Network-Wide Deployment of Intrusion Detection and Prevention Systems

Vyas Sekar*, Ravishankar Krishnaswamy†, Anupam Gupta†, Michael K. Reiter††
* Intel Labs, Berkeley † Carnegie Mellon University †† UNC Chapel Hill

ABSTRACT

Traditional efforts for scaling network intrusion detection (NIDS) and intrusion prevention systems (NIPS) have largely focused on a single-vantage-point view. In this paper, we explore an alternative design that exploits spatial, network-wide opportunities for distributing NIDS and NIPS functions. For the NIDS case, we design a linear programming formulation to assign detection responsibilities to nodes while ensuring that no node is overloaded. We describe a prototype NIDS implementation adapted from the Bro system to analyze traffic per these assignments, and demonstrate the advantages that this approach achieves. For NIPS, we show how to maximally leverage specialized hardware (e.g., TCAMs) to reduce the footprint of unwanted traffic on the network. Such hardware constraints make the optimization problem NP-hard, and we provide practical approximation algorithms based on randomized rounding.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*network monitoring, network management*; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Algorithms, Management, Security

Keywords

Intrusion Detection, Network Management

1. INTRODUCTION

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

Increasingly, however, the challenges of scaling this approach are becoming evident. As traffic volumes and the types of analyses grow over time, the NIDS/NIPS placements become a bottleneck. Approaches to scaling single-vantage-point solutions have focused on building NIDS/NIPS clusters (e.g., [38]). The cluster approach, however, faces its own challenges: Since each packet might be relevant to multiple analyses that may occur on different nodes, these solutions need to replicate traffic across the cluster or share the relevant analysis state. These overheads limit the performance of these solutions or force coverage guarantees to be relaxed (e.g., [34]). Furthermore, a single-vantage-point view precludes opportunities for utilizing spare resources at other network locations to offload some responsibilities under high load. These limitations are further exacerbated in the context of ISPs using NIDS/NIPS to provide security services to its customers [3, 4].

In light of these challenges, we explore a different design alternative. 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 analysis. 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 the derived state to the nodes that need it for analysis, we partition the analysis across locations where a packet can already be observed.

There are three key challenges in managing a network-wide deployment of NIDS and NIPS:

- **Resource constraints:** NIDS/NIPS solutions are constrained by the processing and memory capabilities of the underlying hardware. Additionally, some solutions use specialized hardware (e.g., TCAMs [1, 2, 10, 39]) for performance acceleration.
- **Placement affinity:** NIDS/NIPS are not monolithic systems: they consist of modules that analyze different traffic patterns. In particular, the modules may have topo-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM CoNEXT 2010, November 30 – December 3 2010, Philadelphia, USA.
Copyright 2010 ACM 1-4503-0448-1/10/11 ...$10.00.

logical 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 would like to optimally use their existing infrastructure toward their security objectives. For example, in the NIDS case we want to ensure that all traffic is subjected to the appropriate analysis. Similarly, we want to enable NIPS to minimize the network footprint of unwanted traffic.

In the spirit of recent trends in network management [5, 7, 14, 32], we believe these challenges are best addressed by taking a *network-wide coordinated* approach. We outline our specific contributions next.

NIDS: We design a framework for partitioning NIDS functions across a network to ensure that no node is overloaded. This takes into account the resource footprint of each NIDS component, the capabilities of different nodes, and placement constraints for each function (e.g., ingress nodes for scan detection). We implement a network-wide coordinated NIDS using Bro [28]. Our extensions add low memory and processing overhead. In an emulated network-wide deployment scenario our system reduces the maximum processing load by 50% and the maximum memory load by 20%.

NIPS: For NIPS, we show how to maximally reduce unwanted traffic using specialized, but power- and capacity-constrained hardware capabilities (e.g., TCAMs). We show that this optimization problem is NP-hard and design practical approximation schemes based on randomized rounding [29]. Our algorithms provide near-optimal performance on real ISP topologies, achieving ≥92% of the optimal performance. We also demonstrate the promise of using techniques from online learning to combat adversaries trying to evade these defenses [18].

There are several efforts for scaling NIDS and NIPS (e.g., [6, 13, 21, 35, 38]) that focus on building better single-vantage-point solutions. Because our work focuses on the network-wide aspect it effectively complements these advances as it enables administrators to optimally utilize their current hardware infrastructure toward their security objectives.

2. NIDS DEPLOYMENT

We start by describing the constraints and requirements in deploying NIDS functions throughout a network. Next, we set up an optimization framework to assign NIDS responsibilities across a network such that no single node is overloaded. We describe a prototype NIDS implementation and evaluation using Bro [28].

2.1 System Model

Modern NIDS contain diverse modules that perform different types of traffic analyses such as scan detection, analyzing HTTP traffic, tracking IRC traffic, finding malware signatures, etc. We abstract these functions as *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 type of traffic that C_i analyses. For example, if C_i is interested in port-80 traffic, then $\mathcal{T}_i$ specifies all traffic to or from port 80.

