New Opportunities for Load Balancing in Network-Wide Intrusion Detection Systems

Victor Heorhiadi
UNC Chapel Hill
victor@cs.unc.edu

Michael K. Reiter
UNC Chapel Hill
reiter@cs.unc.edu

Vyas Sekar
Stony Brook University
vyas@cs.stonybrook.edu

ABSTRACT

As traffic volumes and the types of analysis grow, network intrusion detection systems (NIDS) face a continuous scaling challenge. Management realities, however, limit NIDS hardware upgrades to occur typically once every 3-5 years. Given that traffic patterns can change dramatically, this leaves a significant scaling challenge in the interim. This motivates the need for practical solutions that can help administrators better utilize and augment their existing NIDS infrastructure. To this end, we design a general architecture for network-wide NIDS deployment that leverages three scaling opportunities: *on-path* distribution to split responsibilities, *replicating* traffic to NIDS clusters, and *aggregating* intermediate results to split expensive NIDS processing. The challenge here is to balance both the compute load across the network and the total communication cost incurred via replication and aggregation. We implement a backwards-compatible mechanism to enable existing NIDS infrastructure to leverage these benefits. Using emulated and trace-driven evaluations on several real-world network topologies, we show that our proposal can substantially reduce the maximum computation load, provide better resilience under traffic variability, and offer improved detection coverage.

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 systems play a critical role in keeping network infrastructures safe from attacks. The driving forces for increased deployment include regulatory and policy requirements, new application traffic patterns (e.g., cloud, mobile devices), and the growing complexity of attacks [22,41]. In conjunction with these forces, the rapid growth in traffic volumes means that NIDS deployments face a continuous scaling challenge to keep up with the increasing complexity of processing and volume of traffic.

The traditional response in the NIDS community to address this scaling challenge has been along three dimensions: better algorithms (e.g., [33]); specialized hardware capabilities such as TCAMs (e.g., [42]), FPGAs (e.g., [17]), and graphics processors (e.g., [38]); and parallelism through the use of multi-core or cluster-based solutions (e.g., [37, 39]). These have been invaluable in advancing the state-of-the-art in NIDS system design. However, there is a significant delay before these advances are incorporated into production systems. Furthermore, budget constraints and management challenges mean that network administrators upgrade their NIDS infrastructure over a 3-5 year cycle [32]. Even though administrators try to provision the hardware to account for projected growth, disruptive and unforeseen patterns can increase traffic volumes. Thus, it is critical to complement the existing research in building better NIDS systems with more immediately deployable solutions.

Past work has shown that distributing responsibilities across intrusion detection systems on an end-to-end path can offer significant benefits for monitoring applications [6, 29]. This provides a way for administrators to handle higher traffic loads with their *existing* NIDS deployment without requiring a forklift upgrade to deploy new NIDS hardware. Our premise is that these past proposals for distributing monitoring functions do not push the envelope far enough. Consequently, this not only restricts the scaling opportunities, but also constrains the detection capabilities that a network-wide deployment can provide on three key dimensions:

- First, these focus on strictly *on-path* distribution. While such on-path processing is viable (e.g., [7]), there is an equally compelling trend toward consolidating computing resources using datacenter deployments within and outside enterprise networks. These offer natural management and multiplexing benefits that are ideally suited to the compute-intensive and dynamic nature of NIDS workloads. Other, concurrent work in research and in industry is motivated by similar benefits of consolidated cloud deployments for NIDS-like processing, as well [1, 2, 11, 32].
- Second, this past work assumes that the NIDS analysis occurring at a network node is *self-contained*. That is, the NIDS nodes act as standalone entities and provide equivalent monitoring capabilities without needing to interact with other nodes. This restriction leads to certain types of aggregated analysis being topologically constrained. For example, in the case of scan detection, all traffic needs to be processed at the ingress gateway for each host [29].
- Third, prior work implicitly assumes that each NIDS node can al-

ways provide processing capabilities that are *semantically equivalent* to running the analysis at manually created chokepoints. Unfortunately, practical networking realities (e.g., asymmetric routing) may often violate such requirements; i.e., the forward and reverse flows in an end-to-end session may not always be observed at the same location for stateful NIDS processing.

