Network-Wide Deployment of Intrusion Detection and Prevention Systems

**Vyas Sekar†, Ravishankar Krishnaswamy†,
Anupam Gupta†, Michael K. Reiter†† [1]**

Jun 23, 2010
CMU-CS-10-124

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1] † Carnegie Mellon University †† UNC Chapel-Hill
This work was supported in part by NSF awards CNS-0326472, CNS-0433540, and ANI-0331653.

Keywords: Network Monitoring, Intrusion Detection, Intrusion Prevention, Algorithms

Abstract

Traditional research efforts for scaling NIDS and NIPS systems using parallelization and hardware-assisted acceleration have largely focused on a single-vantage-point view. In this chapter, we explore a different design alternative that exploits spatial, network-wide opportunities for distributing NIDS and NIPS functions throughout a network. We present systematic models that capture the operational constraints and requirements in deploying network-wide NIDS and NIPS capabilities. These formulations enable network administrators to optimally leverage their infrastructure toward their security objectives. For the NIDS case, we design a linear programming formulation for partitioning NIDS functions across a network to ensure that no node is overloaded. We also describe and evaluate a prototype implementation using Bro. For NIPS, we show how to maximally reduce unwanted traffic using special hardware-assisted capabilities. In this case, the hardware constraints make the optimization problem NP-hard, and we design and implement practical approximation algorithms based on randomized rounding. These results have immediate practical implications as: (1) enterprise networks become larger and their traffic volumes increase; and (2) ISPs increasingly deploy NIDS/NIPS capabilities as in-network defenses. By leveraging network-wide opportunities for distributing NIDS/NIPS responsibilities, our work effectively complements efforts to scale single-vantage-point NIDS and NIPS.

1 Introduction

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., [46]). 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., [40]). 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 [4, 5].

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 [6, 9, 17, 37]. 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 [34]. 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 [21].

There are several efforts for scaling NIDS and NIPS (e.g., [8, 16, 25, 41, 46]) 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.

2 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 [34].

2.1 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

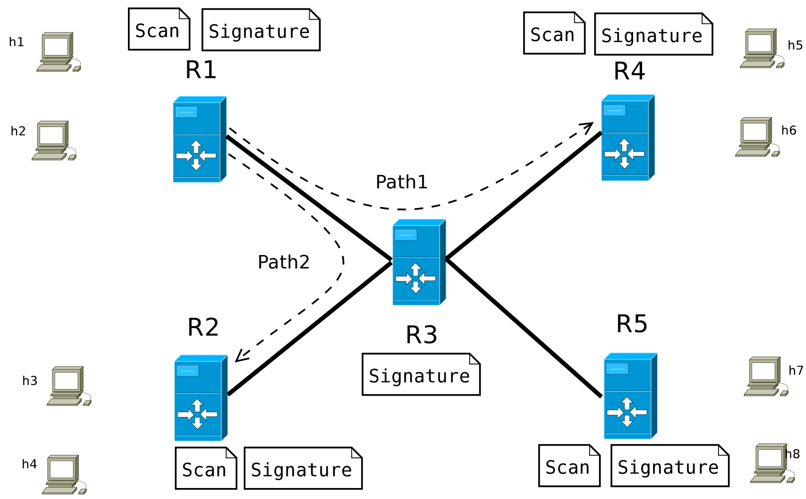


Figure 1: Example of network-wide NIDS instrumentation

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.

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 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, $P_{i1} = \{\mathrm{R1}, \mathrm{R3}, \mathrm{R4}\}$ is the set of nodes that can observe (and, we assume, recognize) traffic matching $\mathcal{T}_{i1}$, and $P_{i2} = \{\mathrm{R1}, \mathrm{R3}, \mathrm{R2}\}$ 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 $P_{i1} = P_{i2} = \{\mathrm{R1}\}$ (for hosts h1–h2), $P_{i3} = P_{i4} = \{\mathrm{R2}\}$, 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 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.

2.2 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 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}.
- 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 [19].
- 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.