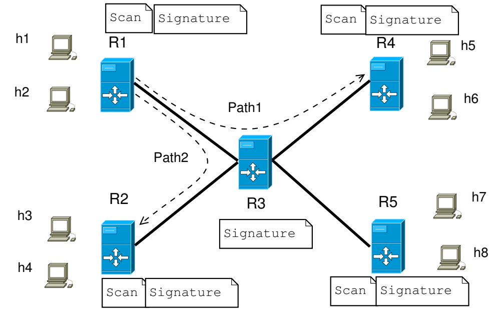


Figure 1: Example of network-wide NIDS configuration

Let $\{\mathcal{T}_{ik}\}_{k=1\ldots}$ denote a partition of $\mathcal{T}_i$ into components such that any packet matching $\mathcal{T}_i$ matches exactly one $\mathcal{T}_{ik}$. We consider classes C_i for which $\mathcal{T}_i$ 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; 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 eligible for performing analysis C_i on traffic matching $\mathcal{T}_{ik}$.

Consider the example network in Figure 1. Suppose class C_1 called `Signature` applies malware signature analysis to traffic $\mathcal{T}_1$. Now, $\mathcal{T}_1$ is partitioned into components $\{\mathcal{T}_{1k}\}_k$ according to the end-to-end path it traverses; e.g., $\mathcal{T}_{11}$ specifies the traffic traversing Path1 and $\mathcal{T}_{12}$ denotes traffic on Path2. Then, $P_{11} = \{\text{R1}, \text{R3}, \text{R4}\}$ is the set of nodes that can observe and recognize traffic matching $\mathcal{T}_{11}$, and $P_{12} = \{\text{R1}, \text{R3}, \text{R2}\}$ is the corresponding set for $\mathcal{T}_{12}$. Similarly, consider the `Scan` module C_2 that checks if any of the hosts h1–h8 exhibit anomalous scanning activity. In this case, the traffic $\mathcal{T}_2$ is partitioned into eight components $\{\mathcal{T}_{2k}\}_{k=1\ldots8}$, corresponding to traffic initiated by each of the eight hosts. Because each host's ingress node is the only node capable of observing all the traffic that a host initiates, the ingresses are ideally suited for scan detection. Thus, we define $P_{21} = P_{22} = \{\text{R1}\}$ (for hosts h1 and h2), $P_{23} = P_{24} = \{\text{R2}\}$, and so forth.

Note that because every node $R_j \in P_{ik}$ can observe all traffic in $\mathcal{T}_{ik}$, it is possible to distribute the work of analyzing traffic in $\mathcal{T}_{ik}$ across them. For example, Figure 1 shows enabling `Signature` on all the nodes on the network; as we will see, we 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, each type of analysis C_i works at some level of traffic aggregation (e.g., sources, destinations, or flows[1]). 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.,

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

flows), that match $\mathcal{T}_i$ and $\mathcal{T}_{ik}$, respectively. We assume that the classes C_i and traffic sets $\mathcal{T}_{ik}$ are defined so that the cost that node $R_j \in P_{ik}$ incurs in performing C_i-analysis on an aggregate in $\mathcal{T}_{ik}$ is independent of where other aggregates in $\mathcal{T}_{ik}$ are analyzed and, in particular, does not require R_j to communicate with nodes analyzing other aggregates in $\mathcal{T}_{ik}$.

2.2 Problem Formulation

We envision that a centralized operations center periodically configures the NIDS responsibilities of the different nodes to achieve the network's security objective. For example, ISPs typically collect traffic reports (e.g., NetFlow, SNMP) every few minutes, and since NIDS configurations would typically be driven from such reports, we envision needing to reconfigure NIDS with roughly the same frequency.

Objective: Our goal is to guarantee complete *coverage*; a network-wide deployment should be logically equivalent to running a single NIDS on the entire traffic. In doing so, we want to ensure that the processing/memory load is balanced (for a suitable balancing function).

Control Variables: Let d_{ikj} denote the fraction of traffic in C_i on coordination unit P_{ik} that node R_j processes. We consider a fractional split to provide more fine-grained opportunities for distributing the load across nodes.

Inputs:

- The 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}.
- For each C_i, the processing load (CPU seconds per packet) is $CpuReq_i$ and the memory load $MemReq_i$ (e.g., bytes per flow or per source).
- The CPU and memory capacity $CpuCap_j$ and $MemCap_j$ of each node R_j. We consider a general model where network elements have heterogeneous capabilities.

Note that these inputs are already available or can be inferred from existing measurements. Network operations centers typically know the traffic matrix, routing policy, and node hardware configurations [12]. Similarly, the resource footprints of the NIDS modules can be obtained from offline profiles [16].

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

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

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

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

$$\forall j,\ CpuLoad \geq CpuLoad_j \quad (4)$$

$$\forall j,\ MemLoad \geq MemLoad_j \quad (5)$$

$$\forall i, \forall k, \forall j,\ 0 \leq d_{ikj} \leq 1 \quad (6)$$

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

Eq (1) says that 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 [16]. 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 in terms of the total volume of packets of each class that the node is assigned [16]. Finally, we model the maximum memory and processing load across all the nodes, and minimize the max of these two measures.

Output: We solve the linear program to generate *sampling manifests* that specify the NIDS responsibilities 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 to map these to non-overlapping hash ranges to ensure that each node $R_j \in P_{ik}$ analyzes a distinct subset of the $\mathcal{T}_{ik}$ traffic.

Given a sampling manifest, the algorithm on a node R_j is shown in Figure 3. As each packet arrives, we find the set of NIDS modules that need to analyze this packet. (In general, the same packet may be analyzed by several modules; e.g., a packet on port 80 may be analyzed by the HTTP, malware signature, and scan detection modules.) For each such module, we check if R_j should analyze this packet. To do so, we compute a HASH from the packet header using a lightweight hash function. 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. The hash may be computed over different fields in the packet header depending on the analysis. For example, for flow-based analysis, the hash is over the unidirectional 5-tuple. For session-based analysis, the hash is over a bidirectional 5-tuple such that the src/dst IP are consistent in both directions.

