) Depending on the metric, this normalization depends on the number of flows, packets, or source IPs seen at each ingress. For example, to obtain the network-wide FSD, we take the per-ingress FSD and normalize it by the number of flows originating at each ingress. However, we cannot estimate the network-wide entropy from the per-ingress entropy values as this does not give us sufficient information. For the coordinated approach, we combine the flow records obtained for each OD-pair and run the estimation procedures on this merged set of the flow records. The estimation step for the uncoordinated case is similar, but needs additional processing to remove duplicate flows.

Table 4.3 compares the application-specific, uncoordinated, and coordinated approaches

[10] The high variance in the application-specific case is not an inherent flaw–the variance decreases with more memory. But as Figure 4.5 shows, adding a few memory-intensive applications makes the case for the minimalist approach stronger.

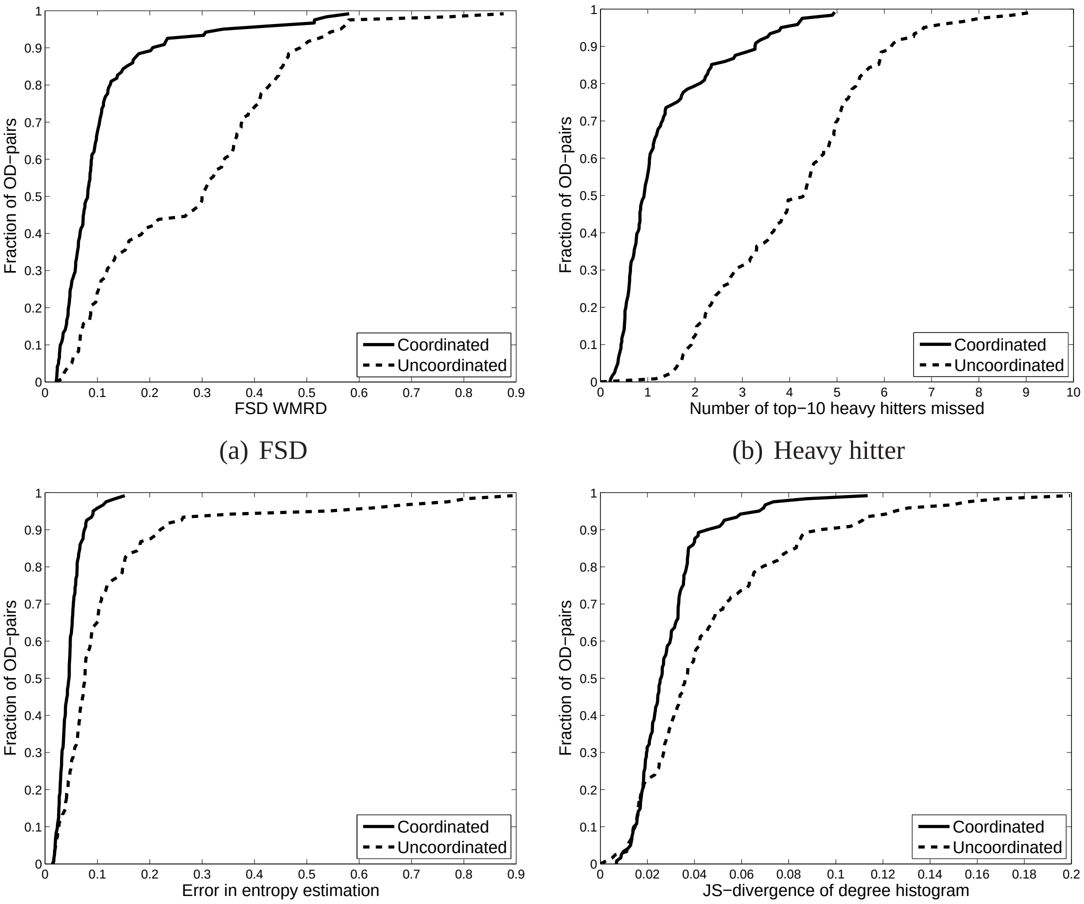
Application (error metric)	Application Specific	Uncoordinated minimalist	Coordinated minimalist
FSD (WMRD)	0.16	0.19	0.02
Heavy hitter (miss rate)	0.02	0.3	0.04
Entropy (relative error)	n/a	0.03	0.02
Superspreader (miss rate)	0.02	0.04	0.009
Deg. histogram (JS)	0.15	0.03	0.02

Table 4.3: Absolute error for network-wide metrics. Lower values imply better performance.

for the network-wide case w.r.t the *absolute error* values. (The entropy row is empty for the application-specific column because of the aforementioned reason.) There are two main observations. First, the coordinated approach has the lowest error overall. The benefits of coordination are particularly significant for the heavy-hitter and FSD estimation applications. Second, while the uncoordinated approach provides some generality (e.g., it can also provide per OD-pair estimates whereas the per-ingress application-specific algorithms cannot), it performs worse in this evaluation. One reason is that the per-ingress application-specific algorithms are implicitly coordinated and avoid ambiguity/biases when we merge the results for the network-wide case. The uncoordinated minimalist approach does not have this property and multiple sources of ambiguity/bias arise when we merge flow reports from multiple routers: (i) different routers may have different sampling rates as they see different traffic volumes, (ii) flows traversing longer paths get higher sampling probabilities, and (iii) large flows are reported multiple times by SH. An additional practical benefit of the coordinated approach is that the merging and estimation algorithms are simpler and more accurate.

Per OD-Pair results: Finally, we consider the different application metrics on a *per OD-pair* basis. Note that the application-specific alternatives as configured cannot provide per OD-pair results. They work at a coarse per-ingress level and we cannot compute the application metrics on a more fine-grained per-OD basis. This is not an inherent limitation of application-specific approaches; we can also configure them on a per-OD basis. However, this significantly increases the complexity since we need an instance per application per OD-pair. Thus, we only consider the minimalist approaches for this result.

Figure 4.8 shows four application metrics for the per OD-pair case. Since superspreader detection and change detection are meaningful only when viewed across all OD-pairs, we do not consider these. Also, we focus on the top-10 heavy hitters per OD-pair.



(a) FSD

(b) Heavy hitter

(c) Entropy

(d) Degree histogram

Figure 4.8: Comparing the coordinated and uncoordinated approaches on a per-OD basis.

The CDFs show that the coordinated approach performs well across most OD-pairs. The 80th percentile of the WMRD, heavy-hitter miss rate, average relative error in entropy estimation, and JS-divergence for the degree histogram are $0.1, 2, 0.05$, and 0.03 respectively. The corresponding results for the uncoordinated case are $0.4, 5, 0.15$, and 0.06. Further, the OD-pairs where the coordinated approach has poor accuracy have low traffic volume (not shown), which indicates that it performs very well for the dominant traffic patterns. The results for network-wide and per OD-pair views demonstrate the benefits of a systematic coordinated approach for network-wide monitoring.

4.5.3 Summary of Main Results

- The accuracy of the minimalist approach configured with the aggregate resources used by the six different applications is better than or comparable to the application-specific approaches.

- With large application portfolios or if there are one or more resource-intensive applications in the portfolio, there is a clear win for a minimalist approach vs. application-specific approaches.

- A 80-20 split between FS and SH is a reasonable tradeoff across the spectrum of applications.

- In a network-wide setting, a coordinated minimalist approach provides more flexibility and better accuracy while projecting results to different spatial views compared to uncoordinated and application-specific approaches.

4.6 Discussion

Bandwidth overhead: In the application-specific architecture, each router only reports summary estimates of the various traffic metrics (e.g., FSD, entropy). Thus the bandwidth overhead for aggregating these reports is negligible. A practical concern with our proposal is the bandwidth overhead for transferring flow records to a logically centralized collector. We give a back-of-the-envelope calculation to estimate the worst-case overhead. The Internet2 dataset has roughly 1.7GB of 1-in-100 packet sampled flow data per PoP per day. This conservatively translates into 170 GB per PoP per day or 0.6GB per five minutes for full flow capture. (This is conservative because we are normalizing the number of flows by the packet sampling rate.) Suppose, we collect this data every five minutes with a near real-time requirement that the data be sent before the start of the next five minute interval. The bandwidth per PoP required for full flow capture would be $\frac{0.6\times 8 \text{ Gbits}}{300 \text{ seconds}} = 0.016$ Gbps. Given OC-192 backbone line rates of 10 Gbps today, it is not unreasonable to expect ISPs to use 0.16% of the network bandwidth per PoP for measurement traffic to aid network management.

Adaptation: Another natural question is how does our minimalist approach deal with network dynamics. Estan et al. [61] and Keys et al. [88] have in-depth discussions on how to adapt the sampling rates for packet sampling, FS, and SH to changing traffic conditions.

Chapter 2 discussed how cSamp can adapt to network dynamics. We can leverage these existing techniques to make the minimalist approach robust to network dynamics.

4.7 Chapter Summary

This chapter is a reflection on recent trends in network monitoring. There is a growing demand for estimating a wide variety of traffic metrics to support different network management applications. The inadequacy of current packet-sampling-based solutions has led to the proliferation of many application-specific algorithms, each catering to a narrow application.

In contrast to these application-specific alternatives, we revisit the case for a minimalist architecture for flow monitoring. Such an architecture dramatically reduces router complexity and enables router vendors to focus their energies on building efficient implementations of a small number of primitives. Further, it allows late binding to what traffic metrics are important, thus insulating router implementations from the changing needs of flow monitoring applications.

This chapter demonstrated a proof-of-concept minimalist approach that combines flow sampling, sample-and-hold, and cSamp. We saw that this approach performs favorably across a wide spectrum of applications compared to application-specific approaches. Our proposal is by no means "optimal" or the final word in this problem space–the goal of this chapter was to demonstrate the *feasibility* of a minimalist approach. In this respect, there are three avenues for future work: (i) developing better minimalist primitives, (ii) designing estimation algorithms that optimally leverage the data collected across different primitives, and (iii) providing formal models to reason about application requirements and performance. We hope that our work motivates further research in these directions.

Chapter 5

SmartRE: A System for Coordinated Network-Wide Redundancy Elimination

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., [6, 5, 3, 19, 7]) and their rapidly growing adoption [8, 20].

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. [30] 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 [5, 3]. 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 [30].

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 chapter, 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. [30], 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 achieving 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.

This chapter presents 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 [30], 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 [158, 30], 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 [147] to facilitate coordination across interior routers with low 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 [122]. 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× better than naively extending a single-vantage point RE solution to the network-wide case. Further, 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.

5.1 Background and Related Work

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.

5.1.1 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 [171]. In similar spirit, CDNs and peer-to-peer caches [18, 2] 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. [158], several commercial vendors have introduced WAN optimizers which remove duplicate content from network transfers. Many of these products [6, 3, 19, 7] work at the level of chunks inside objects and we refer to them as *chunk-level* approaches. In contrast, both Spring et al. [158] and Anand et al. [30] 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., [80, 142]), content distribution (e.g., [164, 131, 127]), and distributed file systems (e.g., [25]). These approaches use fingerprinting mechanisms [132] 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.

5.1.2 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.

5.1.3 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 [158]. 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 [30]. 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 chapter 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. 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.

5.2 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.

5.2.1 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 $k = 3$ [30].

The encoding throughput for a standalone RE device is *at most* R/F packets per second. This is because each packet, whether it can be encoded or not, requires F random accesses to determine if there are any matches. 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.

5.2.2 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., [39, 122]) 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.

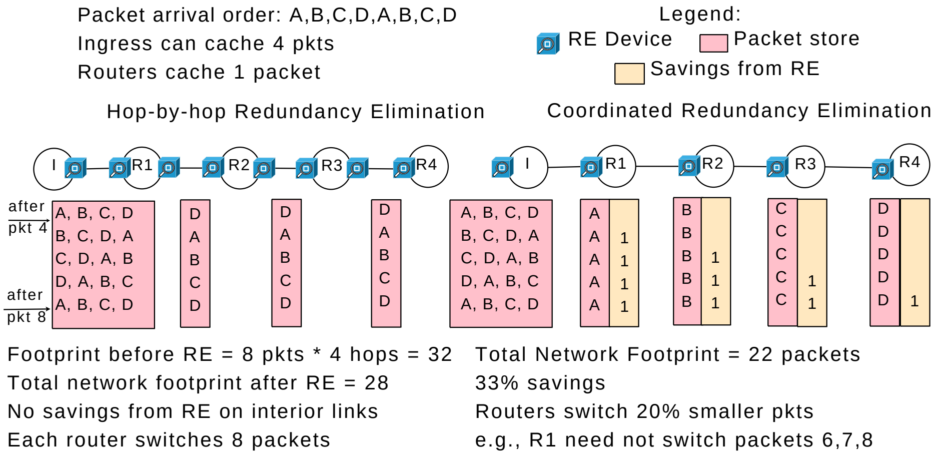

Packet arrival order: A,B,C,D,A,B,C,D
Ingress can cache 4 pkts
Routers cache 1 packet
Legend:
RE Device
Packet store
Savings from RE
Hop-by-hop Redundancy Elimination
Coordinated Redundancy Elimination
I
R1
R2
R3
R4
after pkt 4
after pkt 8
A, B, C, D
B, C, D, A
C, D, A, B
D, A, B, C
A, B, C, D
D A B C D
A A A A A
B B B B B
C C C C C
D D D D D
1 1 1 1
1 1 1
1 1
1
Footprint before RE = 8 pkts * 4 hops = 32
Total network footprint after RE = 28
No savings from RE on interior links
Each router switches 8 packets
Total Network Footprint = 22 packets
33% savings
Routers switch 20% smaller pkts
e.g., R1 need not switch packets 6,7,8

Figure 5.1: Benefits of a coordinated approach when RE devices have constraints on memory size.

Memory efficiency and router benefits: Consider the scenario in Figure 5.1. 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 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 use a given amount of memory more effectively.

Memory access constraints: Consider the example shown in Figure 5.2. 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 Section 5.2.1 where $P = 5$). The encoding RE device at the fast links can perform 10 packet encodings per second (this corresponds to case #2 from Section 5.2.1 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. As-

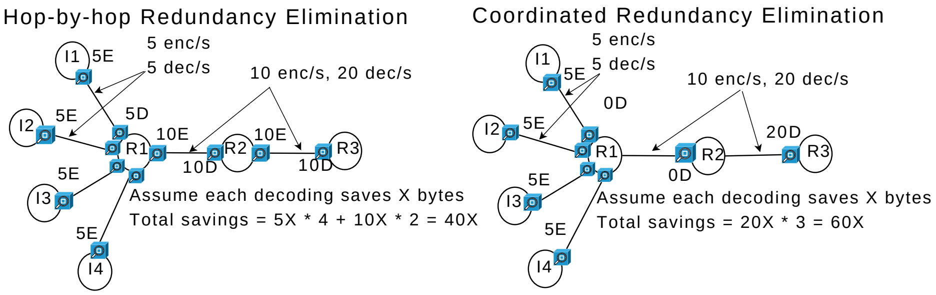

Hop-by-hop Redundancy Elimination
I1
5E
5 enc/s
5 dec/s
10 enc/s, 20 dec/s
I2
5E
5D
10E
R1
R2
10E
R3
10D
10D
5E
I3
Assume each decoding saves X bytes
Total savings = 5X * 4 + 10X * 2 = 40X
5E
I4
Coordinated Redundancy Elimination
I1
5E
5 enc/s
5 dec/s
0D
10 enc/s, 20 dec/s
I2
5E
20D
R1
R2
R3
5E
0D
I3
Assume each decoding saves X bytes
Total savings = 20X * 3 = 60X
5E
I4

Figure 5.2: Benefits of coordination when RE devices have constraints on encoding/decoding throughput.

suming 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 5.2, 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.

5.3 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.

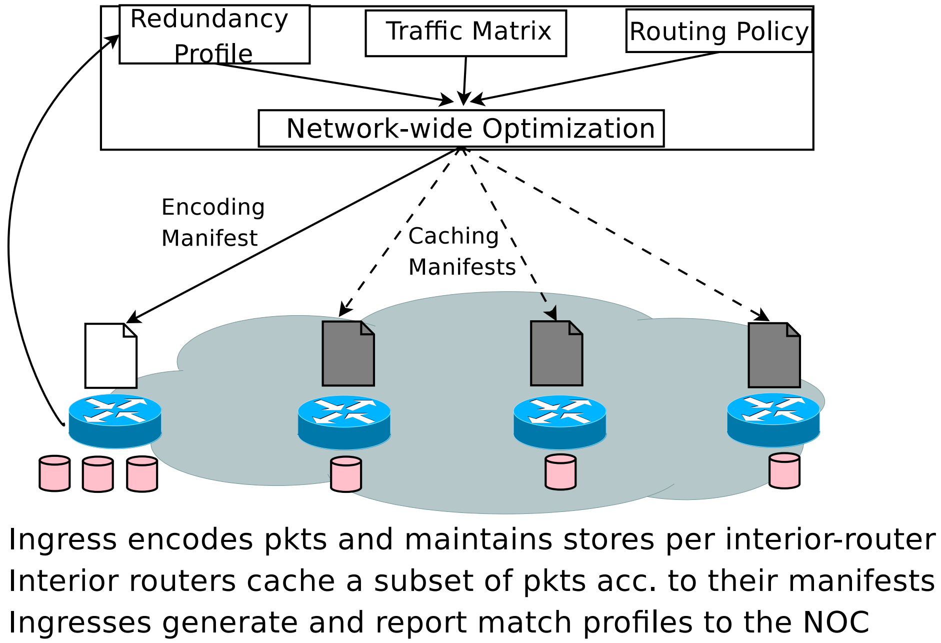

Redundancy Profile
Traffic Matrix
Routing Policy
Network-wide Optimization
Encoding Manifest
Caching Manifests
Ingress encodes pkts and maintains stores per interior-router
Interior routers cache a subset of pkts acc. to their manifests
Ingresses generate and report match profiles to the NOC

Figure 5.3: Schematic depiction of SmartRE.

SmartRE synthesizes two ideas: packet caches for redundancy elimination [158, 30] and cSamp [147]. 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.

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.

5.3.1 System Overview

We focus our discussion on the design of three key elements (Figure 5.3): 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 Section 5.1. In this sense, the role of an ingress node is identical in the 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., [73, 43, 37]), 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.

5.3.2 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:

$$\forall r, \sum_{p:r \in p} d_{p,r} \times v_p \leq M_r \tag{5.1}$$

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

$$\forall r, \sum_{p,r \in p} d_{p,r} \frac{v_p}{avgpktsize} + \sum_{p,q:r \in p,q} d_{q,r}\ match_{p,q} \leq L_r \tag{5.2}$$

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

[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.

$$\forall p, \sum_{r:r\in p} d_{p,r} \leq 1 \tag{5.3}$$

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.

$$S_{p,r} = \sum_{q:r\in q} d_{q,r} \times distance_{p,r} \times match_{p,q} \times matchlen_{p,q} \tag{5.4}$$

The objective then is to maximize $\sum_p \sum_r S_{p,r}$. Note that maximizing this objective, subject to the constraints captured by Eqs (5.1)–(5.3) 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.