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

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

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

$$\forall j,\ MemLoad_j = \frac{\sum_i \sum_k MemReq_i \times T_{ik}^{items} \times d_{ikj}}{MemCap_j} \tag{2}$$

$$\forall j,\ CpuLoad_j = \frac{\sum_i \sum_k CpuReq_i \times T_{ik}^{pkts} \times d_{ikj}}{CpuCap_j} \tag{3}$$

$$\forall j,\ CpuLoad \geq CpuLoad_j \tag{4}$$

$$\forall j,\ MemLoad \geq MemLoad_j \tag{5}$$

$$\forall i, \forall k, \forall j,\ 0 \leq d_{ikj} \leq 1 \tag{6}$$

Eq (1) says that the all the traffic in each coordination unit for each class should be monitored. Eq (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 [19]. 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 (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 [19]. 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 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 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.

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

2.3 Implementation in Bro

We implement the above coordination functions in the Bro IDS [34]. Bro is logically divided into two parts (Figure 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 connection 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 [33].

We consider two implementation alternatives: (1) delaying the sampling checks in Figure 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 2.4) 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.

[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

COORDINATEDNIDS $(pkt, R_j, Manifest(R_j))$

$\{C_i\}_i \leftarrow$ GETCLASS (pkt)
// Each packet may be analyzed by multiple modules
foreach class C_i **do**
    $k \leftarrow$ GETCOORDUNIT (pkt, i)
    // HASH returns a value in $[0, 1]$
    // Specific packet fields used for HASH
    // depend on semantics of C_i
    $h_{pkt} \leftarrow$ HASH(pkt, i)
    if $h_{pkt} \in HashRange(i, k, j)$ **then**
        Run class C_i for pkt

Figure 3: Coordinated NIDS algorithm on node R_j

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 information. 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.

2.4 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 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 in-

is initialized.

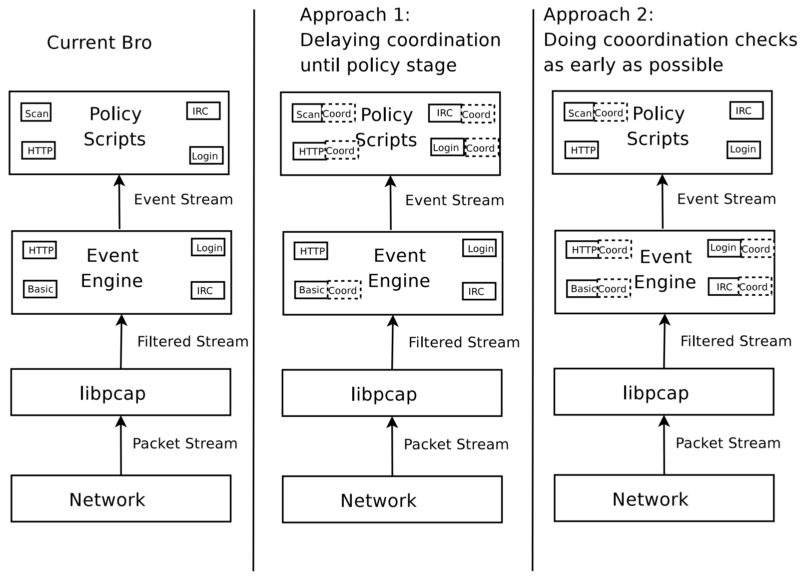


Figure 4: Implementing the coordination functionality in Bro. The "coord" boxes indicate where changes were needed to add in coordination checks in Bro. For some modules (e.g., `Scan`), the coordination checks have to be in the policy engine.

terested 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 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

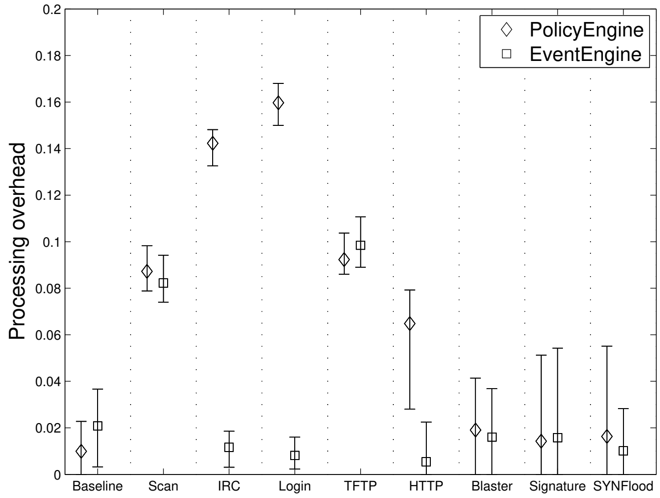


Figure 5: CPU overhead with the coordination-enabled Bro prototypes for different modules