2.3 Implementation in Bro

We implement the above coordination functions in Bro [28]. Bro is logically divided into two parts (Figure 4): (1) an *event engine* that converts a stream of packets into high-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 which is generated in the event engine and carried into the policy engine. The 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 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 studies [26]. As an optimization, we add a check in the basic connection processing step to avoid creating session state for traffic that falls outside the sampling manifest for this Bro instance. (This occurs before running the per-module checks in Figure 3.)

We consider two implementation alternatives for the sampling checks in Figure 3 (i.e., line 5 for each i and k): (1) delaying the sampling checks 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 implements the coordination functions as site-specific extensions in the policy engine as intended in the Bro system design. This may, however, impose high overhead for some modules (Section 2.4). This is because the policy scripts are executed by an interpreter and doing hash lookups/checks is quite expensive. In (2) we run the sampling checks earlier 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 in the event engine, namely where application-protocol modules (e.g., HTTP, IRC) and the signature-based detection module are initialized.

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.

We extend Bro to implement the functions to process the sampling manifests and associated configuration files. We assume that our system has access to configuration files that map each packet matching $\mathcal{T}_{ik}$ to the corresponding P_{ik}. Specifically, these map IP prefixes to their ingress locations and provide the routing paths for a given pair of IP prefixes.

```
GENERATENIDSMANIFEST(d* = ⟨d*_ikj⟩)
1  foreach class C_i do
2     foreach coordination unit P_ik do
3        Range ← 0
         // the order of nodes does not matter
4        foreach j, R_j ∈ P_ik do
5           HashRange(i, k, j) ← [Range, Range + d*_ikj]
6           Range ← Range + d*_ikj
   // Assignments across Classes and Coordination units
7  ∀j, Manifest(R_j) ← {⟨{i, k}, HashRange(i, k, j)⟩}
```

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

```
COORDINATEDNIDS(pkt, R_j, Manifest(R_j))
1  {C_i}_i ← GETCLASS(pkt)
   // Each packet may be analyzed by multiple modules
2  foreach class C_i do
3     k ← GETCOORDUNIT(pkt, i)
      // HASH returns a value in [0, 1]
      // Specific packet fields used for HASH
      // depend on semantics of C_i
4     h_pkt ← HASH(pkt, i)
5     if h_pkt ∈ HashRange(i, k, j) then
6        Run class C_i for pkt
```

Figure 3: Coordinated NIDS algorithm on node R_j

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 prototype. Finally, we describe a network-wide evaluation that shows the benefits of our coordinated network-wide approach vs. a single vantage point approach.

Setup: We use a custom traffic generator that takes 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). We use *template sessions* using real traffic captured for common protocols like HTTP, IRC, and Telnet, and synthetically generate 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 single-vantage-point approach. By design, the network-wide approach provides 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.)

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.

Optimization time: We use CPLEX to solve the linear program. In order to be responsive to traffic dynamics, we may need to rerun the solver periodically to adapt to traffic changes, e.g., every few minutes. It takes 0.42 seconds to compute the optimal solution for a 50-node topology. This suggests that the optimization step will not be a bottleneck.

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 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 imple-

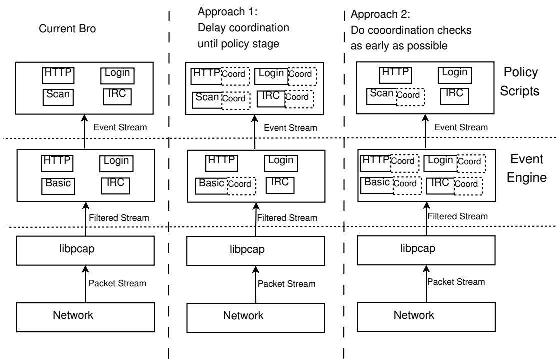


Figure 4: Implementing the coordination functionality in Bro. The "coord" boxes indicate where we add in coordination checks. For some modules (e.g., `Scan`), the checks have to be in the policy engine.

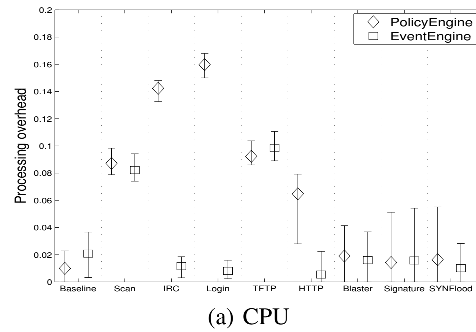
mented in the event engine wherever possible, and Bro with all coordination checks in the policy scripts. The sampling manifests in both cases are configured to specify that this standalone node needs to process all the traffic. We configure Bro to run each analysis module in isolation.

Our goal is to evaluate: (a) the processing overhead induced by the coordination functions — finding the coordination unit identifier for each packet, computing the hashes of various connection fields, and checking if the hash lies in the sampling ranges; and (b) the memory overhead of adding the hash values into the connection record. For each configuration, we perform 5 runs and report the mean, minimum, and maximum overhead.