5.3.3 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 Section 5.4.1).

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 $d^*_{p,r1} = 0.2$, $d^*_{p,r2} = 0.3$,

```
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 Section 5.4)
3   $candidates \leftarrow$ GetCandidates($pathid$)
    // $encodedpkt$ carries the shim header (Figure 5.5)
4   $encodedpkt \leftarrow$ Encode($pkt, candidates$)
    // Steps 5–7 are for caching
    // what is $\sum_{r \in \textsc{Path}(pathid)} d_{pathid,r}$ 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 5.4: Pseudocode for an ingress node in SmartRE.

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 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 [54].

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 Section 5.4.

Figure 5.4 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 5.5). The shim header has 2 parts: a fixed length path identifier field specifying the

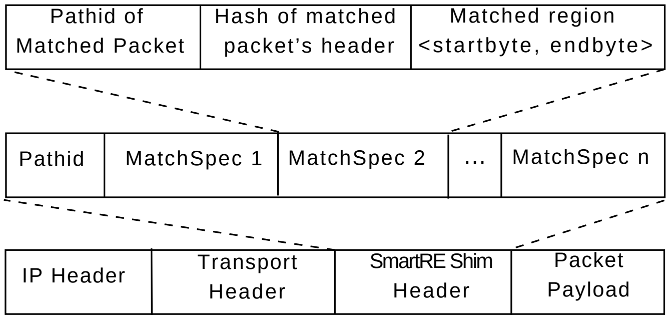


Figure 5.5: Format of the SmartRE shim header.

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.

Decoding at interior nodes: Figure 5.6 shows the algorithm at an interior node. The node reads the shim header and checks if any of the matches correspond to 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.

5.4 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

[3] If interior nodes can get the pathid from MPLS labels or routing information, this is not necessary.
[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 5.6: Pseudocode for an interior node in SmartRE.

(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. 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? That is, will that previous packet be available in a cache on the path taken by the current packet? (Section 5.4.1)

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?

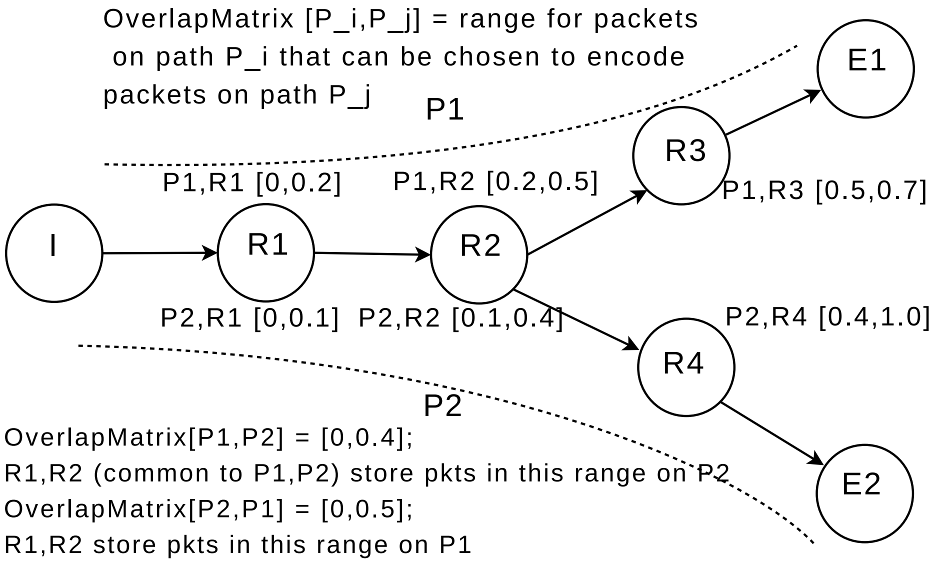

OverlapMatrix [P_i,P_j] = range for packets on path P_i that can be chosen to encode packets on path P_j
P1
E1
R3
P1,R1 [0,0.2]
P1,R2 [0.2,0.5]
P1,R3 [0.5,0.7]
I
R1
R2
P2,R1 [0,0.1]
P2,R2 [0.1,0.4]
P2,R4 [0.4,1.0]
R4
P2
E2
OverlapMatrix[P1,P2] = [0,0.4];
R1,R2 (common to P1,P2) store pkts in this range on P2
OverlapMatrix[P2,P1] = [0,0.5];
R1,R2 store pkts in this range on P1

Figure 5.7: Example showing the overlap matrix.

That is, if an ingress encodes a packet, will the decoders have the required matched packets or would they have evicted them? (Section 5.4.2)

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? (Section 5.4.3)

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.

5.4.1 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 5.4, the function GETCANDIDATES checks $overlapmatrix$ to find valid encodings.

Figure 5.7 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 $\textsc{Hash}(B) \leq \mathit{overlapmatrix}[\mathit{P1}, \mathit{P2}]$, 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 (Section 5.3.3). The configuration module generates the $\mathit{overlapmatrix}$ 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 $\mathit{overlapmatrix}$ and also simplifies the process by which the ingresses identify valid encodings.

5.4.2 Using Cache Buckets for 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 5.4 and 5.6 – 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.

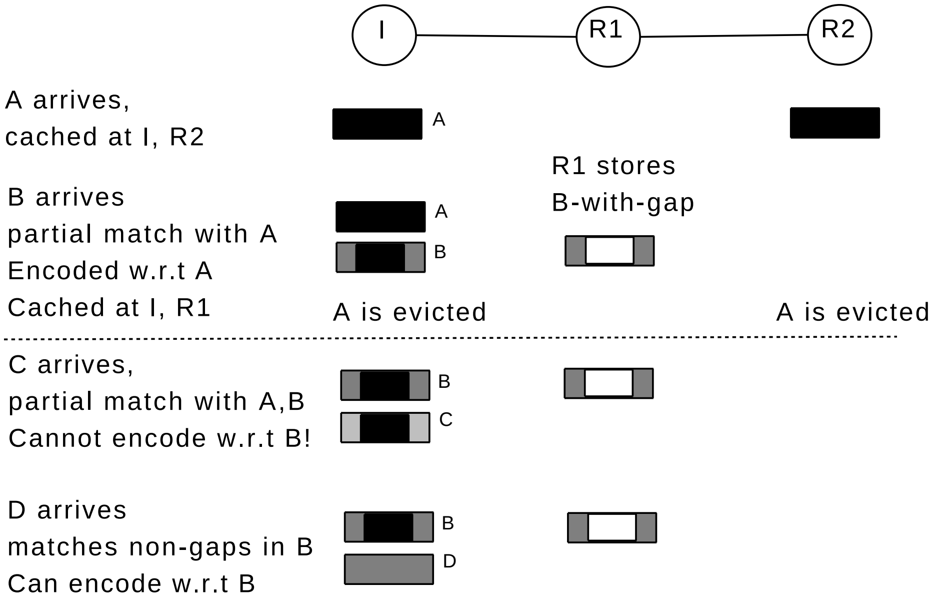


Figure 5.8: Example of how decoding gaps may occur.

5.4.3 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 5.8. 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. Our experiments with real packet traces showed that with the second option, the loss in RE is less than 3%.

5.5 Implementation Issues

5.5.1 Encoder and Decoder Implementation

We implement the encoding and decoding algorithms from Section 5.3.3 and Section 5.4 in Click [122]. 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 [158, 30]).

Like most RE systems, we use Rabin fingerprinting [132]. Each Rabin fingerprint captures a fixed 64 byte region in a packet [30]. 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 (Section 5.4.2). 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 5.5). 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 [158, 30], 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 Section 5.4.) 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 Section 5.3.3. The last decoder on a path clears the flag in the header indicating that the packet has been fully decoded.

5.5.2 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 [140]. 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 [73], 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 [149]) 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 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 without affecting correctness. This is an acceptable operating mode for SmartRE and we evaluate it further in Section 5.6. (2) Significant changes in the redundancy profile may increase the decoding load on each node (Section 5.3.2, Eq (5.2)) and affect feasibility. To handle (2), each ingress tracks the actual number of matches per interior node to avoid overloading nodes with decoding responsibilities. Thus, changes in redundancy profiles do not affect correctness.

[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).

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.

5.5.3 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.

5.6 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.

Network	PoP-level		Router-level	
(AS#)	# PoPs	Time	# Routers	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 5.1: LP solution time (in seconds).

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 [157] and add four access routers to each PoP to obtain router-level topologies.

5.6.1 Performance Benchmarks

LP solution time: Table 5.1 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 [17]. 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 [82] 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. [30]: (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 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) 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 5.2. The decoding store size was set to 600MB. We see that decoding is roughly 3-4× 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 $\geq$ 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.

# 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 5.2: Trade-off in redundancy and decoding throughput with number of match-specs.

5.6.2 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.

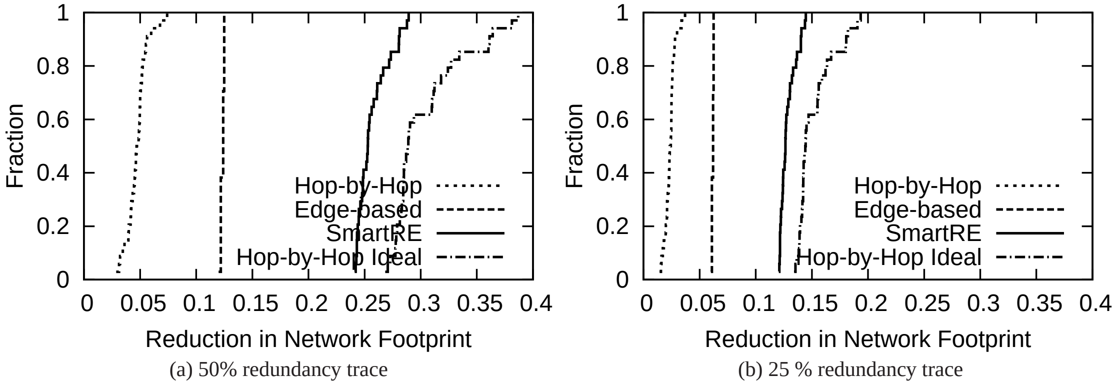
Encoders at each access link store T seconds of packets (e.g., 3 GB memory at 2.4 Gbps implies $T = 10$s). Decoders at the edge have the same cache size as the encoders. Each interior RE device uses a 6GB cache which we believe is practical in terms of cost; we also evaluate the effect of varying cache 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 prototype. Assuming a (conservative) memory latency of 100ns, 20 lookups for encoding, and 4 lookups for decoding, this translates into 0.5 million encodings and 2.5 million decodings per second respectively.

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/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 [23].

Our main metric of interest is the fractional reduction in the network footprint (Section 5.3). Figure 5.9 shows the 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× 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.

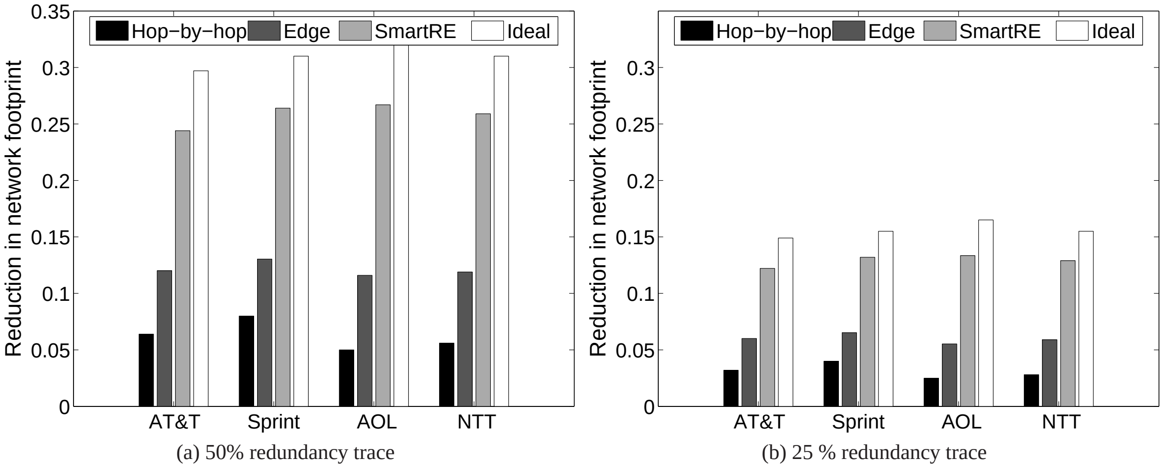
Figure 5.10 shows the network-wide reduction for 4 tier-1 ISPs. Here, we consider the top 20 PoPs (by degree) in each topology, and assume that the total traffic entering each of the 80 ingresses (4 per PoP) is the same. For simplicity, we also assume that the redundancy profile is the same across all 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



(a) 50% redundancy trace

(b) 25 % redundancy trace

Figure 5.9: CDF of network footprint reduction across ingresses for Sprint (AS1239) using synthetic traces.



(a) 50% redundancy trace

(b) 25 % redundancy trace

Figure 5.10: Network-wide footprint reduction for four tier-1 ISP topologies using synthetic traces.

the unconstrained ideal case and provides 80-90% of the ideal savings.

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 Section 5.3.2.

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 5.3 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 5.11); 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

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 5.3: 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 5.4: Exploring different redundancy profiles on the Sprint topology, with total redundancy $\gamma = 0.5$.

not optimal in realistic settings with actual resource constraints. Further, there are other practical difficulties in extending SmartRE to allow overlapping caches (see Section 5.7).

Varying redundancy profiles: Table 5.4 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 [30] can additionally boost the performance of SmartRE.

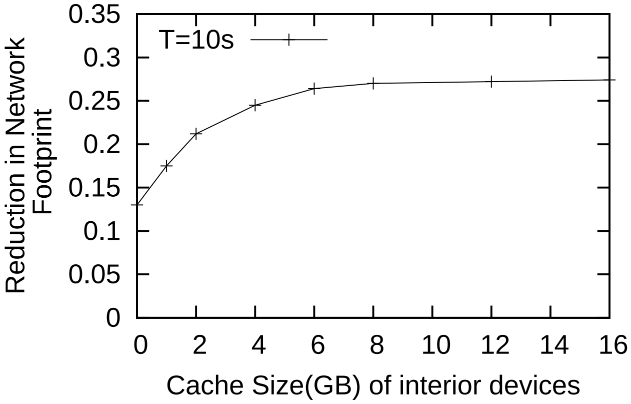


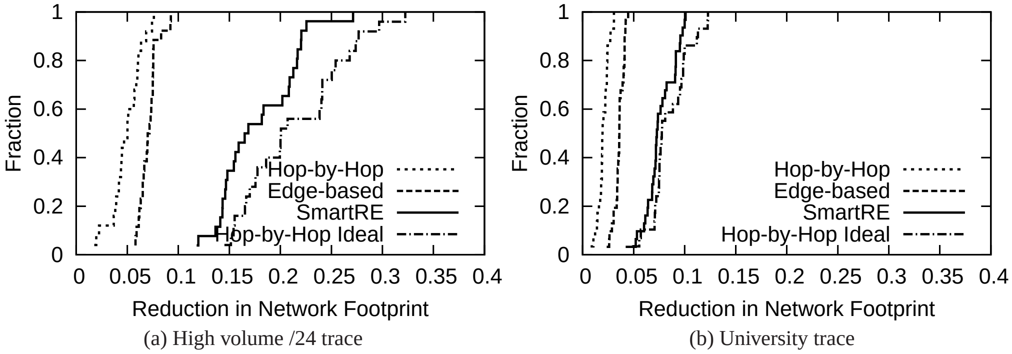
Figure 5.11: Varying cache size in the interior using a synthetic trace over the Sprint topology.

Memory provisioning: Figure 5.11 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.

5.6.3 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 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 5.12 shows the CDF of footprint reduction



(a) High volume /24 trace

(b) University trace

Figure 5.12: CDF of network footprint reduction across ingresses on Sprint topology extrapolating from real traces.

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.

5.6.4 Effect of Stale Redundancy Profiles

As discussed in Section 5.5, 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 Section 5.6.3 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 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 Section 5.5.

5.6.5 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 consider a scenario 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

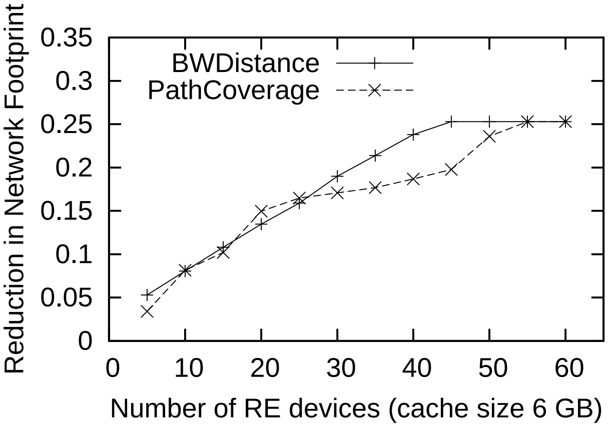


Figure 5.13: Two partial deployment strategies on the Sprint topology (x=65 represents full deployment). Each device has a 6GB cache.

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 5.13 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 5.13 indicates that even simple strategies for partial deployments work well. This can be further enhanced by weighing each path with the expected amount of redundancy based on historical observations.

5.6.6 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 assigning 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.

5.7 Discussion

Multi-hop wireless: We believe that SmartRE can be used to enhance caching systems in other contexts, e.g., multi-hop wireless networks [52]. 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 Section 5.6.2 that allowing caches to overlap may improve RE performance. However, there are two practical difficulties. First, the formulation from Section 5.3.2 becomes more complicated. Specifically, we can no longer model the second term in Eq (5.2) and the savings term in Eq (5.4) 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.

5.8 Chapter Summary

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 an IP-layer service within ISP networks.

This chapter 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× 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.

Chapter 6

Network-Wide Deployment of Intrusion Detection and Prevention Systems

Intrusion detection (NIDS) and prevention systems (NIPS) serve a critical role in detecting and dropping malicious or unwanted network traffic. These have been widely deployed as perimeter defense solutions in enterprise networks at the boundary between a trusted internal network and the untrusted Internet. This traditional deployment model has largely focused on a single-vantage-point view of NIDS/NIPS systems, placed at manually chosen (or created) chokepoints to provide coverage for all suspicious traffic.

Increasingly, however, the challenges of scaling this approach are becoming evident. Due to growth over time in both traffic and the types of analyses, these NIDS/NIPS placements become a bottleneck. Approaches to scaling single-vantage-point solutions have focused on building NIDS/NIPS clusters (e.g., [166]). The cluster approach, however, faces its own challenges: Since each packet might be relevant to multiple analyses for which the relevant state exists on different cluster nodes, these solutions need to replicate traffic across different cluster nodes or otherwise share the relevant analysis state. This results in overheads that limit the performance of these solutions or, if performance cannot be sacrificed, that force guaranteed coverage to be relaxed (e.g., [155]). This limitation is further exacerbated by the growing deployment of NIDS and NIPS functions in ISP networks, in order to provide security services to customers who may not have the necessary resources or expertise to protect their network infrastructure [36, 34].

In this chapter, we explore a different design alternative to scaling NIDS/NIPS. Instead of trying to scale processing at a few chokepoints, our approach exploits the existing replication of each packet along its forwarding path. In doing so, we depart from the single-vantage-point strategy, and permit the different nodes on a packet's forwarding path to be candidates for performing the needed analysis on the packet. As in the cluster solution, stateful analysis will require that certain types of packets be subjected to certain types of analysis at the same node — e.g., connection-oriented analysis will process packets on each direction of the connection at the same place. Rather than explicitly replicating a packet or derived state to the nodes that need it for analysis, we will partition the analysis across locations where a packet can already be observed.

The focus of this chapter is the problem of managing the deployment of NIDS and NIPS functions throughout a network. There are three key challenges in this context:

- **Resource constraints:** NIDS/NIPS solutions are constrained by the processing and memory capabilities of the underlying hardware. Additionally, some solutions use specialized capacity-constrained hardware (e.g., for line-rate string matching) to reduce the performance impact on benign traffic.

- **Placement affinity:** NIDS/NIPS are not monolithic systems: they consist of multiple modules that analyze different traffic patterns. In particular, the modules may have topological constraints on where they will be most effective. For example, outbound scans and inbound floods are best detected close to network gateways.

- **Network-wide objectives:** Network administrators have high-level policy goals to optimally utilize their NIDS/NIPS deployments toward their security objectives. For example, in the NIDS case we may want to avoid overloading specific nodes. Similarly, we want to enable NIPS functions throughout the network to maximally drop unwanted traffic.

We believe these challenges are best addressed by taking a *network-wide coordinated* approach for the deployment of NIDS/NIPS functions [37, 43, 73, 147]. We outline our specific contributions next.

NIDS: For the NIDS case, we design a framework for partitioning NIDS functions across a network to ensure that no node is overloaded. This takes into account the resource footprints of each NIDS component, the capabilities of different nodes, and placement constraints specifying where each function is most effective (e.g., ingress nodes are best suited for scan detection). We demonstrate a proof-of-concept implementation of a network-wide coordinated NIDS using Bro [129]. Our evaluations show that augmenting Bro with the coordination capabilities adds little memory or processing overhead for most modules. We emulate a network-wide deployment scenario and find that such coordination can reduce the maximum processing load by 50% and the maximum memory load by 20%.

NIPS: For NIPS, we show how to maximally reduce unwanted traffic without affecting the performance of benign traffic. We model the use of specialized and power-intensive hardware with limited capacity (e.g., content addressable memories). In these scenarios, the problem of optimally dropping unwanted traffic is NP-hard and we design practical approximation schemes. Using extensive evaluations on real ISP topologies, we show that our approximation algorithms provide near-optimal performance, achieving more than 92% of the optimal possible performance in dropping unwanted traffic. We also demonstrate the promise of leveraging techniques from online learning to combat strategic adversaries who try to evade these defenses [85].

There are several efforts for scaling NIDS and NIPS (e.g., [42, 166, 70, 156, 104]) that focus on building better single-vantage-point solutions. Because our work focuses on the network-wide aspect it effectively complements technical advances in these areas as it enables administrators to optimally utilize their current hardware infrastructure toward their security objectives.

6.1 NIDS Deployment

In this section, we first describe an abstract model that captures the constraints and requirements in deploying NIDS functions throughout a network. Next, we set up an optimization framework that assigns NIDS responsibilities across different network nodes such that no single node is overloaded. We describe a prototype implementation and evaluation using the `Bro` system [129].

6.1.1 System Model

Modern NIDS are not monolithic systems. They are comprised of modules that perform different types of traffic analyses. For example, popular NIDS like Snort and Bro implement modules for scan detection, analyzing HTTP traffic, tracking IRC traffic, finding malware signatures, etc. We abstract the functions performed by these modules into the notion of *classes*, where each class C_i is a specific type of analysis. Associated with each C_i is a specification $\mathcal{T}_i$ of the traffic of interest for analysis using C_i. For example, if C_i is a type of analysis for port-80 traffic, then $\mathcal{T}_i$ specifies all traffic to or from port 80 (on any host) that traverses the network.

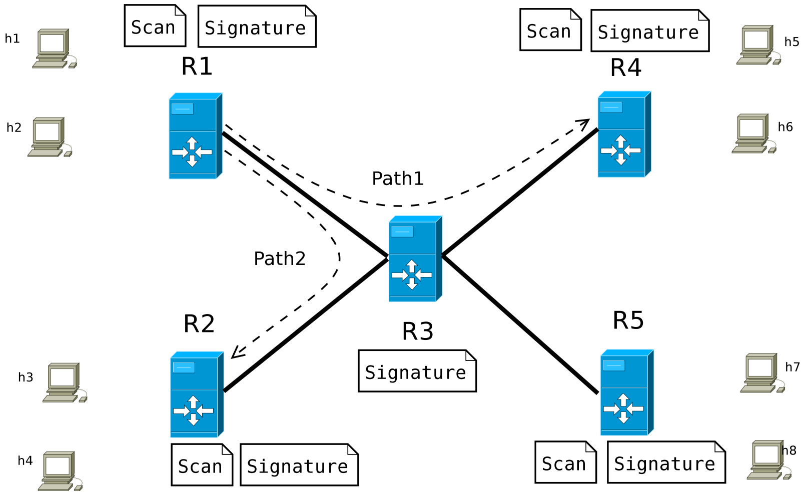


Figure 6.1: Example of network-wide NIDS instrumentation

Let $\{\mathcal{T}_{ik}\}_k$ denote a partition of $\mathcal{T}_i$ into component specifications, in the sense that any packet matching $\mathcal{T}_i$ matches exactly one $\mathcal{T}_{ik}$. We consider only classes C_i for which the associated specification $\mathcal{T}_i$ can be partitioned into $\{\mathcal{T}_{ik}\}_k$ in such a way that for every k, all traffic matching $\mathcal{T}_{ik}$ can be observed by each member of a nonempty set P_{ik} of nodes. That is, if node $R_j \in P_{ik}$, then R_j can observe *all* traffic that matches $\mathcal{T}_{ik}$ (and can recognize it as such). We call each P_{ik} a *coordination unit*. Intuitively, P_{ik} is the set of nodes that are eligible for performing analysis of type C_i on traffic matching $\mathcal{T}_{ik}$.

To make this concrete, consider the example network in Figure 6.1. Suppose there is a class C_i denoted `Signature` that applies malware signature analysis to traffic $\mathcal{T}_i$. Suppose that $\mathcal{T}_i$ is partitioned into specifications $\{\mathcal{T}_{ik}\}_k$ according to the end-to-end path it traverses; e.g., $\mathcal{T}_{i1}$ specifies the traffic traversing Path1, and similarly for $\mathcal{T}_{i2}$. Then, $P_{i1} = \{\text{R1}, \text{R3}, \text{R4}\}$ is the set of nodes that can observe (and, we assume, recognize) traffic matching $\mathcal{T}_{i1}$, and $P_{i2} = \{\text{R1}, \text{R3}, \text{R2}\}$ is the analogous set for $\mathcal{T}_{i2}$. Similarly, consider a scan detection module C_i denoted `Scan` that checks if any of the hosts h1–h8 show signs of anomalous scanning activity. In this case, the traffic $\mathcal{T}_i$ is partitioned into eight blocks $\{\mathcal{T}_{ik}\}_{k=1}^{8}$, corresponding to traffic initiated by each of the eight hosts. Because only each host's corresponding ingress node sees all the traffic the host initiates, we define $P_{i1} = P_{i2} = \{\text{R1}\}$ (for hosts h1–h2), $P_{i3} = P_{i4} = \{\text{R2}\}$, and so forth.

Because every node $R_j \in P_{ik}$ can observe all traffic in $\mathcal{T}_{ik}$, it is possible to divide the analysis of $\mathcal{T}_{ik}$ traffic across all of them, in order to disperse the analysis work across them. For example, Figure 6.1 shows enabling `Signature` on all the nodes on the network; as we will see, we will do so in a way that each node $R_j \in P_{ik}$ analyzes a distinct subset of the $\mathcal{T}_{ik}$ traffic.

We use T_i^{pkts} and T_{ik}^{pkts} to denote the total traffic volumes in packets that matches $\mathcal{T}_i$ and $\mathcal{T}_{ik}$, respectively. Moreover, a type of analysis C_i performs analysis at some level of traffic aggregation (e.g., sources, destinations, flows[1], or sessions). As such, we use T_i^{items} and T_{ik}^{items} to denote the total traffic volumes, expressed in the unit of aggregation appropriate for C_i (e.g., flows), that matches $\mathcal{T}_i$ and $\mathcal{T}_{ik}$, respectively.

6.1.2 Problem Formulation

Next, we describe the optimization problem that allows us to assign NIDS responsibilities in a network-wide fashion.

Objective: The goal is to assign monitoring responsibilities to different nodes such that the processing/memory load is balanced (for a suitably defined balancing function). For example, we may want to minimize the maximum load or make sure that the load is evenly distributed. While assigning these responsibilities, we must ensure that the traffic is *covered* completely. This is the correctness requirement to ensure that the network-wide deployment will be logically equivalent to running a single NIDS on the entire traffic.

Control Variables: d_{ikj} denotes the fraction of traffic in C_i on coordination unit P_{ik} that R_j processes. That is, in Figure 6.1, we can split the `Signature` analysis responsibilities *fractionally* across R1, R3, and R5. We consider a fractional split for two reasons. First, this is the most general formulation possible and thus will yield the best solution. Second, the fractional split allows us to model the optimization problem as a linear program, that can be solved efficiently using solvers like `CPLEX`.

Inputs: We assume that the network administrators provide the following parameters based on their specific infrastructure, NIDS requirements, and traffic patterns as inputs to the optimization:

- The various NIDS classes $\{C_i\}_i$ and, for each C_i, its coordination units $\{P_{ik}\}_k$. T_{ik}^{pkts} and T_{ik}^{items} specify the volume of packets and items (e.g., flows, sources) for C_i traversing P_{ik}.

[1] A flow is a sequence of packets close in time that have the same IP source and destination addresses/ports and protocol.

- The different classes may have different resource footprints. For each C_i, we capture these using the per-packet processing load (e.g., CPU seconds per packet) $CpuReq_i$ and the memory load $MemReq_i$ (e.g., bytes per flow or per source). These can be obtained by profiling the resource consumption of the NIDS for different modules [77].

- The processing and memory capacity $CpuCap_j$ and $MemCap_j$ of each node R_j. We consider a general model in which the network elements could have heterogeneous hardware capabilities.

Optimization problem: For concreteness, we focus on minimizing the maximum processing/memory load on any given node across the network, while guaranteeing complete coverage over the different NIDS classes. This optimization problem can be represented using the following linear programming formulation.

$$\text{Minimize } \max\{CpuLoad, MemLoad\}, \text{ subject to}$$

$$\forall i,\ \forall k,\ \sum_{j:R_j \in P_{ik}} d_{ikj} = 1 \tag{6.1}$$

$$\forall j,\ MemLoad_j = \frac{\sum_i \sum_k MemReq_i \times T_{ik}^{items} \times d_{ikj}}{MemCap_j} \tag{6.2}$$

$$\forall j,\ CpuLoad_j = \frac{\sum_i \sum_k CpuReq_i \times T_{ik}^{pkts} \times d_{ikj}}{CpuCap_j} \tag{6.3}$$

$$\forall j,\ CpuLoad \geq CpuLoad_j \tag{6.4}$$

$$\forall j,\ MemLoad \geq MemLoad_j \tag{6.5}$$

$$\forall i, \forall k, \forall j,\ 0 \leq d_{ikj} \leq 1 \tag{6.6}$$

Eq (6.1) says that the all the traffic in each coordination unit for each class should be monitored. Eq (6.2) models the total memory load on each node, expressed as a fraction of its memory capacity. As a first-order approximation, the memory load depends on T_{ik}^{items}, the number of distinct items corresponding to this analysis [77]. For example, this would be the number of flows in per-flow analysis and the number of distinct source addresses in per-source analysis. Eq (6.3) models the processing load on each node expressed as a fraction of its processing capacity. Again, we model the processing footprint as a function of the total volume (in packets) of each class that the node is assigned [77]. Finally, we model the maximum memory and processing load across all the nodes, and minimize the max of these two metrics.

Output: We solve the linear program to generate *sampling manifests* that specify the monitoring responsibility for each node R_j. These responsibilities are specified in terms of hash-ranges for each coordination unit P_{ik}.

The d_{ikj} values in the optimal solution can be converted into hash-range based sampling manifests for each P_{ik} using the procedure in Figure 6.2. The main idea is that we map the fractional variables into non-overlapping hash ranges while generating the sampling manifests for each node. The non-overlapping hash ranges ensure that each node $R_j \in P_{ik}$ analyzes a distinct subset of the $\mathcal{T}_{ik}$ traffic, without requiring any explicit communication between the different R_js.

Given a sampling manifest, the algorithm on a node R_j is shown in Figure 6.3. As each packet arrives, we find the corresponding NIDS modules that will analyze this packet. In general, the same packet may be analyzed multiple modules; e.g., a packet on port 80 may be analyzed by the HTTP, malware signature detection, and scan detection modules. For each such module, we check if R_j should run the corresponding analysis for this packet. To do so, we compute a HASH from the packet header using a lightweight hash function. Depending on the semantics of the analysis, the hash is computed over specific subsets of the packet header. For example, for flow-based analysis, the hash uses the unidirectional 5-tuple. For session-based analysis, the hash is computed over a bidirectional 5-tuple such that the source/destination IP are consistent for both directions of the session. If the hash falls into the hash-range assigned to node R_j for coordination unit P_{ik}, then this packet is subjected to analysis by class C_i at R_j.

6.1.3 Implementation in Bro

We implement the above coordination functions in the Bro IDS [129]. Bro is logically divided into two parts (Figure 6.4): (1) an *event engine* that converts a stream of packets into higher-level events and (2) a site-specific *policy engine* that operates on the event stream.

Bro maintains a *connection record* for each end-to-end session that is generated in the event engine and carried into the policy engine. This connection record keeps the basic state information regarding the source/destination, application ports, and other tags associated with the connection. We modified the connection record to additionally carry the hashes of different combinations of the connection fields. Adding these to the connec-