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 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. [19] 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 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

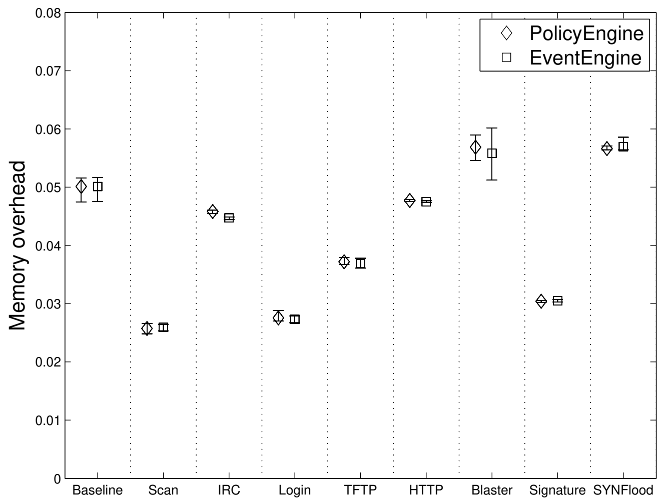


Figure 6: Memory overhead with the coordination-enabled Bro prototype for different modules

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 7 and 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 5 and 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), the results are significantly different in the network-wide case (Figure 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.

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

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

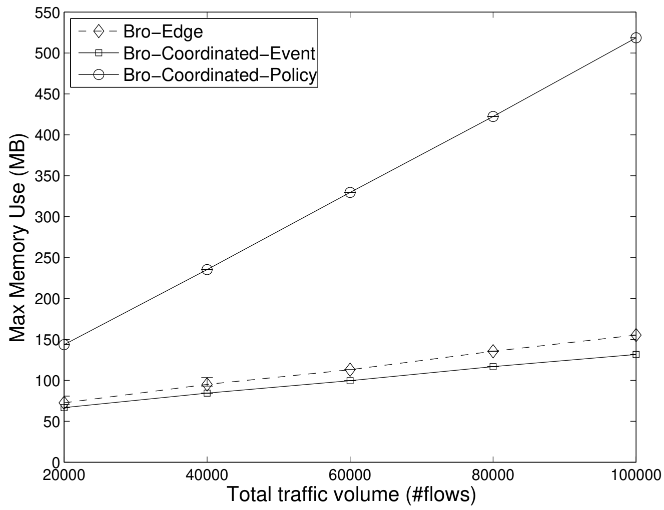


Figure 7: Max memory usage across the network as the total traffic volume increases

former category. Thus, our duplicate instances are indicative of how a NIDS like Bro would be configured with additional modules in practice.

Figures 9 and 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 11 and 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 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.

2.5 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 coordina-

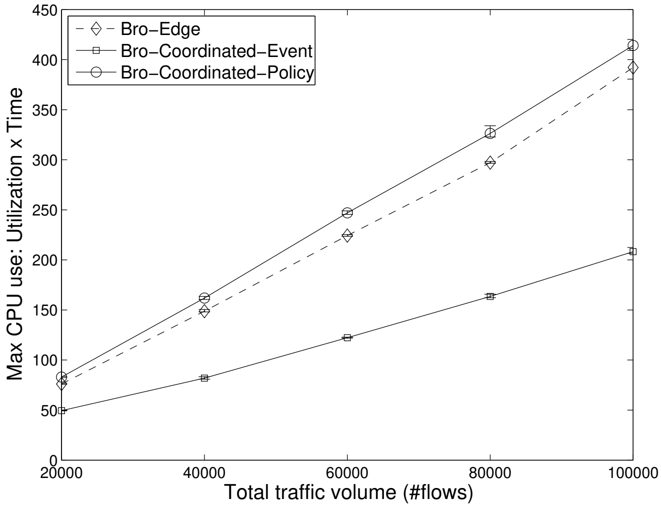


Figure 8: Max CPU usage across the network as the total traffic volume increases

tion capabilities–allowing different granularities of connections, creating more 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 2.1, 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 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 (1) to k instead of 1 and solve the rest of the LP as before. While converting the LP solution into sampling manifests (Figure 2), we proceed as before, except that we logically wraparound the range every time it exceeds 1.