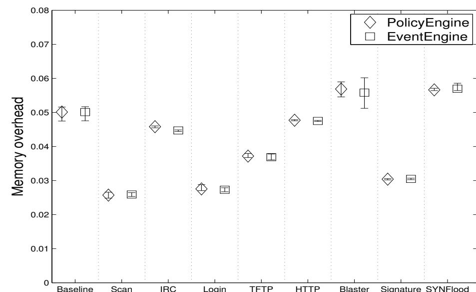
Figure 5(a) compares the processing overhead of our extended Bro implementations vs. the unmodified Bro system (using the total CPU time used) for different modules. For the Baseline, Signature, Blaster, and SYN-flood modules, the overhead of coordination checks is around 2% on average for both implementations. 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 coordinated versions have very similar overhead because the coordination checks occur in the same place; either they cannot be offloaded to the event engine (e.g., scan, TFTP) or they occur solely in the event engine (e.g., Signature). However, in the case of HTTP, IRC, and Login, there is significant overhead when we perform the coordination checks in the policy engine.

Figure 5(b) shows that the memory overhead of the coordinated versions is at most 6%. Recall that this overhead arises because we augment the connection record to carry hashes of different combinations of header fields.

Based on these observations, for a network-wide NIDS we implement the coordination checks for the different modules as early as possible. That is, for modules such as HTTP, IRC, and Login, we add the checks in the event engine, and for modules such as Scan, TFTP, Blaster, and SYNFlood we add the checks in the policy scripts.



(a) CPU

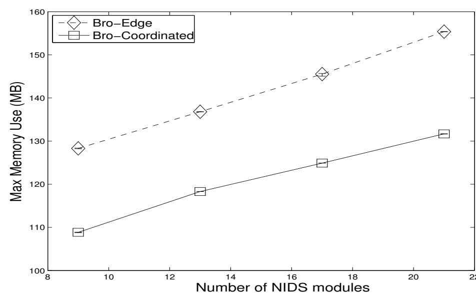


(b) Memory

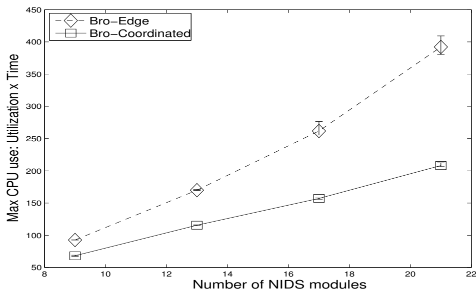
Figure 5: CPU and memory overhead for different modules with the coordination-enabled Bro prototype.

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 [30]. We use shortest-path routing based on link distances to determine the paths between each pair of locations. Our setup configures all locations to have the same processing/memory capabilities. Given this topology and traffic information, we solve the linear program and assign NIDS responsibilities to minimize the maximum CPU/memory load. We use the guidelines of Dreger et al. [16] to obtain the CPU and memory 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 each trace, we run Bro on the trace in pseudo-realtime emulation mode. During each run, we profile 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.



(a) Memory



(b) CPU

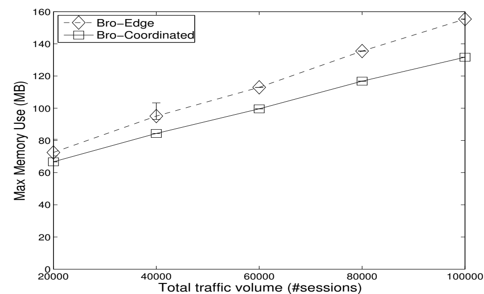
Figure 6: Maximum per-node memory and CPU usage across the network as number of NIDS modules grows. The total traffic volume is 100,000 sessions.

First, we consider the effect of adding more NIDS functionality. For this experiment, we keep the traffic volume fixed at 100,000 sessions, but add more NIDS modules. In order to emulate adding more NIDS functions, we start with the set of modules shown in Figure 5 and create duplicate instances of HTTP, IRC, Login, and TFTP modules.[2] Recall that there were two types of modules: those where we could move the coordination functions into the event engine and others where we could not. We inspected 140 policy scripts in the Bro 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 augmented with more modules in practice. Figures 6(a) and 6(b) show that the coordinated approach scales better as we add more functionality into the NIDS deployment.

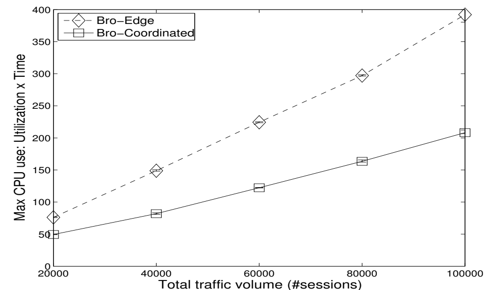
Next, we vary the total number of end-to-end sessions retaining the same traffic matrix structure and using all 21 NIDS modules from Figure 6. Figures 7(a) and 7(b) show the maximum per-node processing and memory load across the network as a function of the total traffic volume. We see that coordination reduces the maximum memory footprint by 20% and the maximum CPU footprint by 50%. The overall trend also shows that the network-wide approach scales better as the workload increases.

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 8(a) and 8(b). We show the result for the configuration corre-

[2]We used fake instances for convenience to avoid having to benchmark and modify scripts for other modules.