```
GenerateNIDSManifest($d^* = \langle d^*_{ikj} \rangle$)
foreach class $C_i$ do
    foreach coordination unit $P_{ik}$ do
        $Range \leftarrow 0$
        // the order of nodes does not matter
        foreach $j, R_j \in P_{ik}$ do
            $HashRange(i,k,j) \leftarrow [Range, Range + d^*_{ikj}]$
            $Range \leftarrow Range + d^*_{ikj}$
// Assignments across Classes and Coordination units
$\forall j, Manifest(R_j) \leftarrow \{\langle \{i,k\}, HashRange(i,k,j) \rangle | d^*_{ikj} > 0\}$
```

Figure 6.2: Translating the optimal solution into a sampling manifests for each NIDS node

tion record increases the memory footprint slightly, but avoids having to recompute the hashes within each policy script. We use the Bob hash function recommended by prior measurement studies [121].

We consider two implementation alternatives: (1) delaying the sampling checks in Figure 6.3 (specifically, line 5 for each i and k) until the policy engine stage and (2) implementing the sampling checks in the event engine as early as possible. The first approach has two advantages. First, it requires minimal changes inside the event engine (except adding the hashes to the connection record). Second, it pushes the coordination intelligence into the *site-specific* configurations as intended in the Bro system design. However, we found (Section 6.1.4) that this induced significant overhead for some modules. This is because the policy scripts are executed by an interpreter and doing hash lookups/checks is quite expensive. In (2), we add the sampling checks and only initialize a module if necessary. For example, we initialize the HTTP module for a session only if the session hash falls in the range assigned to this node for HTTP processing. Fortunately, we do not need to modify each such module to add these checks. We need to add this check only at two places: (a) when application-protocol modules (e.g., HTTP, IRC) are initialized (based on port numbers)[2] and (b) in the event engine for the signature matching module.

For some modules, the only processing that occurs is in the policy stage. For example, scan detection and TFTP processing receive a raw event stream reporting connection in-

[2] Port numbers are not robust for determining application behavior–Bro can also detect application behaviors dynamically. In that case, we can implement this check at the point where the corresponding application-specific module is initialized.