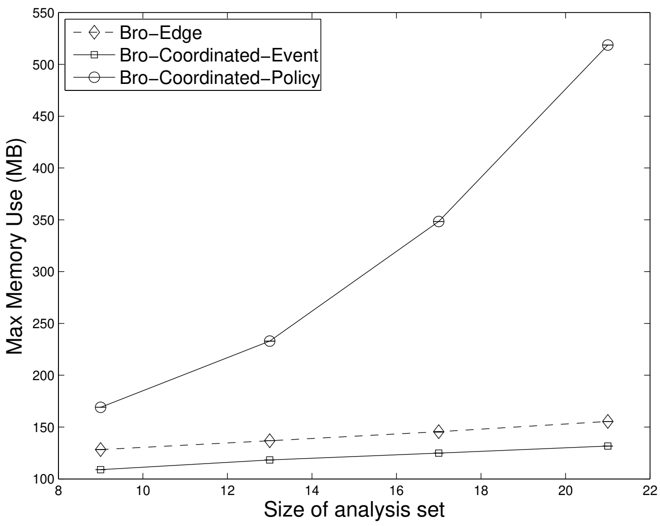


Figure 9: Max network-wide memory use with more modules

3 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.

3.1 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., [47, 48]). However, such hardware capabilities are expensive and power-hungry. This places additional economic and technological limits (imposed by power and cooling requirements)

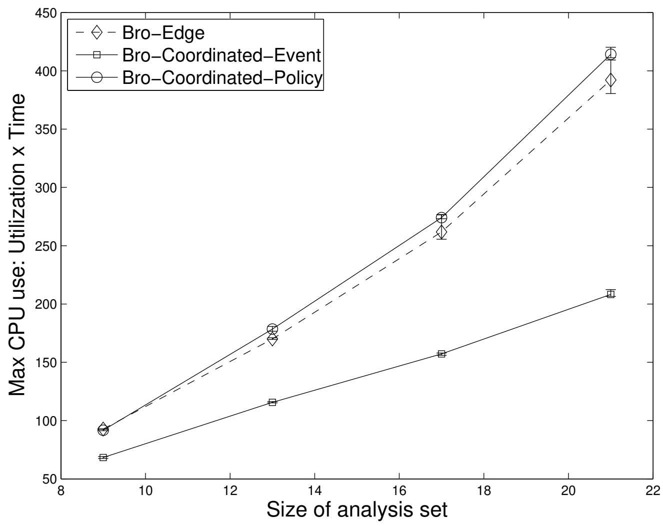


Figure 10: Max network-wide CPU use with more modules

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 the model from the previous section to model the use of special-purpose hardware for NIPS functions.

3.2 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.

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.

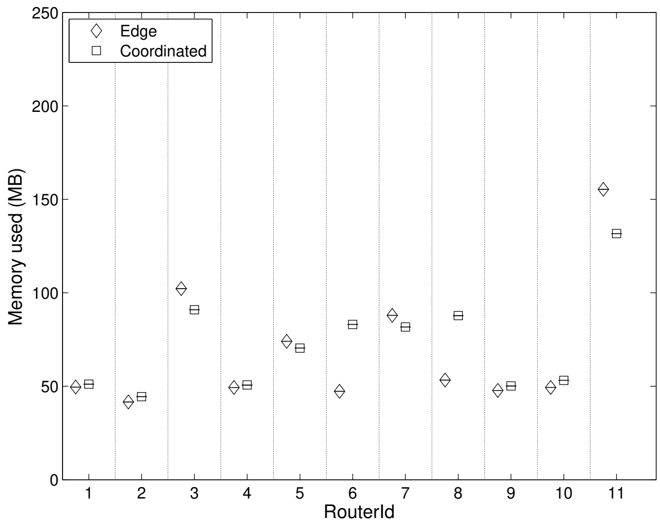


Figure 11: Memory load on each NIDS node in the network

- 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.