Our vision is a *general* NIDS architecture that goes beyond on-path distribution to allow new scaling opportunities via traffic *replication* and analysis *aggregation*. *Replication* enables us to offload processing to lightly loaded nodes that might be off-path and accommodate trends for building consolidated compute clusters. Furthermore, replication enables new detection functionality that would have been previously infeasible. For example, our framework enables stateful NIDS analysis even when the two flows in a session do not traverse a common node. *Aggregation* allows us to split an expensive NIDS task into smaller subtasks that can then be combined to provide equivalent analysis capabilities. This enables more fine-grained scaling opportunities for NIDS analyses that would otherwise be topologically constrained (e.g., scan detection).

A key constraint here is to enable these opportunities with minimal communication costs. Thus, our goal is to assign processing responsibilities to balance the tradeoff between the communication cost imposed by replication and aggregation vs. the reduction in computation load. Our specific focus in this paper is on passive monitoring devices such as NIDS and as such replication or aggregation do not affect the latency perceived by end user applications. We envision a network-wide management module that assigns processing, aggregation, and replication responsibilities across the network [12, 19]. To systematically capture these tradeoffs, we design formal linear programming (LP) based optimization frameworks. In order to execute these management decisions without requiring modifications to existing NIDS implementations, we interpose a lightweight *shim* layer that runs on each NIDS node (or at an upstream router to which the NIDS is attached).

We evaluate our architecture and implementation using a combination of "live" emulation on Emulab [40] and trace-driven simulations on a range of real-world topologies. Our results show that a replication-enabled NIDS architecture can reduce the maximum computation load by up to 10×; is significantly more robust to variability in traffic patterns by reducing the peak load more than 20×; and can lower the detection miss rate from 90% to zero in some scenarios where routes may not be symmetric. These benefits are achieved with little overhead: computing the analysis and replication responsibilities takes < 1.6 seconds with off-the-shelf LP solvers, and our shim layer imposes very low overhead.

Contributions and Roadmap: Our contributions are:

- Identifying new replication and aggregation opportunities for NIDS scaling (Section 2).
- Formal models for balancing compute-communication tradeoffs in a general NIDS architecture that subsumes existing on-path distribution models (Section 4, Section 5, Section 6).
- A backwards-compatible architecture (Section 3) and implementation (Section 7) to realize these benefits.
- Extensive evaluation of the potential benefits over a range of real-world network topologies (Section 8).

We discuss outstanding issues in Section 9 and related work in Section 10, before concluding in Section 11.

2. MOTIVATION

In this section, we begin by describing the prior work for *on-path* distribution [29]. Then, we discuss three motivating scenarios that argue for a general NIDS architecture that can incorporate traffic *replication* and analysis *aggregation*.

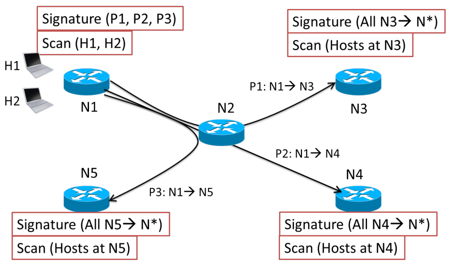

Figure 1: NIDS deployments today are single-vantage-point solutions where the ingress gateway is responsible for monitoring all traffic.

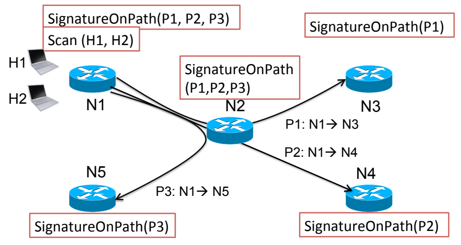

Figure 2: NIDS with on-path distribution [29]: Any node on the path can run `Signature` detection; `Scan` detection cannot be distributed.

2.1 On-path distribution

Suppose there are two types of NIDS analysis: `Signature` for detecting malicious payloads and `Scan` for flagging hosts that contact many destination addresses. Figure 1 shows how today's NIDS deployments operate, wherein all types of analysis occur only at the gateway node. That is, node N1 runs `Scan` and `Signature` detection for all traffic to/from hosts H1–H2 on paths P1–P3; the other nodes run the corresponding analysis for the hosts for which they are the gateway nodes. A natural limitation with this architecture is that if the load exceeds the provisioned capacity on a node, then that node has to either drop some functionality (e.g., disable expensive modules) or drop packets.