(a) Memory



(b) CPU

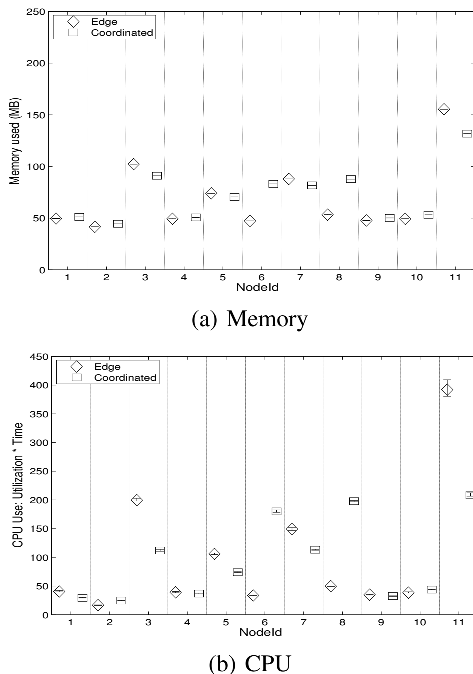
Figure 7: Maximum per-node memory and CPU usage across the network as the total traffic volume increases. The NIDS includes 21 modules.

sponding to 100,000 sessions and 21 NIDS modules from the previous results. 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.) The coordinated deployment can offload NIDS responsibilities that were previously assigned to node 11 to other nodes where the same analysis could have been performed. The figure shows that some nodes (e.g., nodes 6 and 8) perform more NIDS processing than in the edge-only setting.

2.5 Extensions

More fine-grained coordination capabilities: These results show that our 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 Bro is a connection: an end-to-end session between two hosts. While this provides a general abstraction to implement many NIDS modules, some modules may need only a subset of that information. For example, `Scan` needs to observe only the first packet in a connection to track the number of distinct destination IPs that a source contacts. In our setup, because the ingresses need to run the `Scan` module, they need to track all connections. This means that even though we can offload module-specific processing, e.g., from node 11 to other nodes in Figure 8, node 11 still needs to track all packets because a connection is the smallest granularity of processing. Consequently, we duplicate the baseline connection processing and tracking work across the network.



(a) Memory

(b) CPU

Figure 8: Memory and CPU load on each network node for 100,000 sessions and 21 NIDS modules.

One direction of future work is to design NIDS that inherently support fine-grained coordination capabilities – allowing different granularities of connection information, providing interfaces for modules to subscribe to more fine-grained events (e.g., first packet of a flow for `Scan`), and allowing modules to specify how early they can implement coordination checks. Such capabilities would provide more opportunities for distributing the load across a network.

Redundancy for reliability: In order to be robust to NIDS failures, administrators may want enable each analysis module at r 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 at all locations, we need other mechanisms to protect against implementation bugs.)

Extending our model from Section 2.1, this means that we have to divide the hash space for each coordination unit such that: (1) each point in the space is covered r times and (2) no node is responsible for the same point more than once. The second clause ensures that we have r *distinct* nodes to analyze each packet/connection.

One approach is to incorporate the notion of a "redundancy level"; we can generalize the control variables d_{ikj}s to new variables d_{ikjl}s that also indicate the redundancy level each l corresponds to. However, it is intuitively hard to capture requirement (2) that the same node is never responsible for the same point in the space more than once.

Fortunately, there is a simple extension to the LP formulation to provide redundant coverage. The key is not to treat replicated coverage in terms of levels, but as covering a larger hash space. That is, instead of thinking of the problem in terms of covering the space $[0, 1]$ r times, we think of it as covering the space $[0, r]$, while retaining the bound $d_{ikj} \leq 1$. We modify the RHS of Eq (1) to r instead of 1 and solve this new LP. While converting the LP solution into sampling manifests (Figure 2), we proceed as before, except that we wraparound the range every time it exceeds 1.

3. NIPS DEPLOYMENT

In this section, we describe our model to capture the constraints in deploying NIPS functions. We formulate the optimization problem and develop an approximation algorithm based on randomized rounding because it is NP-hard to solve the problem exactly. We evaluate our algorithm on a range of real network topologies and system parameters. Finally, we describe how we can extend the model to be robust to dynamic adversaries using techniques from online algorithms.

3.1 System Model

NIPS typically consist of *filtering rules* matching specific traffic patterns. For example, firewall rules look at the packet header fields; signature-based filters detect 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.

In order to operate at line rates, modern NIPS use special purpose hardware such as Ternary CAMs (TCAM) for pattern matching (e.g., [39, 40]). However, such hardware capabilities are expensive and power-intensive. Thus, there are natural budget and technological limits on how many NIPS rules can be active on each node. To address this concern, we extend the model from the previous section to capture the use of such special hardware capabilities.

3.2 Problem Formulation

The objective is to configure the NIPS modules to minimize the network footprint of unwanted traffic or equivalently to maximize the reduction in the total footprint by dropping 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. $Dist_{ikj}$ 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 use router hops, $Dist_{i11} = 3$, $Dist_{i12} = 2$,

and $Dist_{i13} = 1$. Alternatively, to model the total volume of unwanted traffic dropped, we set all $Dist_{ikj}$ 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) and the TCAM required $CamReq_i$ per rule. Note that the $CamReq$ is *per-rule* rather than per-packet or per-flow.
- The capacity $CpuCap_j$, $MemCap_j$, 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, M_{ik} denotes the fraction of traffic along path P_{ik} that *matches* and will be affected by this rule. For example, if the rule C_i detects a specific malware signature, M_{ik} is the fraction of this malware traffic on the path P_{ik}. These can be estimated from NIDS alerts or other sources (e.g., NetFlow feeds).

$$\text{Max.} \sum_i \sum_k \sum_{j, R_j \in P_{ik}} T_{ik}^{items} \times M_{ik} \times Dist_{ikj} \times d_{ikj} \quad (7)$$

subject to

$$\forall j, \sum_i CamReq_i \times e_{ij} \leq CamCap_j \quad (8)$$

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

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

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

$$\forall j, \forall i, \forall k, d_{ikj} \leq e_{ij} \quad (12)$$

$$\forall k, \forall i, \forall j, \; d_{ikj} \geq 0 \quad (13)$$

$$\forall i, \forall j, \; e_{ij} \in \{0, 1\} \quad (14)$$

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.