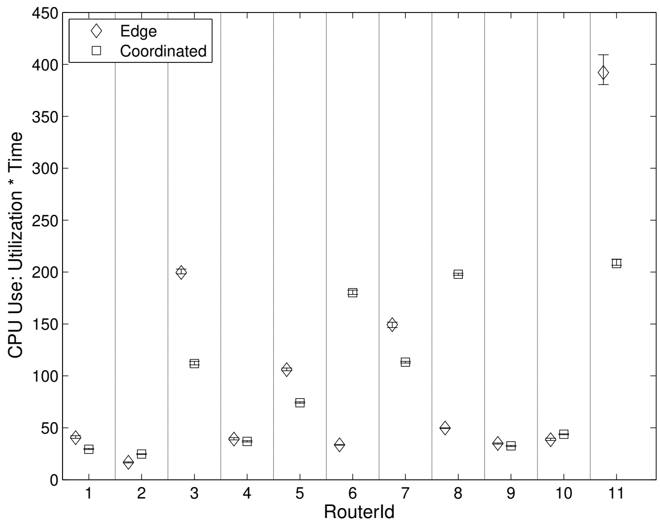


Figure 12: CPU load on each NIDS node in the network

$$\text{Maximize} \sum_i \sum_k \sum_{j, R_j \in P_{ik}} T_{ik}^{items} \times Match_{ki} \times Dist_{kj} \times d_{ikj} \tag{7}$$

subject to

$$\forall j, \sum_i CamReq_i \times e_{ij} \leq CamCap_j \tag{8}$$

$$\forall j, \sum_k \sum_i T_{ik}^{items} \times MemReq_i \times d_{ikj} \leq MemCap_j \tag{9}$$

$$\forall j, \sum_k \sum_i T_{ik}^{pkts} \times CpuReq_i \times d_{ikj} \leq CpuCap_j \tag{10}$$

$$\forall k, \forall i, \sum_{j, R_j \in P_{ik}} d_{ikj} \leq 1 \tag{11}$$

$$\forall j, \forall i, \forall k, d_{ikj} \leq e_{ij} \tag{12}$$

$$\forall k, \forall i, \forall j, \ d_{ikj} \geq 0 \tag{13}$$

$$\forall i, \forall j, \ e_{ij} \in \{0, 1\} \tag{14}$$

The objective in Eq (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 2), we can simply add up the contributions across the different nodes.

Eq (8) models the constraint on the number of rules that can be enabled in the constrained hardware on each node. Eq (9) and Eq (10) model the aggregate memory and processing load on each node. Eq (12) is a sanity check to ensure that a node cannot apply a rule C_i unless it has been enabled and Eq (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 (9) and Eq (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 (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.

3.3 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 (11) and (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.

3.4 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 [35]. Figure 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 (8)–(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 (8), we set $\widehat{e_{ij}}$ to 1 only with probability $\frac{e^*_{ij}}{\alpha}$ (line 5 of Figure 13). While this ensures that most constraints in Eq (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 (9)–(11), we ensure that the solution $\{\widehat{e_{ij}}\}_{ij}$, $\{\widehat{d_{ikj}}\}_{ikj}$ after the loop in lines 4–9 satisfies Eqs (9)–(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 (7)–(13), and with Eq (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 (7)–(14). We show in in the next section that the process in Figure 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 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 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 (9)–(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

```
RandomizedRounding
   // Create LP relaxation
   Replace "e_ij ∈ {0,1}" in Eq (14) with "0 ≤ e_ij ≤ 1".
   Solve the LP relaxation to obtain {e*_ij}_ij and {d*_ikj}_ikj.
   ∀k, i, j, ϵ_ikj ← d*_ikj / e*_ij.
   repeat
      ∀i, j, Randomly set ê_ij ← 1 with probability e*_ij/α,
      and ê_ij ← 0 otherwise
      ∀k, i, j, d̂_ikj ← ϵ_ikj ê_ij.
      Check if any constraint in Eqs (9)–(11)
      is violated by a factor more than β log N.
      If yes, call this trial a failure.
   until not failure
   If for some j the constraint Eq (8) is violated, arbitrarily set
      some ê_ij to 0 until all constraints Eq (8) are satisfied.
   ∀k, i, j, ϵ_ikj ← ϵ_ikj / (β log N).
   ∀k, i, j, d̂_ikj ← ϵ_ikj ê_ij.
   Output ê_ij and d̂_ikj.
```

Figure 13: Approximation algorithm for the NIPS deployment problem via randomized rounding.

1 without violating Eq (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.