```
CoordinatedNIDS($pkt, R_j, Manifest(R_j)$)
{$C_i$}$_i$ ← GetClass($pkt$)
// Each packet may be analyzed by multiple modules
foreach class $C_i$ do
    $k$ ← GetCoordUnit($pkt, i$)
    // Hash returns a value in $[0, 1]$
    // Specific packet fields used for Hash
    // depend on semantics of $C_i$
    $h_{pkt}$ ← Hash($pkt, i$)
    if $h_{pkt} \in HashRange(i, k, j)$ then
        Run class $C_i$ for $pkt$
```

Figure 6.3: Coordinated NIDS algorithm on node R_j

formation. In this case, our only option is to implement the sampling check in the policy engine.

In both (1) and (2), we implement the common functions to process site-specific configurations and sampling manifests. We assume that the network administrator provides site-specific configurations that will map each packet matching $\mathcal{T}_{ik}$ to the corresponding P_{ik}. For example, these could map IP prefixes to their ingress locations or identify the routing paths for a given pair of IP prefixes.

6.1.4 Evaluation

First, we describe our evaluation setup. Then, we use standalone microbenchmarks to profile the resource footprints of the different modules and measure the overhead of our modified Bro prototype. Finally, we describe an emulated network-wide evaluation that shows the benefits of a coordinated network-wide approach vs. a single vantage point approach.

Setup: We use a custom traffic generator that takes in as input a network topology, the traffic matrix (fraction of traffic for each ingress-egress pair), routing policy (nodes on each ingress-egress path), and a traffic profile (e.g., relative popularity of different application ports). Additionally, we provide *template sessions* for different applications using

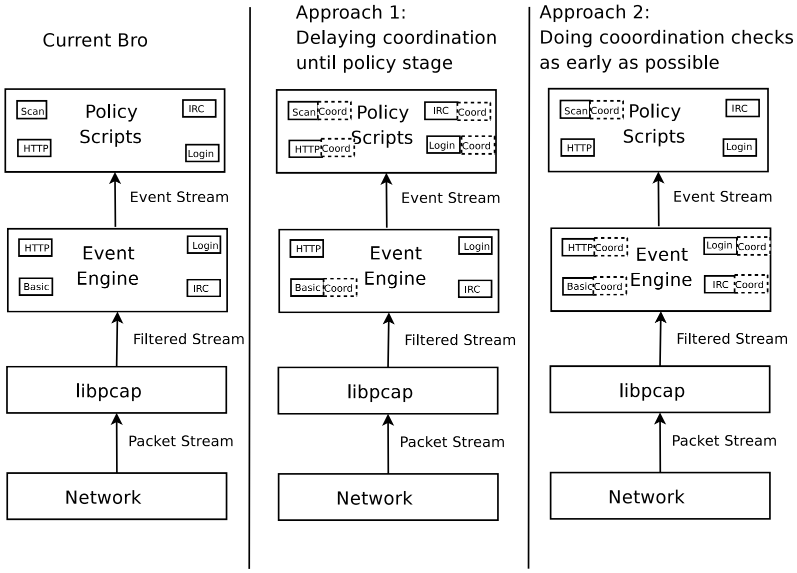


Figure 6.4: Implementing the coordination functionality in Bro. The "coord" boxes indicate where changes were needed to add in coordination checks in Bro. For some modules (e.g., `Scan`), the coordination checks have to be in the policy engine.

real traffic captured for common protocols like HTTP, IRC, Telnet etc., and synthetically generated traffic sessions for other protocols.

The goal of this evaluation is to compare the relative performance (processing, memory load) of a network-wide coordinated approach against a current single vantage point approach. By design, the network-wide approach provides the equivalent functionality. (We verified through manual inspection of Bro logs and profiles that the aggregate behavior of the network-wide and standalone approaches are equivalent. We do not present these results for brevity.) That is, we are not interested in the detection accuracy of the IDS algorithms as such. To this end, our traffic trace generator provides a realistic mix.

The performance benchmarks we present next were obtained using Bro-1.4 on a dual-CPU Intel Pentium 3.4GHz machine with 2GB RAM running Ubuntu 9.04.

Microbenchmarks: First, we perform a standalone evaluation (i.e., with no network-wide coordination) of our prototype implementation and compare it with an unmodified Bro system. We generate a single traffic trace with a total of 100,000 traffic sessions using a mixed traffic profile that stresses different modules. We evaluate both implementation alternatives described earlier: Bro with the coordination checks implemented in the event engine wherever possible, and Bro with all coordination checks in the policy scripts. The sampling manifests in both cases are configured to specify that this standalone node needs to process all the traffic. We setup Bro so that it runs each analysis module in isolation.

Our goal is to evaluate: (a) the processing overhead induced by the coordination functions — identifying the coordination unit, computing the hashes, and checking if the hashes lie in the appropriate sampling ranges; and (b) the memory overhead of adding the hash values into the connection record.

Figure 6.5 shows the processing overhead for our Bro implementations relative to an unmodified Bro system (using the total CPU time used reported by Bro) across these modules. For the Baseline, Signature, Blaster, and SYN-flood scenarios, the overhead of coordination checks is around 2% on average for both implementations. For the scan and TFTP modules, the overhead of both coordinated versions is close to 10% since these involve more processing in the policy engine. In these cases, both the coordinated versions have very similar overhead because the coordination checks occur in the same place; they cannot be offloaded to the event engine (e.g., scan, TFTP etc.) or they occur solely in the event engine (e.g., Signature). However, in the case of HTTP, IRC, and Login, we observe a significant overhead when we perform the coordination checks in the policy engine.

Figure 6.6 shows that the memory overhead of the coordinated versions is at most 6%. Recall that this overhead arises because we augment the connection record in the event and policy engines to carry hashes of different fields in the connection identifier.

Network-wide evaluation: Next, we consider a network-wide evaluation setup. For this, we use the Internet2 topology with 11 nodes distributed throughout the continental US to represent a large enterprise network with several locations. We use a gravity model based on the city populations to determine the traffic matrix; i.e., the split of the total traffic between every pair of locations. We use shortest-path routing based on link distances to determine the paths for traffic between each pair of locations. Given this topology and traffic information, we set up the linear programming formulation to assign the NIDS responsibilities across the different locations to minimize the maximum CPU/memory load on any given location. We assume that all the locations have the same processing/memory capabilities. We use the guidelines of Dreger et al. [77] to generate the per-packet and per-flow/per-source resource footprints for the different Bro modules.

We compare the network-wide coordinated deployment against an edge-only deployment where each location independently runs a Bro instance on the traffic it sees. We emulate a network-wide deployment as follows. From a network-wide trace, we generate traces that each node sees. For the coordinated case, this includes both traffic originating/terminating at a node and transit traffic. For the edge-only case, these consist of

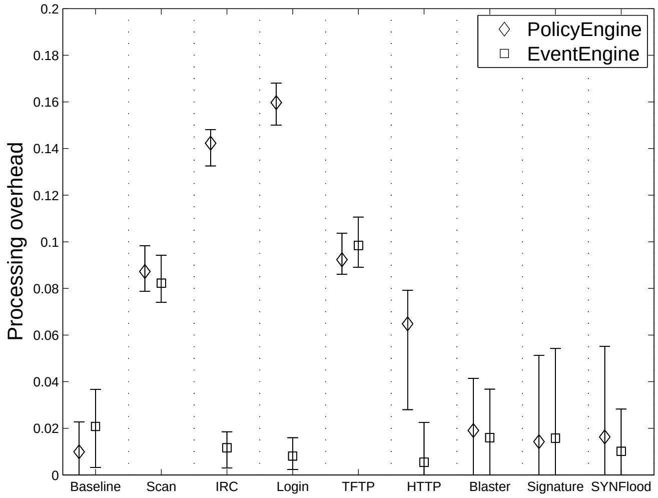


Figure 6.5: CPU overhead with the coordination-enabled Bro prototypes for different modules

traffic originating/terminating at each node. Given these traces, we run Bro on the trace in pseudo-realtime emulation mode. During each run, we measure the CPU utilization and memory load using `atop` sampled every 1 second. We report the CPU footprint as the product of the utilization and the total execution time and the memory footprint in terms of the maximum resident memory size. For each deployment scenario and node, we run the experiment 5 times to report the mean, minimum, and maximum value of these performance metrics.

Figures 6.7 and 6.8 show the maximum per-node memory and processing load across the 11 node network as a function of the total network traffic volume. Here, we increase the total number of end-to-end sessions while keeping the traffic matrix and the NIDS functionality fixed. The NIDS modules in this case are the 8 modules from Figures 6.5 and 6.6. We see that coordination reduces the maximum memory footprint by 20% and the maximum CPU footprint by 50%. The overall trend also shows that the network-wide approaches scales better as the workload increases. Interestingly, we see that even though the memory overhead of the coordinated versions in the policy and event-engine based checks are similar (Figure 6.6), the results are significantly different in the network-wide case (Figure 6.7). The reason is that delaying the coordination checks until the policy engine negates any benefits that the network-wide optimization offers. This is because each node has to keep per-protocol connection state even if it is not logically responsible for analyzing that connection.

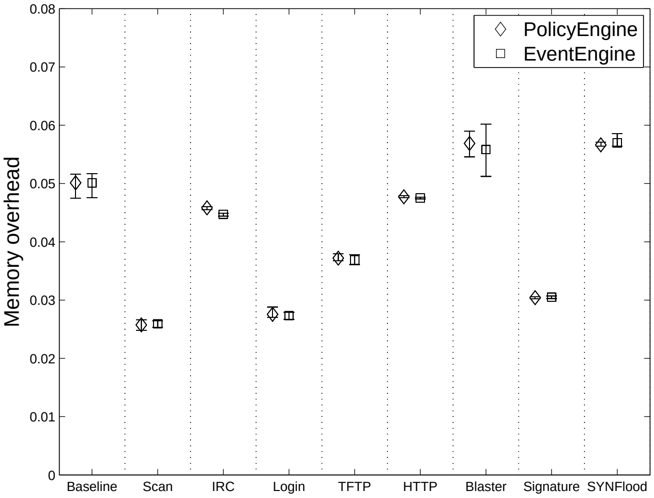


Figure 6.6: Memory overhead with the coordination-enabled Bro prototype for different modules

Next, we consider the effect of adding more functionality to the NIDS. For this experiment, we keep the traffic volume fixed at 100,000 flows, but add more NIDS modules by creating one or more duplicate instances of the analysis modules seen so far. In order to simulate the effect of adding more NIDS functionality, we create duplicate instances of HTTP, IRC, Login, and TFTP modules.[3] Recall that there were two classes of modules: those where we could push most of the coordination functions into the event engine and others where we could not. We manually inspected around 140 Bro policy scripts provided in the default distribution and found that a majority of them fall in the former category. Thus, our duplicate instances are indicative of how a NIDS like Bro would be configured with additional modules in practice.

Figures 6.9 and 6.10 show the effect of increasing the number of NIDS modules. Again, we see that the coordinated approach scales better as we add more functionality into the NIDS deployment.

Finally, to provide insights into how these performance benefits arise, we show how the CPU and memory load metrics vary across the different network locations in Figures 6.11 and 6.12. We see that in the edge-only deployment, the node marked 11 is most loaded. (This corresponds to New York, which in a gravity model based traffic matrix carries

[3] We used fake instances merely for convenience. This let us avoid having to benchmark and modify scripts for other modules.

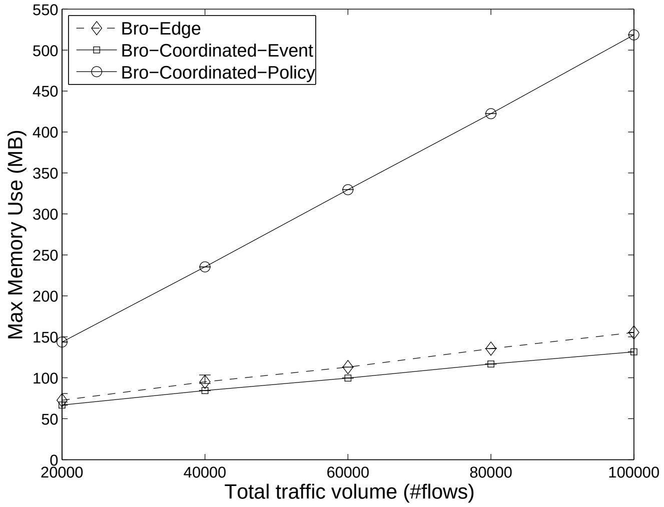


Figure 6.7: Max memory usage across the network as the total traffic volume increases

a significant volume of traffic.) These also show that the coordinated case effectively balances the load across the different nodes– it offloads some responsibilities that were previously assigned to node 11 to other nodes where the same analysis could have been performed with no loss in functionality. For example, we see that some nodes (e.g., nodes 6 and 8) have to perform more NIDS responsibilities than before.

6.1.5 Extensions

More fine-grained coordination capabilities: These results show that our coordinated Bro prototype already provides significant performance benefits in a network-wide setting. However, there are some avenues to further improve the performance.

The basic unit of processing in the Bro event engine is a connection: an end-to-end session between two hosts. This means that the Bro instance at the node 11 in our setup has to track all connections, because it is the only node that can run the `Scan` module. Even though a lot of the processing has been offloaded to other nodes, it has to track all packets because a connection is the smallest granularity of processing. Thus, we have to duplicate the baseline connection processing work across the network.

One direction of future work is to systematically design NIDS to support fine-grained coordination capabilities–allowing different granularities of connections, creating more

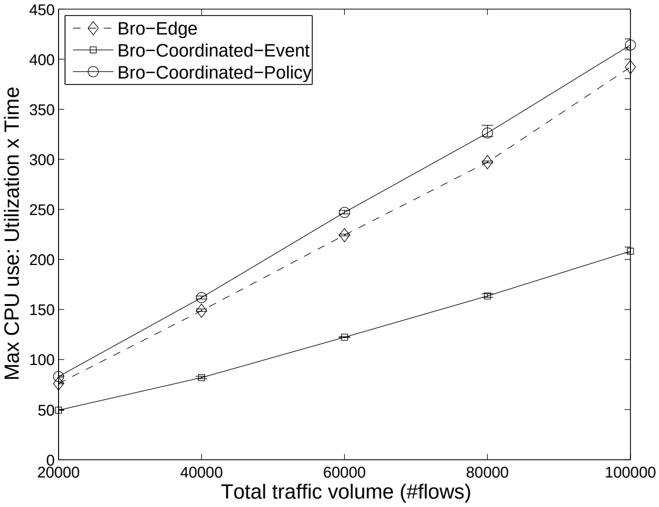


Figure 6.8: Max CPU usage across the network as the total traffic volume increases

fine-grained events (e.g., first packet of a flow for `Scan`), allowing modules to specify how early we can implement the coordination checks etc.

Redundancy for reliability: In order to be robust to NIDS failures, network administrators may want to ensure that each analysis module is enabled at k or more distinct locations for each coordination unit. We are specifically concerned about non-adversarial failure modes; e.g., hardware or OS crashes. (If we are running the same NIDS implementation at all locations, this does not protect against adversaries who craft traffic patterns to target specific implementation bugs.)

Extending our model from Section 6.1.1, this means that we have to divide the hash space for each coordination unit across the nodes such that: (1) each point in the space is covered k times and (2) no node is responsible for the same point more than once. The second clause ensures that we have k *distinct* nodes to analyze each packet/connection.

One approach is to add another dimension to the formulation to incorporate the notion of a redundancy level. That is, we can extend the d_{ikj} to d_{ikjl} to indicate what redundancy level this corresponds to. But it is intuitively hard to capture the constraint in (2) that the same node is never responsible for the same point in the space more than once in this model. At first look, it seems that incorporating such reliability demands is hard.

Fortunately, there is a simple extension to the LP formulation to meet this requirement. The key is not to treat replicated coverage in terms of levels, but simply as fractions of a

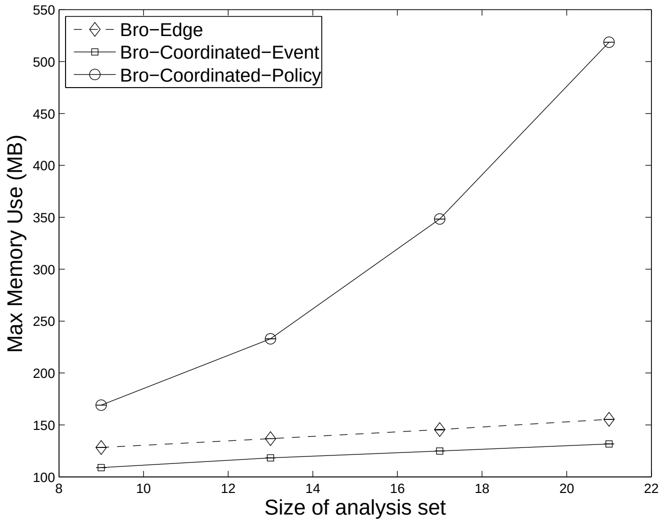


Figure 6.9: Max network-wide memory use with more modules

larger space. That is, instead of thinking of the problem in terms of covering the space $[0, 1]$ k times, we think of it as covering the space $[0, k]$, wrapping around at integral values. We modify the RHS of the constraint Eq (6.1) to k instead of 1 and solve the rest of the LP as before. While converting the LP solution into sampling manifests (Figure 6.2), we proceed as before, except that we logically wraparound the range every time it exceeds 1.