Given this setup, we can formulate the NIPS deployment problem using a Mixed Integer-Linear Program. 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 $T_{ik}^{items} \times M_{ik}$. Each node R_j that lies on P_{ik} contributes $Dist_{ikj} \times d_{ikj}$ toward reducing the total footprint. Since we split the sampling responsibilities across the R_js 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 TCAM hardware on each node. Eq (9) and Eq (10) model the aggregate memory and processing load on each node. Eq (12) checks that a node cannot apply a rule C_i unless it has been enabled and Eq (11) ensures that the fraction of the traffic sampled on each path-rule combination is never more than 1.

There are three implicit assumptions in this formulation. First, we assume that attackers cannot craft traffic that can avoid the sampling checks. That is, both legitimate and unwanted traffic patterns are distributed uniformly through the hash space. In practice, administrators can use private keyed hash functions to prevent adversaries from evading the hash checks. Second, to rigorously model the load on each node, we should take into account the traffic that has already been dropped upstream. In that case, Eq (9) and Eq (10) become non-linear constraints. Specifically, the LHS of these equations will have an extra factor $(1 - \sum_{j'} d_{ikj'})$, where the sum is taken over routers $R_{j'}$ preceding R_j on path P_{ik}, to model the traffic that has already been dropped. We conservatively model the load in terms of the total volume (before any drops). Third, we assume that the rules are non-redundant and the same packet/flow does not match multiple rules. Our high-level goal is to obtain general guidelines for configuring the NIPS modules. To this end, these are reasonable assumptions.

The presence of the discrete e_{ij} variables in Eq (14) makes the optimization problem NP-hard. We can show that this problem is NP-hard via a reduction from the MAX-CUT problem. (Due to space constraints, we refer the reader to our technical report for the proof of NP-hardness [31].)

3.3 Approximation via randomized rounding

Given that it is NP-hard to solve the optimization problem exactly, we use an approximation algorithm using randomized rounding [29]. Figure 9 describes the steps involved in our algorithm.

First, we *relax* the problem by replacing the discrete e_{ij}s by continuous variables in the interval $[0, 1]$ to create a linear program. Then, starting from the solution to this linear program, we generate a feasible solution to the original problem for which the objective function is close to optimal.

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 probability e_{ij}^*, and 0 otherwise. 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 9). 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.)

RANDOMIZEDROUNDING

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

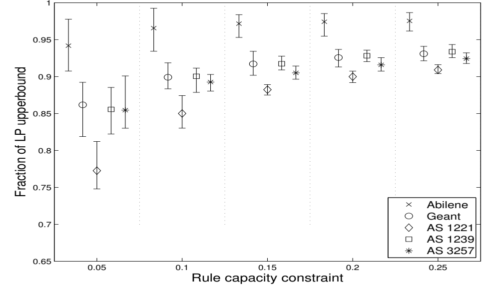
Figure 9: Approximation algorithm for the NIPS deployment problem via randomized rounding.

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 can prove that the algorithm in Figure 9 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. (Due to space constraints, we omit the proof and refer readers to our technical report [31].) Since $\mathsf{Opt}_{LP} \geq \mathsf{Opt}_{NIPS}$, this guarantees that the value of our solution is at least $\frac{\mathsf{Opt}_{NIPS}}{O(\log N)}$.

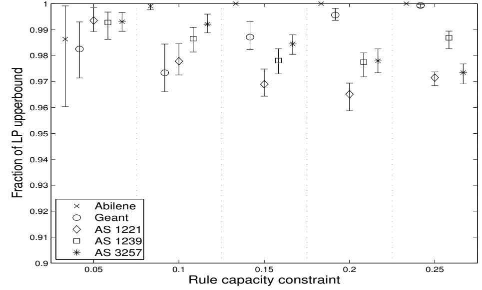
The algorithm in Figure 9 can be improved in two ways. First, the scaling of $\widehat{d_{ikj}}$ (line 11) is likely to be too conservative. A practical alternative is to solve the LP represented by Eqs (9)–(14) after setting the values for $\widehat{e_{ij}}$ obtained in line 5 to be constants, and use the values for $\{\widehat{d_{ikj}}\}_{ikj}$ returned by this solution. Second, we may be conservative in setting some $\widehat{e_{ij}}$s 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 (8), and then solve the LP treating these $\widehat{e_{ij}}$s as constants. These steps do not affect feasibility and can only improve the value of the objective function, and the approximation guarantee holds on these extensions as well.