3.5 Sketch of Rounding Argument

We now present the analysis of the rounding algorithm from Section 3.4. 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 (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}:

$$1 \leq \mathsf{Opt}_{LP} \leq N^4 \tag{15}$$

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 (9):

$$\mathbb{E}\left[T_k^{items} \times MemReq_i \times \widehat{d_{ikj}}\right] \leq \frac{MemCap_j}{\alpha} \tag{16}$$

We can use linearity of expectation, to also get that the expected value for each constraint in Eqs (10) and (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 (9)–(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 (9)–(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 (11) and at most $2N$ other constraints from equations Eq (9) and Eq (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 (8), but using which none of the constraints Eqs (9)–(11) are violated by more than a factor of $\beta \log N$. Before we worry about the violations for constraints Eq (8), let us bound the expected value of the objective function for the rounding procedure. From linearity of expectation, we have

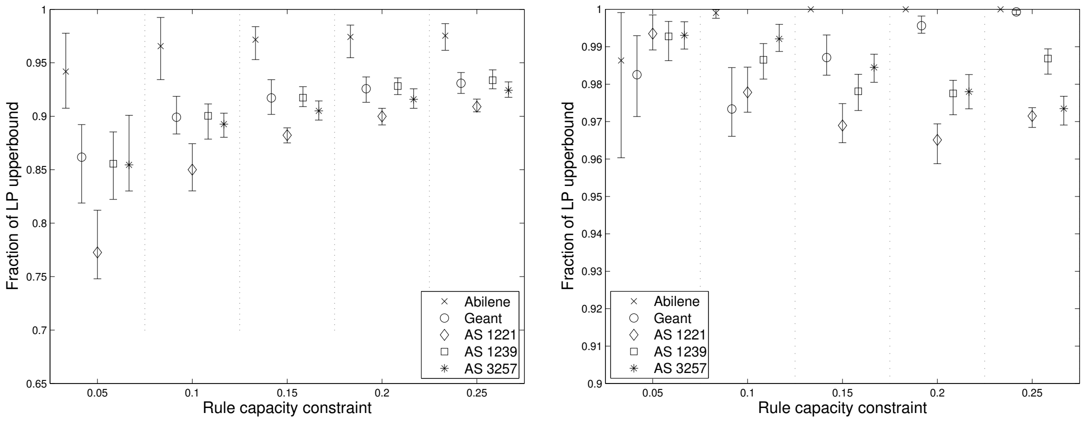
$$\mathbb{E}\left[\sum_k \sum_{j,R_j \in P_k} \sum_i T_k^{items} \times Match_{ki} \times Dist_{kj} \times d_{ikj}\right] \geq \frac{\mathsf{Opt}_{LP}}{\alpha} \tag{17}$$

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 (15)). If $\mathcal{E}$ denotes a failure event, we know that

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

and hence

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



(a) Just rounding

(b) Rounding + Greedy

Figure 14: Performance of the approximation algorithms with a uniform rule match rate distribution

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 (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 (8) exceeds the bound $CamCap_{j'}$. But we know that over all the other rules, the expected load satisfies

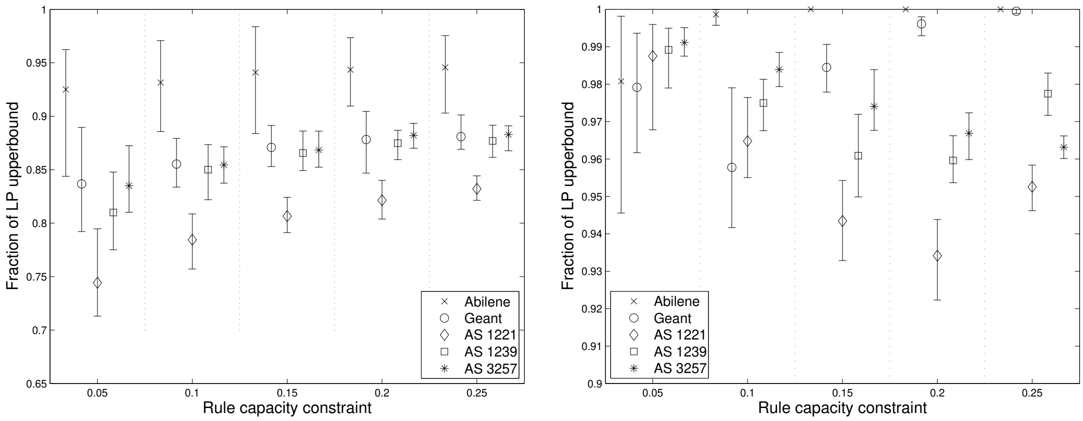
$$\mathbb{E}\left[\sum_{i \neq i'} CamReq_i \times \widehat{e_{ij'}}\right] \leq (CamCap_j - e_{i'j'} \times CamReq_{i'})/\alpha$$

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 (9)–(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.