6.2 NIPS Deployment

In this section, we first describe our model to capture the constraints and requirements in deploying NIPS functions. We describe the optimization problem, show that it is NP-hard, and develop approximation algorithms based on randomized rounding techniques. We evaluate these algorithms on a range of real and inferred ISP topologies and system parameters. Finally, we describe how we can extend the model to be robust to dynamic adversaries by leveraging techniques from online algorithms.

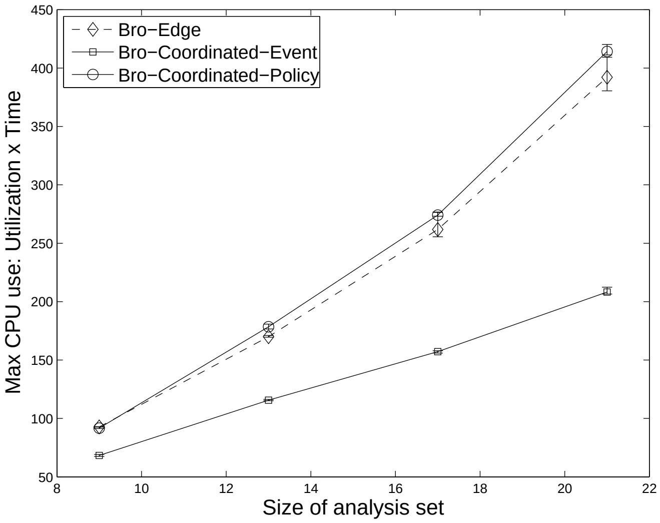


Figure 6.10: Max network-wide CPU use with more modules

6.2.1 System Model

We consider a general model of NIPS that include firewalls and signature-based detection systems. NIPS typically consist of *filtering rules*, each matching a specific traffic pattern. For example, firewall rules look at the packet header fields; signature-based filters detect specific string/regular expression patterns in packet payloads. As in the NIDS case, each rule (class) C_i is associated with two types of resources: (1) CPU processing load $CpuReq_i$ per packet, and (2) memory load $MemReq_i$ if it needs to maintain any per-flow or cross-packet state. For this discussion, we restrict our presentation to rules that operate a per-packet or per-flow granularity, since it is typical of most NIPS functions used today. As such, we consider only coordination units that are end-to-end routing paths; i.e., each P_{ik} is a path of routers.

Unlike the NIDS case, NIPS operate on the *forwarding path* and need to strictly operate at (or close to) the line rate. Many firewalls and payload detection mechanisms today use special purpose hardware such as Ternary CAMs (TCAM) for pattern matching in order to operate at line rates (e.g., [179, 178]). However, such hardware capabilities are expensive and power-hungry. This places additional economic and technological limits (imposed by power and cooling requirements) on how many NIPS modules can be active on each node and adds a new dimension where not all rules can be enabled on all NIPS nodes. To address this concern, we extend our model from the previous section.

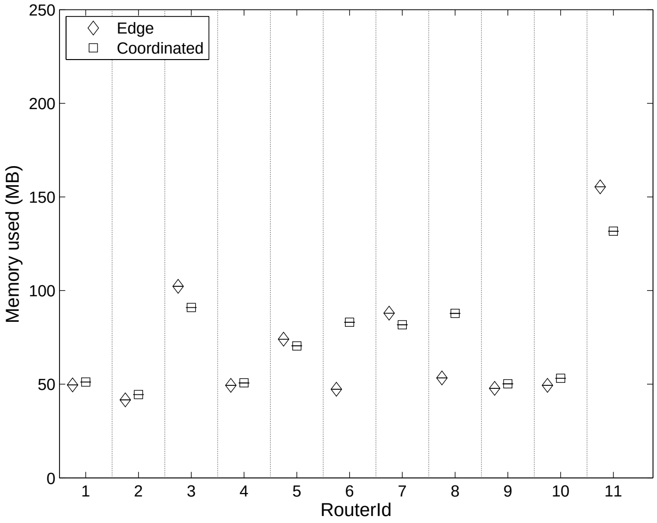


Figure 6.11: Memory load on each NIDS node in the network

6.2.2 Problem Formulation

The objective is to configure the NIPS modules to minimize the network footprint of unwanted traffic or equivalently to maximize how much we reduce the total network footprint by dropping such unwanted traffic. We want to generate *rule placements* specifying which rules are enabled on each NIPS node and *sampling manifests* specifying what fraction of the traffic the node should process for each enabled rule. Given the rule placements, the processing responsibilities are split to ensure that no node exceeds its memory/CPU capacity.

As a generalization, we consider the footprint of each packet in terms of network distance. Let $Dist_{ikj}$ be the downstream distance remaining on the path P_{ik} from R_j. $Dist$ can be measured in number of router hops, fiber distance, or routing weights. For example, if for C_i, the $P_{i1} = R_1, R_2, R_3$ in order, and we measure $Dist$ in router hops, $Dist_{i11} = 3$, $Dist_{i12} = 2$, and $Dist_{i13} = 1$. Alternatively, if we are only interested in the total volume of unwanted traffic dropped, we set all $Dist$ values to be 1.

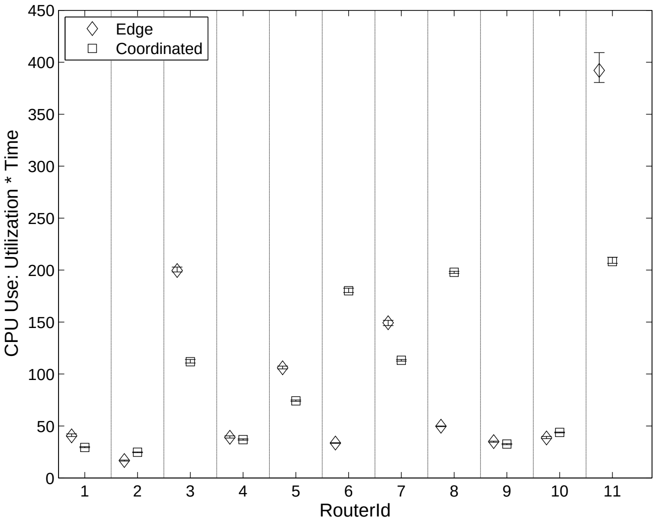


Figure 6.12: CPU load on each NIDS node in the network

Inputs:

- Each rule C_i is associated with three types of resources: (1) CPU processing load $CpuReq_i$ per packet, (2) memory load $MemReq_i$ if it needs to maintain any per-flow or cross-packet state, and (3) TCAM usage $CamReq_i$ per rule. Also, note that the $CamReq$ is ***per-rule*** rather than per-packet or per-flow.

- The capacity constraints $CpuCap_j$, $MemCap_j$, and $CamCap_j$ of each node R_j.

- The paths P_{ik}, their traffic volumes T_{ik}^{items} and T_{ik}^{pkts}, and the $Dist_{ikj}$ values for each node on the path.

- For each rule C_i, $Match_{ki}$ denotes the fraction of traffic along this path that *matches* the specific rule and will be affected by this rule. For example, if the rule C_i is designed to detect a specific malware signature, $Match_{ki}$ is the fraction of this malware traffic on the path P_{ik}. We assume that these can be estimated from measurements or alerts from the NIDS deployments.

Optimization Problem: Let e_{ij} be a $\{0, 1\}$ variable that specifies if rule C_i is *enabled* on node R_j. d_{ikj} denotes the fraction of traffic on path P_{ik} for which node R_j applies the filtering rule C_i.

Alternatively, we can consider the case where each node R_j applies all enabled rules $\{C_i | e_{ij} = 1\}$ to some fraction of the traffic. (In this case, d would depend only on j and k and not on i.) Our definition is more general and subsumes this specific instance.

Given this setup, we can formulate the NIPS deployment problem with these hardware constraints using the following Mixed Integer-Linear Program.

$$\text{Maximize} \sum_i \sum_k \sum_{j, R_j \in P_{ik}} T_{ik}^{items} \times Match_{ki} \times Dist_{kj} \times d_{ikj} \tag{6.7}$$

subject to

$$\forall j,\ \sum_i CamReq_i \times e_{ij} \leq CamCap_j \tag{6.8}$$

$$\forall j,\ \sum_k \sum_i T_{ik}^{items} \times MemReq_i \times d_{ikj} \leq MemCap_j \tag{6.9}$$

$$\forall j,\ \sum_k \sum_i T_{ik}^{pkts} \times CpuReq_i \times d_{ikj} \leq CpuCap_j \tag{6.10}$$

$$\forall k, \forall i,\ \sum_{j, R_j \in P_{ik}} d_{ikj} \leq 1 \tag{6.11}$$

$$\forall j, \forall i, \forall k, d_{ikj} \leq e_{ij} \tag{6.12}$$

$$\forall k, \forall i, \forall j,\ d_{ikj} \geq 0 \tag{6.13}$$

$$\forall i, \forall j,\ e_{ij} \in \{0, 1\} \tag{6.14}$$

The objective in Eq (6.7) models the total reduction in network footprint achieved by dropping unwanted traffic. For a specific i and k, the total number of unwanted flows of this type is $T_{ik}^{items} \times Match_{ki}$. Each node R_j that lies on P_{ik} contributes $Dist_{kj} \times d_{ikj}$ toward reducing the total footprint. Since we can effectively split the sampling responsibilities across the R_j on each P_{ik} by hashing (as in Figure 6.2), we can simply add up the contributions across the different nodes.

Eq (6.8) models the constraint on the number of rules that can be enabled in the constrained hardware on each node. Eq (6.9) and Eq (6.10) model the aggregate memory and processing load on each node. Eq (6.12) is a sanity check to ensure that a node cannot apply a rule C_i unless it has been enabled and Eq (6.11) ensures that the fraction of the total traffic sampled on each path-rule combination is never more than 1.

There are three implicit assumptions in the above formulation. First, for modeling the objective, we assume that attackers cannot explicitly craft patterns to avoid the sampling checks. That is, both legitimate and unwanted traffic patterns are distributed uniformly through the hash space. This is a reasonable assumption in practice: network administrator can use private keyed hash functions to prevent adversaries from evading the hash checks. Second, to rigorously model the load on a node, we should take into account the traffic dropped upstream on each path. In that case, Eq (6.9) and Eq (6.10) will be become non-linear constraints. Specifically, the LHS of these equations will have an extra product term $(1 - \sum_{j'<j} d_{ikj})$ to model the traffic that has already been dropped. We conservatively model the load in terms of the total volume entering the network (before any drops). Third, we assume that the rules themselves are non-redundant and the same packet/flow does not match multiple packets. Our high-level goal is to obtain effective guidelines for configuring the NIPS modules. To this end, these are reasonable assumptions that make the formulation practical.

The presence of the discrete e_{ij} variables (Eq (6.14)) makes such optimization problems NP-hard. Next, we show that our specific NIPS deployment problem is NP-hard via a reduction from the MAX-CUT problem.

6.2.3 Hardness of NIPS problem

The MAX-CUT problem is the following: given a graph $G = (V, E)$, we want to find $S \subset V$ such that the number of edges between S and $V \setminus S$ is *maximized*. It is well known that the MAX-CUT problem is NP-hard. We show NP-hardness of the NIPS deployment problem by reducing MAX-CUT to it.

Given an instance $G = (V, E)$ of the MAX-CUT problem, we construct an instance of the NIPS deployment problem as follows. Each vertex $v \in V$ corresponds to a node R_v in the NIPS deployment problem. Each edge $e = (u, v) \in E$ corresponds to a 2-node path consisting of the nodes R_u and R_v. Each node R_v has a TCAM capacity $CamCap = 1$. There are only two types of rules, C_0 and C_1, that can be enabled on the nodes. Each path P_k has $T_{ik} = 1/2$ for both $i = C_0$ and for C_1. Both rules have a match rate of 1 (i.e. $Match_{ki} = 1$). All nodes have no constraints on $CpuCap$ and $MemCap$.

CLAIM: *There is a max cut of size c if and only if the optimal solution to the NIPS deployment problem has value $m + \frac{c}{2}$, where m is the number of edges in G.*

The basic idea here is that enabling C_0 on a node R_v corresponds to assigning it to S and enabling C_1 equivalently corresponds to assigning it to $V \setminus S$. By doing this, we can drop all traffic corresponding to edges which cross the cut, i.e for all paths k such that one vertex of k is in S and the other $V \setminus S$. Each remaining path has the same rule enabled on both nodes and thus can get a maximum reduction of 0.5 in terms of volume of traffic dropped. (The sampling bounds on each path-rule combination in Eq (6.11) and (6.12) ensure this.)

First, we see that if there is a cut of size c that we get a total reduction of $m + \frac{c}{2}$. This is because of the following: For each vertex in S, let us enable C_0 and for each vertex in $V \setminus S$, enable C_1. The paths corresponding to the edges which cross the cut contribute a reduction of $\frac{1}{2} \times 2 + \frac{1}{2} \times 1 = \frac{3}{2}$ (because one of the rules will catch $1/2$ the volume of traffic at a downstream distance of 2, and the other rule will catch $1/2$ the volume traffic at a downstream distance of 1). For each other path (those corresponding to edges not crossing the cut) we can get a reduction of contribute a reduction of $\frac{1}{2} \times 2$. The total reduction then is $c \times \frac{3}{2} + (m - c) \times 1 = m + \frac{c}{2}$.

Conversely, we see that if the NIPS deployment problem has value $m + \frac{c}{2}$, then there is a cut of size c. Now, among the different paths, suppose c' of them have a reduction of $\frac{3}{2}$ and the remaining $m - c'$ have a reduction of 1. Since the total reduction is $m + \frac{c}{2}$, it must mean that $c' \geq c$. Again, if in the optimal solution, rule C_0 is enabled to node R_u, assign u to S, and to $V \setminus S$ otherwise. Thus, there is a cut of size at least c.

6.2.4 Approximation via Randomized Rounding

Given that it is NP-hard to solve the above optimization problem exactly, we use an approximation algorithm using randomized rounding [133]. Figure 6.13 describes the steps involved in our algorithm.

First, we solve a *relaxed* version of the problem by replacing the discrete e_{ij}s by continuous variables in the interval $[0, 1]$ and solving the resulting linear program. Then, starting from the solution to this linear program, we generate a solution to the original problem that (a) satisfies the constraints Eqs (6.8)–(6.11) and (b) is close to the optimal value.

As a first step, we would like to "round" the optimal fractional value e^*_{ij} in the LP solution to a binary value $\widehat{e_{ij}}$, by setting each $\widehat{e_{ij}}$ independently and randomly to 1 with the probability e^*_{ij}, and 0 otherwise. However, to decrease the chance of violating the constraint Eq (6.8), we set $\widehat{e_{ij}}$ to 1 only with probability $\frac{e^*_{ij}}{\alpha}$ (line 5 of Figure 6.13). While this ensures that most constraints in Eq (6.8) are satisfied, it could still violate a few of them. To rectify this, we reset some of these variables to zero (line 10) as necessary. To make sure that we do not violate the constraints Eqs (6.9)–(6.11), we ensure that the solution $\{\widehat{e_{ij}}\}_{ij}$, $\{\widehat{d_{ikj}}\}_{ikj}$ after the loop in lines 4–9 satisfies Eqs (6.9)–(6.11) to within some factor $\beta \log N$, where $N = \max\{\#nodes, \#rules\}$—see line 7. These constraints will be satisfied when we rescale the $\widehat{d_{ikj}}$s in lines 11–12. (We can do this because the $\widehat{d_{ikj}}$s are fractional quantities.)

Let Opt_{LP} denote the value of the objective function of the optimal LP solution (i.e., Eqs (6.7)–(6.13), and with Eq (6.14) replaced by the constraint $e_{ij} \in [0, 1]$. Let Opt_{NIPS} be the objective value of the optimal solution to the original "integer" formulation Eqs (6.7)–(6.14). We show in the next section that the process in Figure 6.13 outputs a feasible solution with objective function at least $\frac{\mathsf{Opt}_{LP}}{O(\log N)}$, where the constants in the big-oh depend