3.4 Evaluation

For this evaluation, we use network topologies from educational backbones (Internet2 and Geant) and tier-1 ISP topologies inferred by Rocketfuel [36]. We construct ingress-egress paths for each pair of nodes using shortest-path routing [25]. We use a gravity model traffic matrix based on city



(a) Rounding + LP solve



(b) Rounding + Greedy + LP solve

Figure 10: Performance of the approximation algorithms with a uniform rule match rate distribution. There are 100 NIPS rules. Each node can enable 100$\times$ "rule capacity constraint" of these rules.

populations [33]. 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 has a $MemCap_j$ of 400,000 flows and a $CpuCap_j$ of 2 million packets that it can process in this 5-minute interval. We use $Dist_{ikj}$ 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., for all i, $CamReq_i = CpuReq_i = MemReq_i = 1$. We present results for the case when M_{ik} values are distributed uniformly in the range $[0, 0.01]$. For the following results, we vary the $CamCap_j$ of each node as a fraction of the total number of NIPS rules; this fraction is called the "rule capacity constraint". For each setting, we generate 30 different M_{ik} values. Then, we run 10 iterations of the rounding-based algorithms and take the best solution across these 10 runs.

Optimization time: As in the NIDS case, we may need to rerun the algorithm as traffic profiles change. It takes roughly 220 seconds to run our algorithm for a 50-node topology. Given that we expect to periodically recompute the solution every few minutes, this is a reasonable cost. Most of the time is spent in solving the LP in Step 2 and the final step where we solve a second LP. We can further reduce this time by *seeding* the LP solver with starting solutions from previous computations.

Optimality gap: As discussed in Section 3.3, the basic algorithm in Figure 9 is conservative and can be improved in two ways: (1) Solving an LP after the rounding stage instead of scaling down the $\widehat{d_{ikj}}$s and (2) greedily setting more $\widehat{e_{ij}}$s to 1 after the rounding stage without violating the TCAM constraints and then solving the LP as in (1).

Figure 10 presents the mean, minimum, and maximum value obtained by these two rounding-based algorithms across the 30 M_{ik} scenarios as a fraction of Opt_{LP}.[3] First, we see that the performance our algorithms is much better than the approximation ratio of $\frac{1}{O(\log N)}$ as we get more than 70% of Opt_{LP}. Second, the greedy step can significantly boost the performance, achieving more than 92% of Opt_{LP}. These results are consistent across the different topologies and across the values of the $CamCap_j$ constraint. These results hold for other M_{ik} distributions as well (not shown for brevity).

3.5 Online Adaptation

The above formulation considers a static scenario where the match rates $\{M_{ik}\}$s are known and fixed. However, an adversary can control the sources and nature of the unwanted traffic. For example, an attacker who controls a botnet can modify the attack profile – the sources and destinations of the malicious traffic and the attack mix. Our goal is to make NIPS deployment 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 [18] 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 space of possible 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 S_t is revealed only after the decision for the tth step O_t has been made; we do not have access to the current S_t before making the decision O_t.

Next, we describe how to use 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. That is, we remove the discrete e_{ij} variables and the associated constraints Eqs (8), (12), and (14) from the formulation in Section 3.2.

To model the time-varying 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 $T_{ik}^{items} \times M_{ik} \times Dist_{ikj}$. 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., $d_{ikj} \times \left(T_{ik}^{items} \times M_{ik} \times Dist_{ikj}\right)$.[4] An adversary can change the different M_{ik} values over time to vary the traffic mix. Our goal is to adapt the NIPS deployment without knowing the exact M_{ik} values in each epoch.

The goal is to have a total cost over γ epochs, $\sum_{t=1}^{\gamma} O_t.S_t$, that is close to $mincost_\gamma = \min_{O \in \mathcal{O}} \sum_{t=1}^{\gamma} O.S_t$. That is, we want our cost to be comparable to the cost of the best possible single solution in hindsight.[5] The *regret* is defined as $\sum_{t=1}^{\gamma} O_t.S_t - mincost_\gamma$; the difference between the costs incurred by the online decision procedure and this single best decision chosen in hindsight.

Kalai and Vempala [18] 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 $O_t = \Lambda(\sum_{q=1}^{t-1} S_q + p_t)$.

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 distance 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 $\epsilon = \sqrt{\frac{D}{RA\gamma}}$ gives,

THEOREM 3.1. $\frac{E[cost(FPL(\epsilon)) - mincost_\gamma]}{\gamma} \leq \sqrt{\frac{DRA}{\gamma}}$ *[18].*

That is, the average regret goes to zero as γ 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

[3] 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.

[4] 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.

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

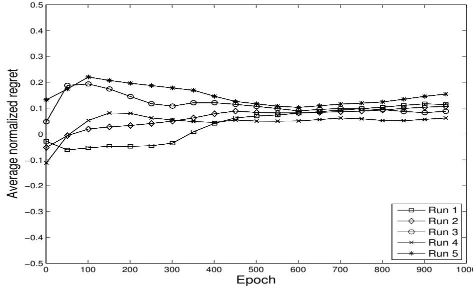


Figure 11: Result showing the normalized regret over time for different runs of the online adaptation algorithm. We normalize the regret by the objective value of the best static solution.

$M_{ik} = \frac{\sum_{q=1}^{t-1} M_{ik}^{Obs}(q)}{t-1} + \frac{p_t}{t \times T_{ik}^{items}}$, where p_t is computed as described in the FPL procedure and where $M_{ik}^{Obs}(q)$ is the observed fraction of traffic on path P_{ik} in epoch q that matched C_i. (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 3.3 (without the rule capacity constraints). We consider a dynamic setting where the M_{ik}s are drawn from a uniform match rate distribution, but are revealed only at the end of each epoch.