3.6 Evaluation

For this evaluation, we use network topologies from educational backbones (Internet2 and Geant) and tier-1 ISP backbone topologies inferred by Rocketfuel [42]. We construct ingress-egress paths for each pair of nodes using shortest-path routing [32]. We use a gravity model traffic matrix based



(a) Just rounding

(b) Rounding + Greedy

Figure 15: Performance of the approximation algorithms with an exponential rule match rate distribution

on city populations [38]. 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 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 14 and 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 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.

[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.

3.7 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 [21] 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.

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

$$\text{subject to}$$

$$\forall j, \; \sum_k \sum_i T_{ik}^{items} \times MemReq_i \times d_{ikj} \leq MemCap_j \tag{18}$$

$$\forall j, \; \sum_k \sum_i T_{ik}^{pkts} \times CpuReq_i \times d_{ikj} \leq CpuCap_j \tag{19}$$

$$\forall k, \forall i, \; \sum_{j, R_j \in P_k} d_{ikj} \leq 1 \tag{20}$$

$$\forall k, \forall i, \forall j, \; d_{ikj} \geq 0 \tag{21}$$

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.

[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.

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 [21] 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.

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 3.1. $\frac{E[cost(FPL(\epsilon)) - mincost_\tau]}{\tau} \leq \sqrt{\frac{DRA}{\tau}}$ *[21].*

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 3.4 (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.

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

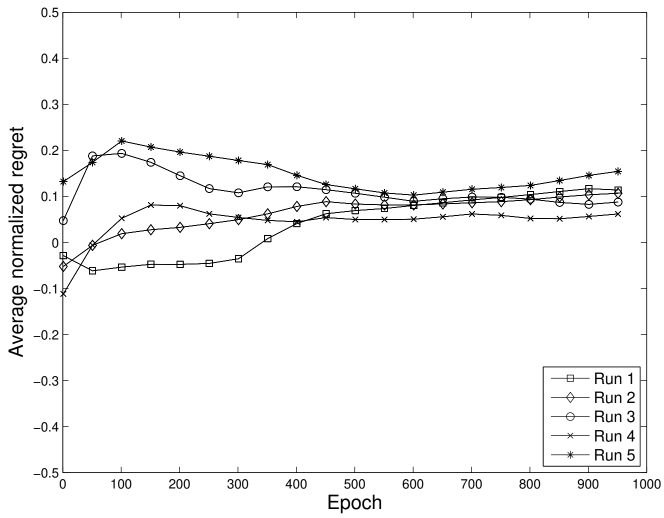


Figure 16: Result showing the normalized regret over time for different runs of the online adaptation algorithm. We normalize the regret by the objective value of the best static solution.

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 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, we will explore how well such strategies perform in the presence of strategic adversaries and extend this framework to the general formulation from Section 3.2.[7]

4 Related Work

Network management:: Several recent efforts have demonstrated the benefits of a coordinated approach for network management [6, 9, 15, 17, 50]. In the context of monitoring and sampling, hash-based packet selection to coordinate monitoring responsibilities has been used in the context of Trajectory Sampling [12] and cSamp [37]. 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 [10, 43]. These show that the problems are NP-hard and propose greedy algorithms. Kodialam et al [30] consider the problem

[7]There are known extensions for the case where Λ is an approximation algorithm [21, 28].

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., [8,16,25,26,41,46]), hardware-assisted acceleration (e.g., [20]), more efficient algorithms (e.g., [24]), models for understanding their resource consumption (e.g., [18, 19]), and optimizing rule patterns (e.g, [2, 3, 13, 47, 48]). 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., [1,7,14,22,39,44,45]). As applications and attacks become distributed, we need to aggregate information across a network for effective analysis [23, 27, 29]. For example, understanding peer-to-peer traffic [11], hit-list worms [31], and understanding DDoS attacks [36] 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).

5 Discussion

Provisioning and Upgrades:: So far, we considered the problem of optimally configuring a NIDS/NIPS infrastructure. We can extend the formulations from Sections 2.2 and 3.2 to describe what-if provisioning scenarios: where should an administrator add more resources (e.g., [46]) or augment existing deployments with more powerful hardware (e.g., [20]).

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 [49]. 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 [40]. Also, adding in redundant functionality as outlined in Section 2.5 can further reduce the impact of routing changes.

6 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.

References

[1] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proc. RAID*, 2001.