```
RandomizedRounding
     // Create LP relaxation
 1   Replace "$e_{ij} \in \{0,1\}$" in Eq (6.14) with "$0 \leq e_{ij} \leq 1$".
 2   Solve the LP relaxation to obtain $\{e^*_{ij}\}_{ij}$ and $\{d^*_{ikj}\}_{ikj}$.
 3   $\forall k, i, j$, $\epsilon_{ikj} \leftarrow d^*_{ikj}/e^*_{ij}$.
 4   repeat
 5       $\forall i, j$, Randomly set $\widehat{e_{ij}} \leftarrow 1$ with probability $\frac{e^*_{ij}}{\alpha}$,
         and $\widehat{e_{ij}} \leftarrow 0$ otherwise
 6       $\forall k, i, j$, $\widehat{d_{ikj}} \leftarrow \epsilon_{ikj}\widehat{e_{ij}}$.
 7       Check if any constraint in Eqs (6.9)–(6.11)
         is violated by a factor more than $\beta \log N$.
 8       If yes, call this trial a failure.
 9   until not failure
10   If for some $j$ the constraint Eq (6.8) is violated, arbitrarily set
         some $\widehat{e_{ij}}$ to $0$ until all constraints Eq (6.8) are satisfied.
11   $\forall k, i, j$, $\epsilon_{ikj} \leftarrow \frac{\epsilon_{ikj}}{\beta \log N}$.
12   $\forall k, i, j$, $\widehat{d_{ikj}} \leftarrow \epsilon_{ikj}\widehat{e_{ij}}$.
13   Output $\widehat{e_{ij}}$ and $\widehat{d_{ikj}}$.
```

Figure 6.13: Approximation algorithm for the NIPS deployment problem via randomized rounding.

on the scaling factors α and β. Since $\mathsf{Opt}_{LP} \geq \mathsf{Opt}_{NIPS}$, this guarantees that the value of our solution is at least $\frac{\mathsf{Opt}_{NIPS}}{O(\log N)}$. (Reasonable values are $\alpha = 4$ and $\beta = \sqrt{6}$.)

The algorithm in Figure 6.13 can be heuristically improved in two ways. First, the scaling of $\widehat{d_{ikj}}$ (line 11) is likely to be too conservative. A practical alternative is to solve the LP represented by Eqs (6.9)–(6.14) after setting the values for $\widehat{e_{ij}}$ obtained in line 5 to be constants, and use the values for $\{\widehat{d_{ikj}}\}_{ikj}$ returned by this solution. Second, we may be conservative in setting some $\widehat{e_{ij}}$ to zero (lines 10 and 5)—to fix this, we can greedily try to set $\widehat{e_{ij}}$s to 1 until no more can be set to 1 without violating Eq (6.8), and then solve the LP treating these $\widehat{e_{ij}}$ as constants. Since none of these steps affect feasibility and can only improve the value of the objective function, the above approximation guarantee holds on this extended heuristic as well. In practice, these heuristics boost the algorithm's performance significantly.

6.2.5 Sketch of Rounding Argument

We now present the analysis of the rounding algorithm from Section 6.2.4. Recall that $N = \max\{\#nodes, \#rules\}$. We begin by first (loosely) bounding Opt_{LP}, which will be useful later. To get an upper bound, imagine that we scale down the traffic volumes for every path to $\widetilde{T_{ik}}^{items} = \frac{T_{ik}^{items}}{\lambda}$, where $\lambda = \max_{i,k,j} T_{ik}^{items} \times Match_{ki} \times Dist_{kj} \times d_{ikj}^{*}$. Here, for any fixed i, k, j, d_{ikj}^{*} denotes the maximum value the variable can take so that all the constraints remain satisfied, even if no other rules are enabled. (Note that this scaling is only for the analysis and does not affect the algorithm as such.) Since we have scaled all T_{ik}^{items}s by λ, we also rescale the $MemCap_j$ bounds in Eq (6.9). Thus, any LP solution that was feasible with T_{ik}^{items} values is also feasible under the values $\widetilde{T_{ik}}^{items}$. Further, the quantity $\widetilde{T_{ik}}^{items} \times Match_{ki} \times Dist_{kj} \times d_{ikj} \leq 1$, for each k, i, j triplet. (Otherwise, this would violate the property that λ is the maximum value.) Therefore, the total objective function Opt_{LP} for the scaled problem is at most $1 \times N \times N^2 \times N = N^4$ (there could be at most N rules on N routers for each path, and there could be at most N^2 different paths).

At the other end, clearly we can enable just one rule i^* on a router j^* for a path k^*, and set $d_{i^*k^*j^*}$ to the maximum feasible value while still preserving all constraints, (this corresponds to $\arg\max_{i,k,j} T_{ik}^{items} \times Match_{ki} \times Dist_{kj} \times d_{ikj}$) and get a total objective of at least 1, while meeting all the constraints. Hence, $\mathsf{Opt}_{LP} \geq 1$. Therefore, we have the following bound on Opt_{LP}:

$$1 \leq \mathsf{Opt}_{LP} \leq N^4 \tag{6.15}$$

As described in the algorithm, the first step is to perform the randomized rounding in Steps 4–9. Notice that because we set $\widehat{e_{ij}}$ to 1 with probability $\frac{e_{ij}^*}{\alpha}$, we can apply linearity of expectation and observe that, for any constraint in Eq (6.9):

$$\mathbb{E}\left[T_k^{items} \times MemReq_i \times \widehat{d_{ikj}}\right] \leq \frac{MemCap_j}{\alpha} \tag{6.16}$$

We can use linearity of expectation, to also get that the expected value for each constraint in Eqs (6.10) and (6.11) are also at most $1/\alpha$ times their corresponding bounds. Now since each e_{ij} variable was rounded *independently* of the others, we can use a Chernoff bound (on sums of independent bounded random variables) to bound the probability that each fixed constraint in Eqs (6.9)–(6.11) is violated by a factor of $\beta \log N$ by $\frac{1}{N^{\alpha\beta^2/2}}$.

Next, we apply the union bound (on all the constraints) to get that the probability of *any* constraint from Eq (6.9)–(6.11) being violated (i.e., a failure event occurs) is at most $\frac{2N^3}{N^{\alpha\beta^2/2}}$ (there are at most N^3 constraints of the form Eq (6.11) and at most $2N$ other constraints from equations Eq (6.9) and Eq (6.10)). We can ensure that this is at most $1/N^8$, by setting $\alpha = 4$ and $\beta = \sqrt{6}$. Hence, we have with high probability, a 0-1 solution for the $\widehat{e_{ij}}$ variables which may violate some of the constraints Eq (6.8), but using which none of the constraints Eqs (6.9)–(6.11) are violated by more than a factor of $\beta \log N$. Before we worry about the violations for constraints Eq (6.8), let us bound the expected value of the objective function for the rounding procedure. From linearity of expectation, we have

$$\mathbb{E}\left[\sum_k \sum_{j, R_j \in P_k} \sum_i T_k^{items} \times Match_{ki} \times Dist_{kj} \times d_{ikj}\right] \geq \frac{\mathsf{Opt}_{LP}}{\alpha} \tag{6.17}$$

However, remember that we are interested in the expected objective function value *conditioned* on a non-failure. To calculate this, we use the two facts that (a) the probability of a failure is negligible (at most $1/N^8$), and when a failure occurs the value of the objective function is bounded by N^4 (see Eq (6.15)). If $\mathcal{E}$ denotes a failure event, we know that

$$\mathbb{E}\left[X\right] = \mathbb{E}\left[X|\mathcal{E}\right] \mathsf{Pr}\left[\mathcal{E}\right] + \mathbb{E}\left[X|\bar{\mathcal{E}}\right] \mathsf{Pr}\left[\bar{\mathcal{E}}\right],$$

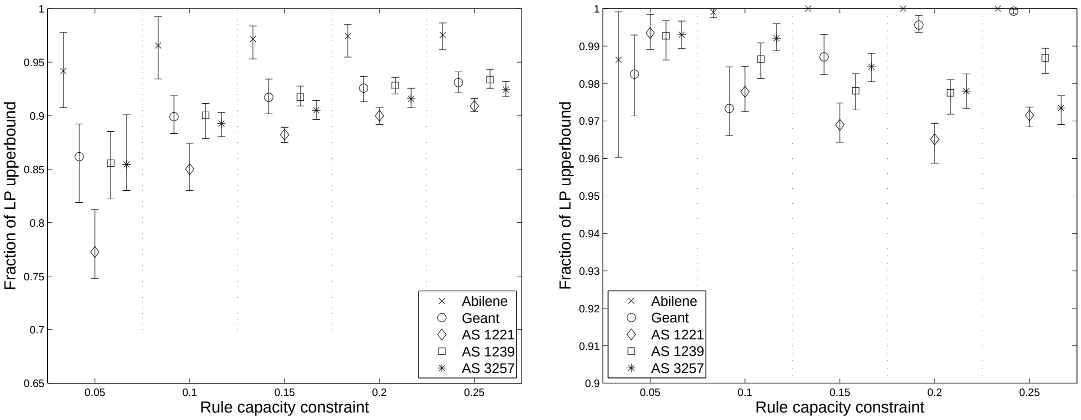
and hence

$$\begin{aligned}\mathbb{E}\left[X|\bar{\mathcal{E}}\right] &= (\mathbb{E}\left[X\right] - \mathbb{E}\left[X|\mathcal{E}\right] \mathsf{Pr}\left[\mathcal{E}\right])/\mathsf{Pr}\left[\bar{\mathcal{E}}\right] \\ &\geq (\mathsf{Opt}_{LP}/\alpha - 1/N^8 \cdot N^4).\end{aligned}$$

Now, because $\mathsf{Opt}_{LP} \geq 1$ and α will be set to a small constant, we have that $(\mathsf{Opt}_{LP}/\alpha - 1/N^8 \cdot N^4) \geq \frac{\mathsf{Opt}_{LP}}{\alpha+1}$. Therefore, the expected objective, conditioned on a non-failure is at least $\frac{\mathsf{Opt}_{LP}}{\alpha+1}$.

What remains is handling the possible violations in constraints Eq (6.8). To fix this, we reset some of the $\widehat{e_{ij}}$ values to 0 in Step 10. To this end, let us look at the probability of a fixed $\widehat{e_{i'j'}}$ variable getting dropped, conditioned on it being set to 1 originally. This happens when Eq (6.8) exceeds the bound $CamCap_{j'}$. But we know that over all the other rules, the expected load satisfies

$$\mathbb{E}\left[\sum_{i \neq i'} CamReq_i \times \widehat{e_{ij'}}\right] \leq (CamCap_j - e_{i'j'} \times CamReq_{i'})/\alpha$$



(a) Just rounding

(b) Rounding + Greedy

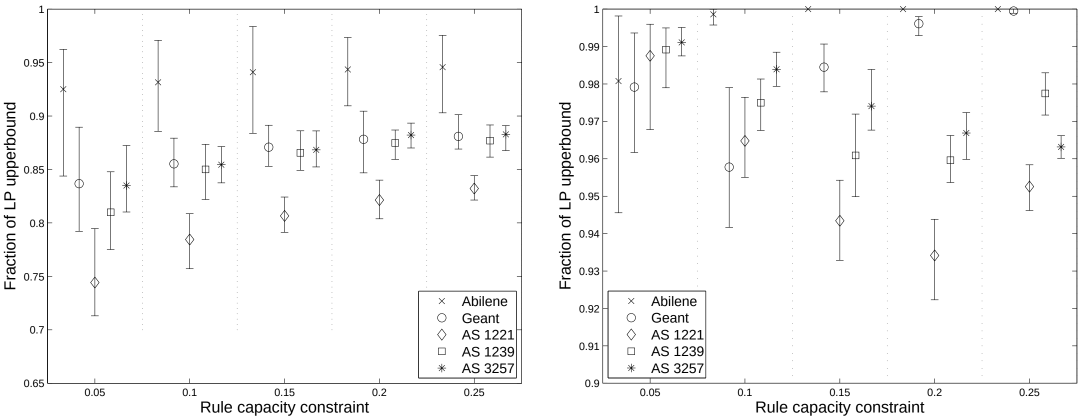
Figure 6.14: Performance of the approximation algorithms with a uniform rule match rate distribution

Therefore, we can use Markov's inequality and bound the probability that this sum of random variables exceeds ($CamCap_j - CamReq_{i'}$) to be at most $2/\alpha$.

Therefore, with probability at least $1 - \frac{2}{\alpha}$, any $\widehat{e_{ij}}$ which was set to 1 in Steps 4–9 is retained as 1. Therefore, the expected value of the objective function, after Step 10, is at least $\left(\frac{\alpha-2}{\alpha}\right)\left(\frac{1}{\alpha+1}\right)\mathsf{Opt}_{LP}$, and the only violated constraints are those in Eqs (6.9)–(6.11) — and even these are violated by only a factor of $\beta \log N$. But this is rectified in Step 11, when we scale each of the ϵ values by this factor. Therefore, all the constraints are satisfied, and the objective function value drops by a factor of $\beta \log N$. Therefore, the final expected objective is at least $\frac{(\alpha-2)}{\alpha(\alpha+1)\beta \log N}\mathsf{Opt}_{LP}$ and all constraints are satisfied with very high probability. Specifically, if we set $\alpha = 4$, and $\beta = \sqrt{6}$, we get an $1/(25 \log N)$-approximation.

6.2.6 Evaluation

For this evaluation, we use network topologies from educational backbones (Internet2 and Geant) and tier-1 ISP backbone topologies inferred by Rocketfuel [157]. We construct ingress-egress paths for each pair of nodes using shortest-path routing [117]. We use a gravity model traffic matrix based on city populations [150]. To model the total volume, we start with a baseline of 8 million flows and 40 million packets (per 5 minute interval) for Internet2 based on publicly available estimates. For the other networks (Geant, AS



(a) Just rounding

(b) Rounding + Greedy

Figure 6.15: Performance of the approximation algorithms with an exponential rule match rate distribution

1221, AS 1239, AS 3257) we scale the total volume linearly as a function of network size from this baseline estimate. Each node R_j in the network has a total $MemCap_j$ of 400000 flows and a $CpuCap_j$ of 2 million packets that it can process in this 5-minute interval. We use $Dist$ values measured in router hops.

We assume that there are a total of 100 NIPS rules, each having a unit requirement of TCAM, packet processing, and flow memory units; i.e., $\forall i, CamReq_i = CpuReq_i = MemReq_i = 1$. We present results for two scenarios: (1) $Match_{ki}$ values are distributed uniformly in the range $[0, 0.01]$ and (2) $Match_{ki}$ values follow an exponential distribution with mean 0.01. For the following results, we vary the $CamCap_j$ of each node as a fraction of the total number of NIPS rules. For each setting, we generate 30 different $Match_{ki}$ values; run 10 iterations of the rounding algorithm and take the best solution across these 10 runs.

Figures 6.14 and 6.15 present the mean, minimum, and maximum value obtained by the rounding algorithm across the 30 $Match_{ki}$ scenarios as a function of Opt_{LP}.[4] In each case, we show the performance of the basic rounding algorithm and the rounding algorithm augmented with the heuristic improvements described above.

First, we notice that the performance of the basic rounding algorithm is much better than the approximation ratio of $\frac{1}{O(\log N)}$ as we get more than 70% of Opt_{LP}. Second, we

[4] Since it is hard to find the true optimum, we use the LP upper bound as a proxy. Note that this is a conservative estimate of the true performance of our approximation algorithms.

notice that the greedy heuristic step can significantly boost the performance to consistently get more than 92% of Opt_{LP}. We note that these results are consistent across the different topologies and $CamCap_j$ constraints; we have verified these for other distributions of $Match_{ki}$ values as well.

6.2.7 Online Adaptation

The above formulation considers a static scenario where the match rates are known and fixed. However, an adversary can control the sources and nature of the unwanted traffic. For example, an attacker who controls a large botnet can modify the attack profile–the sources and destinations of the malicious traffic and the attack mix– to evade NIPS-based defenses. Our goal is to adapt the NIPS deployment to be robust to such adversaries.

To model the online or adaptive version of the NIPS deployment problem, we leverage the framework described by Kalai and Vempala [85] for modeling *online linear optimization problems*. The general problem can be described as follows. We have to make a series of decisions $O_1, O_2, \ldots$, from some possible space of decisions $\mathcal{O} \subset \Re^n$. At each step t, there is a cost $O_t.S_t$ associated with making the decision O_t, where $S_t \in \mathcal{S} \subset \Re^n$ represents the state of the world at time t, and '.' denotes the dot product between the two vectors O_t and S_t. However, the state S_t is revealed only after the decision for the t^{th} step O_t has been made and we do not have access to the current state S_t before making the decision O_t.

$$\text{Maximize} \sum_i \sum_k \sum_{j, R_j \in P_k} T_{ik}^{items} \times Match_{ki} \times Dist_{kj} \times d_{ikj}$$

$$\text{subject to}$$

$$\forall j,\ \sum_k \sum_i T_{ik}^{items} \times MemReq_i \times d_{ikj} \leq MemCap_j \tag{6.18}$$

$$\forall j,\ \sum_k \sum_i T_{ik}^{pkts} \times CpuReq_i \times d_{ikj} \leq CpuCap_j \tag{6.19}$$

$$\forall k, \forall i,\ \sum_{j, R_j \in P_k} d_{ikj} \leq 1 \tag{6.20}$$

$$\forall k, \forall i, \forall j,\ d_{ikj} \geq 0 \tag{6.21}$$

Next, we describe how to leverage this framework for adaptive NIPS deployment. As a starting point, we consider a simplified version of the NIPS deployment problem where we do not have the TCAM constraints. The above linear program models the optimization problem for the static case.

To permit adaptation, we divide time into *epochs*. In each epoch t, O_t is a vector of the sampling variables d_{ikj}s. The state of the world S_t at time t captures the traffic profile in terms of the match rates for the different rules. Specifically, each S_t is a vector of values, each of the form $T_{ik}^{items} \times Match_{ki} \times Dist_{kj}$ for some $i,\ k,\ j$. The size n of the decision and state vectors is thus $n = M \times N \times L$, where M is the number of paths in the network (over which k ranges), N is the number of NIPS nodes (over which j ranges), and L is the total number of NIPS rules/classes (over which i ranges). Each "cost" term directly corresponds to a term in our objective; i.e., $d_{ikj} \times (T_{ik}^{items} \times Match_{ki} \times Dist_{kj})$.[5] An adversary can change the different $Match_{ki}$ values over time to vary the traffic mix. Our goal is to adapt the NIPS deployment without knowing the exact $Match_{ki}$ values in each epoch.

The goal is to have a total cost over τ epochs, $\sum_{t=1}^{\tau} O_t.S_t$, that is close to $mincost_\tau = \min_{O \in \mathcal{O}} \sum_{t=1}^{\tau} O.S_t$. That is, we want our cost to be comparable to the cost of the best possible single solution in hindsight.[6] The *regret* is defined as $\sum_{t=1}^{\tau} O_t.S_t - mincost_\tau$; the difference between the costs incurred by the online decision procedure and this single best decision chosen in hindsight.

Kalai and Vempala [85] show how to convert a black-box optimization algorithm for computing the best static solution into an online algorithm that minimizes the worst-case regret. Given a procedure Λ that takes as input the state S and returns $\arg\min_{O \in \mathcal{O}} O.S$, they suggest a *follow the perturbed leader (FPL)* strategy, where at each time step t and for some $\epsilon > 0$:

1. Choose p_t uniformly at random in $[0, \frac{1}{\epsilon}]^n$.

2. Use $O_t = \Lambda(\sum_{j=1}^{t-1} S_j + p_t)$.

Intuitively, to make the decision O_t at time t, the algorithm uses as input to Λ a *perturbed* function of the historical sum of the state vectors observed up to $t-1$. The perturbation term guards against adversaries who know our strategy. If we chose O_t simply using the sum of S up to $t-1$, an adversary can generate values of S_t such that the regret will be very high.

[5] Even though we describe the NIPS problem as a maximization, we can think of the "cost" as the volume of unwanted traffic that we let through.

[6] In general, it is not possible to provide guarantees with respect to the best possible dynamic solution.

It can be shown that the FPL strategy has provably low regret. In particular, if we define constants D, R, and A such that,

- $\forall O, O' \in \mathcal{O}, D \geq |O - O'|_1$ (i.e., maximum L1-norm difference between any two decision vectors)
- $\forall O \in \mathcal{O}, S \in \mathcal{S}, R \geq |O.S|$ (i.e., maximum possible value of the cost function)
- $\forall S \in \mathcal{S}, A \geq |S|_1$ (i.e., maximum possible L1-norm of the state vector),

then, FPL with parameter $\epsilon = \sqrt{\frac{D}{RA\tau}}$ gives,

Theorem 6.2.1 $\frac{E[cost(FPL(\epsilon)) - mincost_\tau]}{\tau} \leq \sqrt{\frac{DRA}{\tau}}$ *[85].*

That is, the average regret goes to zero as τ increases.

The optimization procedure Λ in our case involves solving the linear program. To apply the theorem, we set the constants D, R, and A as follows: $D = M \times N \times L$ and $R = A = \sum_{ik} T_{ik}^{items} \times maxdrop$, where $maxdrop$ is a conservative upper bound on the maximum fraction of traffic we expect to be dropped. Then, in each epoch t, we set $Match_{ki} = \frac{\sum_{j=1}^{t-1} Match_{ki}^{Obs}(j)}{t-1} + \frac{p_t}{t \times T_{ki}^{items}}$, where p_t is computed as described in the FPL procedure. (The normalization factors in the p_t term arise because the state variables S correspond to the product of the match rate and traffic.)

Preliminary Evaluation: To evaluate this online adaptation procedure, we use the same setup from Section 6.2.4 (without the rule capacity constraints). We consider a dynamic setting in which the $Match_{ki}$ are chosen at random from a uniform match rate distribution, but are revealed to us only at the end of each epoch.

The metric we are interested is the average normalized regret as function of time: $\frac{\sum_{t=1}^{\tau} Obj_t^{staticopt} - Obj_t^{FPL}}{\sum_{t=1}^{\tau} Obj_t^{staticopt}}$, where Obj denotes the value of the objective function achieved by the different decision procedures. That is, we normalize the total regret by the total objective value achieved by the best possible static solution. Figure 6.16 shows this normalized regret metric over time for 5 independent runs for the Internet2 setup. Across the different runs, the regret is at most 15% of the best single solution we could have chosen in hindsight. (In some epochs, the regret is negative, meaning that the online algorithm is actually better than the best static strategy.) This preliminary result demonstrates the promise of leveraging such online adaptation strategies for robust NIPS deployment. As future work,

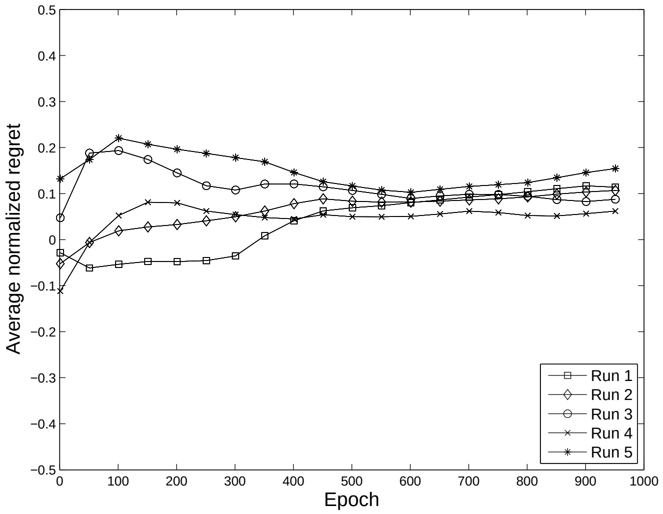


Figure 6.16: Result showing the normalized regret over time for different runs of the online adaptation algorithm. We normalize the regret by the objective value of the best static solution.

we will explore how well such strategies perform in the presence of strategic adversaries and extend this framework to the general formulation from Section 6.2.2.[7]

6.3 Related Work

Network management: Several recent efforts have demonstrated the benefits of a coordinated approach for network management [66, 184, 37, 43, 73]. In the context of monitoring and sampling, hash-based packet selection to coordinate monitoring responsibilities has been used in the context of Trajectory Sampling [54] and cSamp [147]. We build on this prior work. However, NIDS/NIPS deployment present unique constraints in modeling the problems that we address in this chapter.

Monitor placement: Several research efforts have studied the problem of placing network monitors to cover all routing paths using as few monitors as possible [159, 45]. These show that the problems are NP-hard and propose greedy algorithms. Kodialam et al [113]

[7] There are known extensions for the case where Λ is an approximation algorithm [85, 109].

consider the problem of routing traffic such that each end-to-end path passes through at least one content filtering node. Our formulations differ in two key respects. First, we model the problem as one of enabling different modules with different sampling rates subject to resource constraints. Second, we operate within the current routing framework and do not modify routing policies.

Scaling NIDS/NIPS: There are several efforts for building scalable NIDS/NIPS systems using parallelization (e.g., [42, 166, 70, 156, 105, 104]), hardware-assisted acceleration (e.g., [81]), more efficient algorithms (e.g., [103]), models for understanding their resource consumption (e.g., [76, 77]), and optimizing rule patterns (e.g, [33, 27, 59, 179, 178]). Our work effectively complements these because we exploit *spatial* opportunities for distributing NIDS/NIPS functions across a network.