The metric in which we are interested is the average normalized regret as function of time: $\frac{\sum_{t=1}^{\gamma} Obj_t^{staticopt} - Obj_t^{FPL}}{\sum_{t=1}^{\gamma} Obj_t^{staticopt}}$, 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 11 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. Two directions of future work are to evaluate the performance of this online algorithm in the presence of strategic adversaries and to apply this framework to the formulation from Section 3.2.[6]

4. RELATED WORK

Network management and monitoring: Many recent proposals have argued the benefits of a coordinated approach for network management [5, 7, 12, 14, 42]. Hash-based packet selection to coordinate monitoring responsibilities has been used in the context of Trajectory Sampling [9] and cSamp [32]. We build on this prior work. There have been efforts for automated configuration management for enterprise networks to satisfy security and reachability policies [27]. However, NIDS/NIPS deployment present unique constraints that we address in this paper.

Monitor placement: Several efforts have studied the problem of placing network monitors to cover all routing paths using as few monitors as possible [8, 37]. These show that the problems are NP-hard and propose greedy algorithms. Kodialam et al. [24] 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 aspects. First, we model the problem as one of enabling modules with different sampling rates subject to resource constraints. Second, we operate within a given routing framework and do not modify routing policies.

Scaling NIDS/NIPS: There are many proposals for scaling NIDS/NIPS using parallelization (e.g., [6,13,21,35,38]), hardware-assisted acceleration (e.g., [17]), better algorithms (e.g., [20]), models for understanding their resource consumption (e.g., [15,16]), and optimizing rule patterns (e.g, [1, 2,10,39]). Our work effectively complements these because we exploit *spatial* opportunities for distributing NIDS and NIPS functions across a network.

5. DISCUSSION

Traffic changes: Sections 2 and 3 formulate the problem in a static setting. This raises concerns regarding traffic bursts, changing traffic profiles, etc. Since our optimization modules interface with other network management tools (e.g., NetFlow), we can periodically rerun the optimizations to adapt to long-term (e.g., timescales of tens of minutes) changes. Our experiments show that the optimization procedures are responsive enough to handle such recomputations. To handle short-term bursts, we can use conservative values; e.g., 95%ile values to account for bursty patterns and tradeoff some loss in optimality for better robustness.

Routing changes: Network paths are largely stable on the timescales for per-session analysis [41]. However, when route changes do occur and we recompute the optimal solutions, there is a concern that this may affect correctness. Specifically, the new optimal solution may be such that a node maintaining some specific connection state is no longer responsible for monitoring that connection.

To ensure correctness in the presence of such dynamics, we can ensure that nodes temporarily retain the old responsibilities until existing connections in these assignments expire. That is, each node picks up new assignments immediately but takes on no new connections in the old assignments. This may result in some duplication, but provides correct operation. However, it may be the case that new packets for connections in the old assignment no longer traverse the 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 [34]. Adding redundant functionality as out-

[6]There are known extensions for the case where Λ is an approximation algorithm [18,23].

lined in Section 2.5 can further reduce the impact of routing changes.

Provisioning and Upgrades: We can also 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., [38]) or augment existing deployments with more powerful hardware (e.g., [17]).

Aggregated analysis: Certain kinds of analysis need aggregated network-wide views (e.g., [11, 19, 22]). We believe that our models can be extended to such scenarios as well, e.g., by explicitly incorporating communication costs between NIDS instances.

6. CONCLUSIONS

In this paper, 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 policies, and the resources available at each location.

Our models and algorithms enable 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.

Acknowledgments

This work was supported in part by the National Science Foundation under grants CNS-0756998, CNS-0831245, and CCF-1016799, grant N000141010155 from the Office of Naval Research, and the Alfred P. Sloan fellowship of A. Gupta.

7. REFERENCES

[1] 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.

[2] 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.

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

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

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

[6] 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.

[7] 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. NSDI*, 2005.

[8] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the Monitor Placement problem: Optimal Network-Wide Sampling. In *Proc. CoNeXT*, 2006.

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

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

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

[12] 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. ACM SIGCOMM*, 2000.

[13] 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.

[14] 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.

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

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

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

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

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

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

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

[22] 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. IEEE INFOCOM*, 2006.

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

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

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

[26] M. Molina, S. Niccolini, and N. Duffield. A Comparative Experimental Study of Hash Functions Applied to Packet Sampling. In *Proc. International Teletraffic Congress (ITC)*, 2005.

[27] S. Narain, G. Levin, S. Malik, and V. Kaul. Declarative infrastructure configuration synthesis and debugging. *Journal of Network and Systems Management*, 16(3), 2008.

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

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

[30] M. Roughan, M. Thorup, and Y. Zhang. Performance of estimated traffic matrices in traffic engineering. In *SIGMETRICS*, 2003.

[31] V. Sekar, R. Krishnaswamy, A. Gupta, and M. K. Reiter. Network-Wide Deployment of Intrusion Detection and Prevention Systems. Technical Report, CMU-CS-10-124, Comp. Sci. Dept., CMU, 2010.

[32] 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. NSDI*, 2008.

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

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

[35] 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.

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

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

[38] 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. RAID*, 2007.

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

[40] 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.

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

[42] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast Accurate Computation of Large-scale IP Traffic Matrices from Link Loads. In *Proc. ACM SIGMETRICS*, 2003.