[2] 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.

[3] 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.

[4] Arbor networks. `http://www.arbor.com`.

[5] AT&T Enterprise Threat Management. `http://www.business.att.com/enterprise/Family/business-continuity-enterprise/threat-management-enterprise/`.

[6] H. Ballani and P. Francis. CONMan: A Step Towards Network Manageability. In *Proc. of ACM SIGCOMM*, 2007.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] M. P. Collins and M. K. Reiter. Finding Peer-to-Peer File-sharing using Coarse Network Behaviors. In *Proc. of ESORICS*, 2006.

[12] N. Duffield and M. Grossglauser. Trajectory Sampling for Direct Traffic Observation. In *Proc. of ACM SIGCOMM*, 2001.

[13] E. W. Fulp. Optimization of network firewalls policies using directed acyclic graphs. In *Proc. Internet Management Conference*, 2005.

[14] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proc. IEEE Symposium on Security and Privacy*, 2002.

[15] 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.

[16] 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.

[17] 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.

[18] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational Experiences with High-Volume Network Intrusion Detection. In *Proc. ACM CCS*, 2004.

[19] H. Dreger, A. Feldmann, V. Paxson and R. Sommer. Predicting the Resource Consumption of Network Intrusion Detection Systems. In *Proc. RAID*, 2008.

[20] J. Gonzalez, V. Paxson, and N. Weaver. Shunting: A Hardware/Software Architecture for Flexible, High-Performance Network Intrusion Prevention. In *Proc. ACM CCS*, 2007.

[21] A. Kalai and S. Vempala. Efficient Algorithms for Online Decision Problems. *Journal of Computer System Sciences*, 71(3), Oct. 2005.

[22] A. D. Keromytis, V. Misra, and D. Rubenstein. Secure Overlay Services. In *Proc. SIGCOMM*, 2002.

[23] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. ACM SIGCOMM*, 2005.

[24] V. T. Lam, M. Mitzenmacher, and G. Varghese. Carousel: Scalable Logging for Intrusion Prevention Systems. In *Proc. NSDI*, 2010.

[25] A. Le, E. Al-Shaer, and R. Batouba. Correlation-Based Load Balancing for Intrusion Detection and Prevention Systems. In *Proc. SECURECOMM*, 2008.

[26] A. Le, E. Al-Shaer, and R. Batouba. On Optimizing Load Balancing of Intrusion Detection and Prevention Systems. In *Proc. INFOCOM*, 2008.

[27] 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.

[28] K. Ligett, S. Kakade, and A. T. Kalai. Playing Games with Approximation Algorithms. In *Proc. STOC*, 2007.

[29] Y. Liu, L. Zhang, and Y. Guan. Sketch-based Streaming PCA Algorithm for Network-wide Traffic Anomaly Detection . In *Proc. ICDCS*, 2010.

[30] M. Kodialam, T. V. Lakshman, and Sudipta Sengupta. Configuring Networks with Content Filtering Nodes with Applications to Network Security. In *Proc. INFOCOM*, 2005.

[31] M. P. Collins and M. K. Reiter. Hit-list Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. In *Proc. RAID*, 2007.

[32] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring Link Weights using End-to-End Measurements. In *Proc. of IMW*, 2002.

[33] 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.

[34] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23–24):2435–2463, 1999.

[35] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4), Dec. 1987.

[36] 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.

[37] 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.

[38] M. R. Sharma and J. W. Byers. Scalable Coordination Techniques for Distributed Network Monitoring. In *Proc. of PAM*, 2005.

[39] 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.

[40] R. Sommer and V. Paxson. Exploiting Independent State for Network Intrusion Detection. In *Proc. ACSAC*, 2005.

[41] 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.

[42] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. of ACM SIGCOMM*, 2002.

[43] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating Network Monitors: Complexity, heuristics and coverage. In *Proc. of IEEE INFOCOM*, 2005.

[44] Symantec Corporation. Deepsight. `http://www.enterprisesecurity.symantec.com`.

[45] J. Ullrich. Dshield.org. `http://www.dshield.org`.

[46] 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.

[47] F. Yu, R. H. Katz, and T. V. Lakshman. Gigabit Rate Packet Pattern-Matching Using TCAM. In *Proc. ICNP*, 2004.

[48] 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.

[49] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the Constancy of Internet Path Properties. In *Proc. IMW*, 2001.

[50] 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.