Distributed intrusion detection: Distributed intrusion and anomaly detection systems have been actively studied in the research literature and commercial deployments (e.g., [87, 64, 26, 152, 162, 165, 38]). As applications and attacks become distributed, we need to aggregate information across a network for effective analysis [100, 108, 110]. For example, understanding peer-to-peer traffic [49], hit-list worms [115], and understanding DDoS attacks [145] require a network-wide view from multiple vantage points. Our current formulation is restricted to the case where each NIDS/NIPS operation can be performed at one network location. As future work, we plan to extend our models to include such network-wide analysis modules (e.g., incorporating communication costs).

6.4 Discussion

Provisioning and Upgrades: So far, we considered the problem of optimally configuring a NIDS/NIPS infrastructure. We can extend the formulations from Sections 6.1.2 and 6.2.2 to describe what-if provisioning scenarios: where should an administrator add more resources (e.g., [166]) or augment existing deployments with more powerful hardware (e.g., [81]).

Handling routing changes: A natural concern with splitting the analysis functions across a network is with routing changes. Network paths are largely stable on the timescales we are interested in for per-session analysis [182]. However, when route changes do occur and we recompute the optimal solutions, there is a concern that this may affect the correctness of stateful analysis. Specifically, the new optimal solution may be such that the node maintaining some specific connection state is no longer responsible for monitoring that connection.

The key challenge is to ensure correctness in the presence of such routing dynamics. In this regard, we can tradeoff some loss in performance to ensure correctness. The main idea is that nodes temporarily retain the old responsibilities until any existing connections associated with these assignments expire. That is, each node picks up the new assignment work immediately but takes on no new connections that belong to the old assignments. This may result in some duplication, but provides correct operation and will not result in false negatives. However, it may be the case that new packets for connections in the old assignment no longer traverse this node as a result of the routing change. In this case, we may have to transfer the current NIDS state associated with these connections to the new node responsible for analyzing these [155]. Also, adding in redundant functionality as outlined in Section 6.1.5 can further reduce the impact of routing changes.

6.5 Chapter Summary

In this chapter, we provided systematic formulations for effectively managing NIDS and NIPS deployments. In doing so, we used a network-wide coordinated approach, where different NIDS/NIPS capabilities can be optimally distributed across different network locations depending on the operating constraints–traffic profiles, routing patterns, and the resources available at each location.

Our models and algorithms will help administrators to optimally leverage their existing infrastructure toward their security objectives. Moreover, by focusing on the network-wide aspect, it effectively complements other efforts to scale single-vantage-point NIDS and NIPS. Furthermore, it can offer better incremental scalability to upgrade installations as new systems become available.

Chapter 7

Conclusions and Future Work

The work in this dissertation was motivated by the gap between the goals of network management applications and the tools available to administrators. Much of this disconnect stems from the device-centric approach taken by current solutions. This view has led to the development of narrow, incremental, and inefficient workarounds to address the limitations of existing solutions.

One of the key observations in this dissertation was that several network management tasks can be cast as system-wide resource management problems. Having cast the problems as such, we provided systematic solutions based on three high-level principles: choosing and placing the appropriate device-level primitives, coordinating different network elements to leverage the available resources effectively, and using network-wide optimization models to configure network elements to meet specific policy objectives.

Next, we briefly summarize the main contributions and implications of the work presented in this dissertation before highlighting some potential avenues for future work.

7.1 Contributions and Implications

Traffic Monitoring: Flow-level traffic monitoring is a critical aspect of network management that enables and guides several other facets of management such as anomaly detection, traffic engineering, and network security. Several measurement and analytical studies have demonstrated the limitations of current monitoring solutions based on packet sampling for such applications. As a result, several application-specific solutions have emerged to address this disconnect between the requirements of flow monitoring applications and the capabilities available today. These narrow solutions increase router complexity without providing the requisite generality.

The architecture we discussed in Chapters 2–4 has both immediate and long-term implications for router vendors, network operators, and researchers. First, it reduces router complexity without compromising a vendor's ability to meet customer demands. Second, it helps network operators insulate their deployment efforts from the changing needs of management applications. Third, it provides the impetus to motivate further research on developing robust generic primitives.

Redundancy Elimination:

The success of redundancy elimination in enterprise networks has sparked growing interest in a network-wide RE service. A network-wide RE service benefits ISPs by reducing link loads and increasing effective network capacity to better accommodate bandwidth-intensive applications. Further, it generalizes the benefits of RE to all end-to-end traffic. The design and implementation of SmartRE, presented in Chapter 5, takes this vision closer to reality by achieving close-to-optimal benefits under practical constraints.

NIDS/NIPS Deployment:

Network intrusion detection (NIDS) and prevention systems (NIPS) serve a critical role in detecting and dropping malicious traffic. There are several efforts for scaling NIDS and NIPS using parallelization (e.g., [42, 166, 70, 156, 104]), hardware-assisted acceleration (e.g., [81]), more efficient algorithms (e.g., [103]), models for understanding their resource consumption (e.g., [76, 77]), and optimizing rule patterns (e.g, [33, 27, 59, 179, 178]). These existing approaches primarily target single-vantage-point solutions. However, such efforts for scaling NIDS/NIPS systems are insufficient in the context of large enterprise networks, ISPs, and emerging contexts such as data centers.

The work presented in Chapter 6 targets the network-wide aspect and effectively complements advances in these areas. Thus, it enables administrators to optimally protect their infrastructure against attacks with existing deployments. It also offers incremental scalability for upgrading installations as newer generations of NIDS and NIPS become available.

7.2 Potential Limitations

Before describing some avenues for future work, we reflect on some potential limitations.

Scalability:

The first question in any centralized optimization is the issue of scale–Can the optimization module handle large network topologies on the order of hundreds of nodes? In some cases (e.g., cSamp-T, NIPS deployment) the optimization problems are provably NP-hard, making this question more relevant. Fortunately, we have shown that we can address this challenge by leveraging existing algorithmic techniques such as using Max-Flow based reformulation, binary search, lazy submodular evaluation, parallel execution, etc. We can use two additional optimizations: (1) precomputing solutions for expected configurations (e.g., to adapt to predictable traffic dynamics), or (2) seeding the optimization solvers with previous starting solutions to avoid running the algorithms from scratch. Another option to address the scalability concerns is to extend the models to loosely federated network settings–This would allow distributed agents for individual network components to run local algorithms (of smaller size) toward a global objective.

Availability of inputs to optimization:

The optimization formulations presented in the preceding chapters require the following inputs: (1) the traffic matrix (in terms of number of bytes, packets, and flows), (2) the routing paths for each pair of ingress-egress routers, (3) the capabilities and resources available at each network node (e.g., memory, processing, TCAM), and (4) in some cases more fine-grained properties of the traffic (e.g., attack match rates, redundancy profiles etc.).

Fortunately, there is a rich literature on traffic matrix estimation [184, 66, 185, 154] and other management tools for tracking routing state (e.g., [148]) that are deployed by operational networks today. The technology capabilities of network elements can be obtained from vendor and configuration databases or by benchmarking (e.g., [77]). Furthermore, the systems we describe have a natural *positive feedback* in that the data generated from these deployments will provide more fine-grained information that will improve the of these inputs. For example, cSamp will yield more fine-grained flow-level measurements; SmartRE and the NIDS deployments can provide a better view into the traffic mix.

Sensitivity to input parameters:

Even if the above input parameters are available, there is the issue of sensitivity–Will a management framework based on optimization models be useful if the input parameters are not entirely accurate, which is often the case in practice? For example, there are known issues with errors in traffic matrix estimation, routing paths need to be recomputed as links/nodes go down, and the redundancy/attack profiles could change over time.

While good input data for traffic profiles (i.e., the traffic matrix and attack/redundancy profiles) are necessary for the optimization modules, having perfect input data is less crucial. In other words, it is possible to obtain the benefits of a system-wide approach even with *approximate* inputs. In many cases, the major contribution to the performance benefits arise from patterns than tend to be stable and predictable. For example, large traffic matrix elements tend to be more stable over time. Similarly, the most common sources of redundant traffic also tend to be stable [31]. Also, we can develop specific heuristics to workaround errors in input estimates. For example, we describe one such scaling suggestion in Section 2.2.4, where we can handle bounded errors in the traffic matrix estimates.

Errors in routing data can lead to reduced coverage and thus a small loss of performance in the cSamp case. However, these are a more serious problem in the case of NIDS and SmartRE deployments, because it can affect *correctness*. In this case, we have to develop domain-specific strategies that will provide correctness even in the presence of routing dynamics or errors in routing data–e.g., failover configurations or explicitly providing redundant coverage.

Additionally, periodic recomputation can help adapt to changing conditions. As we described in the specific chapters, the time taken to compute the optimal solution is on the order of tens to hundreds of seconds, even for very large network topologies. Thus, we can recompute the optimal solution as network conditions change.

Does optimization make management more "black-box"?

Network operators often want direct control over the configurations of network elements and might be reluctant to use third-party software tools for management. In this respect, there is a concern that optimization might seem like a "black-box" which generates configurations that may not be intuitive, and thus our techniques may not be adopted.

We note that there is growing evidence that network operators in enterprise networks and ISPs are beginning to use centralized processes for network configuration [73, 43, 37, 69, 160, 60, 126, 125]. The motivation behind such proposals is to make network configuration less of a "black art" and provide more direct mechanisms for operators to specify and achieve their policy goals. In fact, these are arguably less black-box than the current alternative where operators purchase third-party "middleboxes" to provide some functionality (e.g., [34, 19, 7, 12, 162]). The systems presented in this dissertation are designed in the same spirit to enable to operators to specify the high-level intent to the configuration module. Nevertheless, developing suitable user-interface and network visualization tools will ease the adoption of the systems proposed in this dissertation [67, 65, 22, 163, 95].

Does coordination make it easier for adversaries to evade detection?

With random sampling, it is difficult for an adversary to determine which packets or flows will get monitored. With a more coordinated approach, as in cSamp, where the monitoring assignments are determined by an optimization algorithm as in Chapters 2 and 6, there is a natural concern that adversaries can *guess* the network's monitoring configuration. Thus, they can use this to either generate or redirect their malicious traffic to evade the monitoring infrastructure.

While the increased coverage and scalability provided by cSamp and the NIDS deployments reduce the likelihood of adversaries evading detection, network operators can take additional measures to further alleviate such concerns. Specifically, the actual hash range assignments can be randomized by making the mapping procedure in Figure 2.1 less deterministic. Further, they can seed the hash function with a private key/seed value that will not be exposed to adversaries.

7.3 Future Work

Going beyond monitoring 5-tuples:

Some settings require more fine-grained monitoring capabilities that look beyond flow-level statistics. These include analyzing end-to-end performance metrics (e.g., loss, throughput, latency) and on-demand analysis (e.g., analyze hosts that show specific patterns). Our minimalist primitives, as described in this dissertation, do not provide these capabilities. However, we believe that the broad principles underlying a minimalist approach will still apply and assume more importance with more complex monitoring requirements. One possible solution is to include a few flexible primitives that support such capabilities [46, 180] within the minimalist framework.

A unified model for flow monitoring applications:

The promise of a minimalist monitoring approach leads to a broader question:
Can we design a unified framework to understand how a given monitoring infrastructure performs for potential applications?

That is, given a specific application portfolio consisting of a variety of traffic metrics that we want to estimate and an available set of monitoring primitives (e.g., packet sampling, flow sampling), we want formal models that will help us reason what the estimation errors for the various applications will be. There are three challenges to address this question: (1) developing suitable abstractions for modeling application requirements (e.g., how sensitive is an estimation task to missing data), (2) deriving optimal estimators with available primitives (e.g., what is a good way to combine the reports from different sampling solutions for each estimation task?), and (3) reasoning about what additional primitives would best serve specific applications (e.g., what-if scenarios to analyze how adding some new capability would change the performance).

From a practical viewpoint, such a framework will help guide provisioning decisions (e.g., retain current infrastructure? upgrade to new hardware?). From a theoretical perspective, this will generalize existing work that analyzes the accuracy of algorithms fine-tuned for particular applications.

Robust provisioning and deployment:

Some of the chapters described models for provisioning network elements or formulations for incremental upgrades and deployment. A natural concern is the robustness of these upgrades to routing and traffic dynamics. Consider the simple fact that traffic matrices exhibit distinct diurnal trends; in this case choosing an upgrade policy based on a specific snapshot in time might be suboptimal. One direction of future work is to incorporate techniques from oblivious routing [175, 32] to obtain good guidelines for provisioning that are robust to network dynamics.

Integrating routing and other aspects of management:

In this dissertation, we considered the monitoring, redundancy elimination, and intrusion detection/prevention problems in the context of a fixed routing infrastructure. We can extend these problems to consider more flexible alternatives that integrate routing and different management applications (e.g., [30, 137]). These become particularly appealing in emerging contexts with programmable routers for data centers and enterprise networks [83, 125].

Coordination and optimization in loosely federated settings:

The models presented in this dissertation assume that the entire network is under a single administrative domain. Even within a single logical domain, there are policy and technology considerations that often lead to hierarchical or loosely federated management structures. For example, ISPs typically use “areas” for simplifying routing management. A natural extension to our formulations is to consider efficient coordination and optimization models for such settings.

There are two key issues here: the information available to each device to make decisions and granularity at which the objective functions are specified. For example, in the case of cSamp, routers might not have end-to-end path identifiers that identify ingress/egress routers, but only have coarser path identifiers that to identify the ingress/egress PoPs or areas. One potential approach is to consider a multi-level optimization process that first generates the sampling assignments at the coarsest level, and then subsequently solves optimization problems for the lower layers. For example, in the cSamp case, the first step might be to generate PoP-level assignments, and then each PoP runs a cSamp-like optimization to assign responsibilities to routers within the PoP. However, this might lead to situations where there is no feasible solution at the more fine-grained level. Going back to the cSamp scenario, this might mean that the optimum minimum fractional objective at the PoP and router-level granularities might be different. In this case, we need mechanisms to refine the optimization model by introducing new constraints, adding more information at the coarse-level formulation, or systematically trading off the performance.

Bibliography

[1] http://www.snort.org.

[2] Akamai Technologies. http://www.akamai.com.

[3] BlueCoat: WAN Optimization. http://www.bluecoat.com/.

[4] Bob hash. http://burtleburtle.net/bob/hash/doobs.html.

[5] Cisco Content Aware Networks – Some Areas of Interest. http://www.cisco.com/web/about/ac50/ac207/crc_new/ciscoarea/content.ht%ml.

[6] Cisco Wide Area Application Acceleration Services. http://www.cisco.com/en/US/products/ps5680/Products_Sub_Category_Home.%html.

[7] Citrix, application delivery infrastructure. http://www.citrix.com/.

[8] Computerworld - WAN optimization continues growth. www.computerworld.com.au/index.php/id;1174462047;fp;16;fpid;0/.

[9] Flexible Netflow. http://www.cisco.com/en/US/products/ps6965/products_ios_protocol_optio%n_home.html,.

[10] IETF IPFIX Working Group. http://datatracker.ietf.org/wg/ipfix/.

[11] Juniper cflowd. http://www.juniper.net/techpubs/software/junos/junos91/swconfig-policy/%cflowd.html.

[12] Narus Intercept Solution. http://www.narus.com/index.php/solutions/intercept.

[13] Netequalizer Bandwidth Shaper. `http://www.netequalizer.com/`.

[14] NetFlow Input Filters. `http://www.cisco.com/en/US/docs/ios/12_3t/12_3t4/feature/guide/gtnfinpf%.html`.

[15] OpenMP. `www.openmp.org`.

[16] Packet Sampling, IETF Working Group Charter. `http://www.ietf.org/html.charters/psamp/charter/`.

[17] PAPI: Performance Application Programming Interface. `http://icl.cs.utk.edu/papi/`.

[18] PeerApp: P2P and Media Caching. `http://www.peerapp.com`.

[19] Riverbed Networks: WAN Optimization. `http://www.riverbed.com/solutions/optimize/`.

[20] WAN optimization revenues grow 16% - IT Facts. `www.itfacts.biz/wan-optimization-market-to-grow-16/1205/`.

[21] Yaf flow collector. `http://tools.netsa.cert.org/yaf`.

[22] Y. Moses, Z. Polunsky, A. Tal, and L. Ulitsky. Algorithm Visualization for Distributed Environments. In *Proc. Infovis*, 1998.

[23] A. Anand and C. Muthukrishnan and A. Akella and R. Ramachandran. Redundancy in Network Traffic: Findings and Implications. In *Proc. of SIGMETRICS*, 2009.

[24] A. Chakrabarti, K. Do Ba, and S. Muthukrishnan. Estimating entropy and entropy norm on data streams. In *Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2006.

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

[26] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proc. RAID*, 2001.

[27] S. Acharya, M. Abliz, B. Mills, T. F. Znati, J. Wang, Z. Ge, and A. Greenberg. OPTWALL: A Traffic-Aware Hierarchical Firewall Optimization. In *Proc. NDSS*, 2007.

[28] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. STOC*, 1996.

[29] Personal Communication with Aman Shaikh, AT&T Research.

[30] 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.

[31] A. Anand, V. Sekar, and A. Akella. SmartRE: An Architecture for Coordinated Network-Wide Redundancy Elimination. In *Proc. SIGCOMM*, 2009.

[32] D. Applegate and E. Cohen. Making Intra-Domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs. In *Proc. of ACM SIGCOMM*, 2003.

[33] D. L. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang. Compressing Rectilinear Pictures and Minimizing Access Control Lists. In *Proc. SODA*, 2007.

[34] Arbor networks. `http://www.arbor.com`.

[35] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and Delay Accountability for the Internet. In *Proc. of ICNP*, 2007.

[36] AT&T Enterprise Threat Management. `http://www.business.att.com/enterprise/Family/business-continuity-enter%prise/threat-management-enterprise/`.

[37] H. Ballani and P. Francis. CONMan: A Step Towards Network Manageability. In *Proc. of ACM SIGCOMM*, 2007.

[38] P. Barford, S. Jha, and V. Yegneswaran. Fusion and Filtering in Distributed Intrusion Detection Systems. In *Proc. Allerton Conference on Communication, Control and Computing*, 2004.

[39] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: realistic and controlled network experimentation. In *Proc. SIGCOMM*, 2006.

[40] D. Brauckhoff, B. Tellenbach, A. Wagner, A. Lakhina, and M. May. Impact of Traffic Sampling on Anomaly Detection Metrics. In *Proc. of IMC*, 2006.

[41] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proc. ACM SIGMOD*, 2003.

[42] C. Kruegel, F. Valeur, G. Vigna, and R. A. Kemmerer. Stateful Intrusion Detection for High-Speed Networks. In *Proc. IEEE Symposium on Security and Privacy*, 2002.

[43] 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.

[44] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *12th Integer Programming and Combinatorial Optimization Conference*, pages 182–196, 2007.

[45] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the Monitor Placement problem: Optimal Network-Wide Sampling. In *Proc. of CoNeXT*, 2006.

[46] L. D. Carli, Y. Pan, A. Kumar, C. Estan, and K. Sankaralingam. PLUG: Flexible Lookup Modules for Rapid Deployment of New Protocols in High-speed Routers. In *Proc. SIGCOMM*, 2010.

[47] C. Chadet, E. Fleury, I. Lassous, Hervé, and M.-E. Voge. Optimal Positioning of Active and Passive Monitoring Devices . In *Proc. of CoNeXT*, 2005.

[48] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954.

[49] M. P. Collins and M. K. Reiter. Finding Peer-to-Peer File-sharing using Coarse Network Behaviors. In *Proc. of ESORICS*, 2006.

[50] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55, 2005.

[51] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991.

[52] 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.

[53] H. Dreger, A. Feldmann, B. Krishnamurthy, J. Wallerich, and W. Willinger. A Methodology for Studying Persistency Aspects of Internet Flows. *ACM SIGCOMM CCR*, 35(2), Apr. 2005.

[54] N. Duffield and M. Grossglauser. Trajectory Sampling for Direct Traffic Observation. In *Proc. of ACM SIGCOMM*, 2001.

[55] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *Proc. of IMW*, 2001.

[56] N. Duffield, C. Lund, and M. Thorup. Estimating Flow Distributions from Sampled Flow Statistics. In *Proc. of ACM SIGCOMM*, 2003.

[57] N. Duffield, C. Lund, and M. Thorup. Learn more, sample less: Control of volume and variance in network measurement. *IEEE Transactions in Information Theory*, 51(5):1756–1775, 2005.

[58] N. Duffield, C. Lund, and M. Thorup. Optimal Combination of Sampled Network Measurements. In *Proc. of IMC*, 2005.

[59] E. W. Fulp. Optimization of network firewalls policies using directed acyclic graphs. In *Proc. Internet Management Conference*, 2005.

[60] W. Enck, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, S. Rao, and W. Aiello. Configuration Management at Massive Scale: System Design and Experience. In *Proceedings of Usenix Annual Technical Conference*, 2007.

[61] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *Proc. of ACM SIGCOMM*, 2004.

[62] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. In *Proc. of ACM SIGCOMM*, 2002.

[63] J. R. Evans and K. Martin. A Note on Feasible Flows in Lower-Bounded Multicommodity Networks. *Journal of Operations Research Society*, 29(9):923–927, 1978.

[64] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proc. IEEE Symposium on Security and Privacy*, 2002.

[65] F. Mansmann, D. Keim, S. North, B. Rexroad, and D. Sheleheda . Visual Analysis of Network Traffic. In *Proc. Infovis*, 2007.

[66] A. Feldmann, A. G. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. In *Proc. of ACM SIGCOMM*, 2000.

[67] D. Fisher, D. A. Maltz, A. Greenberg, X. Wang, H. Warncke, G. Robertson, and M. Czerwinski. Using Visualization to Support Network and Application Management in a Data Center . In *Proceedings of INM*, 2008.

[68] B. Fortz, J. Rexford, and M. Thorup. Traffic Engineering with Traditional IP Routing Protocols. IEEE Communications Magazine, Oct. 2002.

[69] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS Weights in a Changing World. *IEEE Journal on Selected Areas in Communications*, 20(4), May 2002.

[70] L. Foschini, A. V. Thapliyal, L. Cavallaro, C. Kruegel, and G. Vigna. A Parallel Architecture for Stateful, High-Speed Intrusion Detection. In *Proc. ICISS*, 2008.

[71] G. Nemhauser, L. Wosley, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[72] G. Robertazzi and S. C. Schwartz. An accelerated sequential algorithm for producing D-optimal designs. *SIAM Journal of Scientific and Statistical Computing*, 10(2):341–358, Mar. 1989.

[73] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Meyers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM CCR*, 35(5), Oct. 2005.

[74] C. Guestrin, A. Krause, and A. Singh. Near-optimal Sensor Placements in Gaussian Processes. In *Proc. of ICML*, 2005.

[75] S. Guha, A. McGregor, and S. Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *Proceedings of ACM Symposium on Discrete Algorithms (SODA)*, 2006.

[76] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational Experiences with High-Volume Network Intrusion Detection. In *Proc. ACM CCS*, 2004.

[77] H. Dreger, A. Feldmann, V. Paxson and R. Sommer. Predicting the Resource Consumption of Network Intrusion Detection Systems. In *Proc. RAID*, 2008.

[78] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood. Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing. In *Proc. ACM SIGCOMM*, 2005.

[79] N. Hohn and D. Veitch. Inverting Sampled Traffic. In *Proc. of IMC*, 2003.

[80] J. C. Mogul, Y. M. Chan, and T. Kelly. Design, implementation, and evaluation of duplicate transfer detection in HTTP . In *Proc. of NSDI*, 2004.

[81] J. Gonzalez, V. Paxson, and N. Weaver. Shunting: A Hardware/Software Architecture for Flexible, High-Performance Network Intrusion Prevention. In *Proc. ACM CCS*, 2007.

[82] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA - An Open Platform for Gigabit-rate Network Switching and Routing . In *Proc. IEEE MSE*, 2007.

[83] D. A. Joseph, A. Tavakoli, and I. Stoica. A Policy-aware Switching Layer for Data Centers. In *Proc. SIGCOMM*, 2008.

[84] K. Argyraki, P. Maniatis, D. Cheriton, and S. Shenker. Providing Packet Obituaries. In *Proc. of Hotnets*, 2004.

[85] A. Kalai and S. Vempala. Efficient Algorithms for Online Decision Problems. *Journal of Computer System Sciences*, 71(3), Oct. 2005.

[86] T. Karagiannis, D. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *Proc. of ACM SIGCOMM*, 2005.

[87] A. D. Keromytis, V. Misra, and D. Rubenstein. Secure Overlay Services. In *Proc. SIGCOMM*, 2002.

[88] K. Keys, D. Moore, and C. Estan. A Robust System for Accurate Real-time Summaries of Internet Traffic. In *Proc. SIGMETRICS*, 2005.

[89] R. Kompella and C. Estan. The Power of Slicing in Internet Flow Measurement. In *Proc. of IMC*, 2005.

[90] R. R. Kompella, S. Singh, and G. Varghese. On scalable attack detection in the network. In *Proc. IMC*, 2004.

[91] A. Krause and C. Guestrin. Near-optimal Observation Selection Using Submodular Functions. In *Proc. of AAAI*, 2007.

[92] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal Sensor Placements: Maximizing Information while Minimizing Communication Cost . In *Proc. of IPSN*, 2006.

[93] A. Krause, B. McMahan, C. Guestrin, and A. Gupta. Selecting Observations Against Adversarial Objectives. In *Proc. of NIPS*, 2007.

[94] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proc. ACM IMC*, 2003.

[95] S. Krothapalli, X. Sun, Y.-W. Sung, S. A. Yeo, and S. Rao. A Toolkit for Automating and Visualizing VLAN Configuration. In *Proceedings of ACM CCS Workshop on Assurable and Usable Security Configuration (SafeConfig)*, 2009.

[96] A. Kumar, M. Sung, J. Xu, and J. Wang. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Distribution. In *Proc. of ACM SIGMETRICS*, 2004.

[97] A. Kumar, M. Sung, J. Xu, and E. Zegura. A data streaming algorithm for estimating subpopulation flow size distribution. In *Proceedings of ACM SIGMETRICS*, 2005.

[98] A. Kumar and J. Xu. Sketch Guided Sampling – Using On-Line Estimates of Flow Size for Adaptive Data Collection. In *Proc. IEEE Infocom*, 2006.

[99] A. Lakhina, M. Crovella, and C. Diot. Diagnosing Network-Wide Traffic Anomalies. In *Proc. of ACM SIGCOMM*, 2004.

[100] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. ACM SIGCOMM*, 2005.

[101] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. Kolaczyk, and N. Taft. Structural Analysis of Network Traffic Flows. In *Proc. of ACM SIGMETRICS*, 2004.

[102] A. Lall, V. Sekar, J. Xu, M. Ogihara, and H. Zhang. Data Streaming Algorithms for Estimating Entropy of Network Traffic. In *Proc. of ACM SIGMETRICS*, 2006.

[103] V. T. Lam, M. Mitzenmacher, and G. Varghese. Carousel: Scalable Logging for Intrusion Prevention Systems. In *Proc. NSDI*, 2010.

[104] A. Le, E. Al-Shaer, and R. Batouba. Correlation-Based Load Balancing for Intrusion Detection and Prevention Systems. In *Proc. SECURECOMM*, 2008.

[105] A. Le, E. Al-Shaer, and R. Batouba. On Optimizing Load Balancing of Intrusion Detection and Prevention Systems. In *Proc. INFOCOM*, 2008.

[106] S. Lee, T. Wong, and H. Kim. Secure Split Assignment Trajectory Sampling: A Malicious Router Detection System. In *Proc. of IEEE/IFIP DSN*, 2006.

[107] J. Li, M. Sung, J. Xu, L. Li, and Q. Zhao. Large-scale IP Traceback in High-speed Internet: Practical Techniques and Theoretical Foundation. In *Proc. of IEEE Symposium of Security and Privacy*, 2004.

[108] X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, W. Hong, and G. Iannaccone. MIND: A Distributed Multidimensional Indexing for Network Diagnosis. In *Proc. of IEEE INFOCOM*, 2006.

[109] K. Ligett, S. Kakade, and A. T. Kalai. Playing Games with Approximation Algorithms. In *Proc. STOC*, 2007.

[110] Y. Liu, L. Zhang, and Y. Guan. Sketch-based Streaming PCA Algorithm for Network-wide Traffic Anomaly Detection . In *Proc. ICDCS*, 2010.

[111] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter Braids: A Novel Counter Architecture for Per-Flow Measurement. In *Proc. SIGMETRICS*, 2008.

[112] M. Allman, C. Kreibich, V. Paxson, R. Sommer and N. Weaver. Principles for Developing Comprehensive Network Visibility. In *USENIX Workshop on Hot Topics in Security*, 2008.

[113] M. Kodialam, T. V. Lakshman, and Sudipta Sengupta. Configuring Networks with Content Filtering Nodes with Applications to Network Security. In *Proc. INFOCOM*, 2005.

[114] M. Minoux. Accelerated Greedy Algorithms for Maximizing Submodular Set Functions. In *Proc. of 8th IFIP Conference, Springer-Verlag*, 1977.

[115] M. P. Collins and M. K. Reiter. Hit-list Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. In *Proc. RAID*, 2007.

[116] H. Madhyastha and B. Krishnamurthy. A Generic Language for Application-Specific Flow Sampling. *ACM CCR*, 38(2):7–15, Apr. 2008.

[117] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring Link Weights using End-to-End Measurements. In *Proc. of IMW*, 2002.

[118] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. Is Sampled Data Sufficient for Anomaly Detection? In *Proc. of IMC*, 2006.

[119] A. Mankin, D. Massey, C.-L. Wu, S. F. Wu, and L. Zhang. On Design and Evaluation of ”Intention-Driven” ICMP Traceback. In *Proc. of IEEE ICCCN*, 2001.

[120] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE Transactions on Networking*, 8(5), Oct 2000.

[121] M. Molina, S. Niccolini, and N. Duffield. A Comparative Experimental Study of Hash Functions Applied to Packet Sampling. In *Proc. of International Teletraffic Congress (ITC)*, 2005.

[122] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. *SIGOPS Operating Systems Review*, 33(5):217–231, 1999.

[123] S. Muthukrishnan. Data streams: algorithms and applications. `http://athos.rutgers.edu/~muthu/stream-1-1.ps`.

[124] E. N. Duffield. A Framework for Packet Selection and Reporting. IETF Draft `http://www.ietf.org/internet-drafts/draft-ietf-psamp-framework-10.txt`, 2005.

[125] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[126] S. Narain, G. Levin, S. Malik, and V. Kaul. Declarative infrastructure configuration synthesis and debugging. *Journal of Network and Systems Management*, 16(3), 2008.

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

[128] K. Park and H. Lee. On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets . In *Proc. of ACM SIGCOMM*, 2001.

[129] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23–24):2435–2463, 1999.

[130] P. Phaal, S. Panchen, and N. Mckee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks . RFC 3176, 2001.

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

[132] M. Rabin. Fingerprinting by random polynomials. Technical report, Harvard University, 1981. Technical Report, TR-15-81.

[133] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4), Dec. 1987.

[134] A. Ramachandran, S. Seetharaman, and N. Feamster. Fast Monitoring of Traffic Subpopulations . In *Proc. of IMC*, 2008.

[135] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani. Building a Better Mousetrap. Georgia Tech Technical Report, GIT-CSS-07-01, 2007.

[136] M. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient Hardware Hashing Functions for High Performance Computers. *IEEE Transactions on Computers*, 46(12):1378–1381, 1997.

[137] S. Raza, G. Huang, C.-N. Chuah, S. Seetharaman, and J. P. Singh. MeasuRouting: A Framework for Routing Assisted Traffic Monitoring. In *Proc. INFOCOM*, 2010.

[138] B. Ribeiro, D. Towsley, T. Ye, and J. Bolot. Fisher information of sampled packets: an application to flow size estimation. In *Proc. IMC*, 2006.

[139] M. Roughan. Simplifying the Synthesis of Internet Traffic Matrices. *ACM SIGCOMM CCR*, 35(5), 2005.

[140] M. Roughan, A. Greenberg, C. Kalmanek, M. Rumsewicz, J. Yates, and Y. Zhang. Experience in Measuring Internet Backbone Traffic Variability:Models, Metrics, Measurements and Meaning. In *Proc. of International Teletraffic Congress (ITC)*, 2003.

[141] S. Kumar and P. Crowley. Segmented Hash: An Efficient Hash Table Implementation for High Performance Networking Subsystems. In *Proc. ACM ANCS*, 2005.

[142] S. Rhea, K. Liang, and E. Brewer. Value-based web caching. In *Proc. of WWW*, 2003.

[143] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proc. of ACM SIGCOMM*, 2000.

[144] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M. Kao, and G. Memik. Reverse hashing for high-speed network monitoring: Algorithms, evaluation, and applications. In *Proc. IEEE INFOCOM*, 2006.

[145] V. Sekar, N. Duffield, K. van der Merwe, O. Spatscheck, and H. Zhang. LADS: Large-scale Automated DDoS Detection System. In *Proc. of USENIX ATC*, 2006.

[146] V. Sekar, A. Gupta, M. K. Reiter, and H. Zhang. Coordinated Sampling sans Origin-Destination Identifiers: Algorithms and Analysis . In *Proc. COMSNETS*, 2010.

[147] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. Kompella, and D. G. Andersen. cSamp: A System for Network-Wide Flow Monitoring. In *Proc. of NSDI*, 2008.

[148] A. Shaikh, M. Goyal, A. Greenberg, R. Rajan, and K. Ramakrishnan. An OSPF topology server: Design and evaluation. *IEEE Journal on Selected Areas in Communications*, 20(4), 2002.

[149] A. Shaikh and A. Greenberg. OSPF Monitoring: Architecture, Design and Deployment Experience. In *Proc. of NSDI*, 2004.

[150] M. R. Sharma and J. W. Byers. Scalable Coordination Techniques for Distributed Network Monitoring. In *Proc. of PAM*, 2005.

[151] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting . In *Proc. OSDI*, 2004.

[152] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C. lin Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. DIDS (distributed intrusion detection system), - motivation, architecture, and an early prototype. In *Proc. National Computer Security Conference,* 1991.

[153] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-Based IP Traceback . In *Proc. of ACM SIGCOMM*, 2001.

[154] A. Solue, A. Lakhina, N. Taft, K. Papagiannaki, K. Salamatian, A. Nucci, M. Crovella, and C. Diot. Traffic Matrices: Balancing Measurements, Inference, and Modeling. In *Proc. of ACM SIGMETRICS*, 2005.

[155] R. Sommer and V. Paxson. Exploiting Independent State for Network Intrusion Detection. In *Proc. ACSAC*, 2005.

[156] R. Sommer, V. Paxson, and N. Weaver. An Architecture for Exploiting Multi-Core Processors to Parallelize Network Intrusion Prevention. *Concurrency and Computation: Practice and Experience, Wiley*, 21(10):1255–1279, 2009.

[157] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. of ACM SIGCOMM*, 2002.

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

[159] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating Network Monitors: Complexity, heuristics and coverage. In *Proc. of IEEE INFOCOM*, 2005.

[160] Y.-W. E. Sung, S. Rao, G. Xie, and D. Maltz. Towards Systematic Design of Enterprise Networks. In *Proceedings of ACM CoNEXT*, 2008.

[161] M. Sviridenko. A note on maximizing a submodular set function subject to knapsack constraint. *Operations Research Letters*, pages 41–43, 2004.

[162] Symantec Corporation. Deepsight. `http://www.enterprisesecurity.symantec.com`.

[163] T. McCartney, K. Goldman, and D. Saff. EUPHORIA: End User Construction of Direct Manipulation User Interfaces for Distributed Algorithms. *Software Concepts and Tools*, (4):147–159, 1995.

[164] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An architecture for internet data transfer. In *Proc. of NSDI*, 2006.

[165] J. Ullrich. Dshield.org. `http://www.dshield.org`.

[166] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney. The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware. In *Proc. of RAID*, 2007.

[167] G. Varghese. *Network Algorithmics*. Morgan Kaufman, 2005.

[168] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum. New Streaming Algorithms for Fast Detection of Superspreaders . In *Proc. NDSS*, 2005.

[169] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of OSDI*, 2002.

[170] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, 1997.

[171] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, , and H. M. Levy. On the scale and performance of cooperative Web proxy caching. In *Proc. of SOSP*, 1999.

[172] L. A. Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research*, 7(3):410–425, 1982.

[173] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang. Worm Origin Identification Using Random Moonwalks. In *Proc. of IEEE Symposium on Security and Privacy*, 2005.

[174] K. Xu, Z.-L. Zhang, and S. Bhattacharya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *Proc. of ACM SIGCOMM*, 2005.

[175] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Racke. Optimal oblivious routing in polynomial time. In *Proc. STOC*, 2003.

[176] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song and M.-Y. Kao. Detecting Stealthy Attacks Using Online Histograms. In *Proc. IWQoS*, 2007.

[177] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *Proc. of ACM SIGCOMM*, 2005.

[178] F. Yu, R. H. Katz, and T. V. Lakshman. Gigabit Rate Packet Pattern-Matching Using TCAM. In *Proc. ICNP*, 2004.

[179] F. Yu, T. V. Lakshman, M. A. Motoyama, and R. H. Katz. SSA: A Power and Memory Efficient Scheme to Multi-Match Packet Classification. In *Proc. ANCS*, 2005.

[180] L. Yuan, C.-N. Chuah, and P. Mohapatra. ProgME: Towards Programmable Network MEasurement. In *Proc. SIGCOMM*, 2007.

[181] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the Characteristics and Origins of Internet Flow Rates. In *Proc. of ACM SIGCOMM*, 2002.

[182] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the Constancy of Internet Path Properties. In *Proc. IMW*, 2001.

[183] Y. Zhang, Z. Ge, M. Roughan, and A. Greenberg. Network Anomography. In *Proc. of IMC*, 2005.

[184] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast Accurate Computation of Large-scale IP Traffic Matrices from Link Loads. In *Proc. of ACM SIGMETRICS*, 2003.

[185] Q. Zhao, Z. Ge, J. Wang, and J. Xu. Robust Traffic Matrix Estimation with Imperfect Information: Making use of Multiple Data Sources. In *Proc. of ACM SIGMETRICS*, 2006.

[186] Q. Zhao, A. Kumar, and J. Xu. Joint Data Streaming and Sampling Techniques for Detection of Super Sources and Destinations. In *Proc. of IMC*, 2005.

[187] Q. Zhao, J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. In *Proc. of ACM SIGMETRICS*, 2006.