A natural solution to this problem is to exploit spare resources elsewhere in the network. For example, nodes N2–N5 may have some spare compute capacity when N1 is overloaded. Prior work shows an architecture in which any node on the end-to-end path of a session can run the analysis if it can perform the analysis in a self-contained fashion without needing any post-processing [29]. Many NIDS analysis such as `Signature` detection occur at a per-session granularity. Thus, the signature detection responsibilities can be split across the nodes on each end-to-end path by dividing the sessions across the path as shown in Figure 2. This can reduce the load on node N1 by leveraging spare compute resources on N2–N5. Note, however, that the `Scan` module cannot be distributed. `Scan` detection involves counting the number of unique destinations a source contacts, which requires a complete view of all traffic to/from a given host. Thus, the ingress node alone is capable of running the analysis in a self-contained manner.

2.2 New Opportunities

Relaxing the on-path requirement: Now, the traffic on P1 might

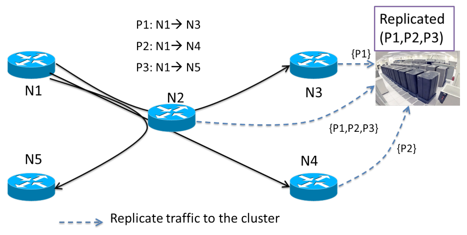

Figure 3: Replicating traffic to a compute cluster. With on-path alone, the cluster at N3 cannot be used to handle traffic on P2 and P3.

overload all nodes N1–N3 on the path. In this case, it is necessary to look for spare resources that are *off-path*. For example, nodes could locally offload some analysis to one-hop neighbors. Additionally, administrators may want to exploit compute clusters elsewhere in the network. Such consolidated clusters or datacenters are appealing because they amortize deployment and management costs.

Consider the scenario in Figure 3. The network has a compute cluster located at node N3. When the processing load on the paths P2 and P3 exceed the provisioned capacity of their on-path nodes, we can potentially replicate traffic from node N2 to node N3 and run the analysis at the cluster. This assumes that: (1) there is sufficient network bandwidth to replicate this traffic and (2) the logic to do such replication has low overhead. For (1), we note that the primary bottleneck for many NIDS deployments is typically the number of active connections and the complexity of analysis, and not volume of traffic in bytes [8]. As we will show in Section 7, we can implement a lightweight shim layer to implement (2).

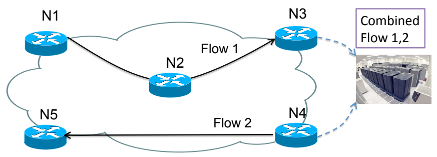

Figure 4: The analysis needs to combine Flow 1 and Flow 2 (e.g., two directions of a session or two connections in a stepping stone), but they traverse non-intersecting paths. In this case, replication is necessary to avoid detection misses.

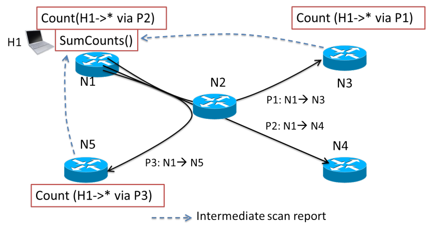

Figure 5: Aggregating intermediate results lets us distribute analyses that might be topologically constrained.

Network-wide views: Certain kinds of scenarios and analysis may need to combine traffic from different nodes. For example, "hot-potato" like effects may result in non-intersecting routing paths for the forward and reverse flows within a bidirectional session [36]. Thus, stateful NIDS analysis that needs to observe both sides of a session is impossible. A similar scenario occurs for stepping stone detection [43], if the two stages in the stepping stone do not encounter a common NIDS node. In Figure 4, traffic flows Flow 1 and Flow 2 need to be combined, but no single node can observe both flows. Thus, we need to replicate this traffic to a common location to analyze this traffic. Similarly, certain types of anomaly detection [3, 16] require a network-wide view that no single node can provide.

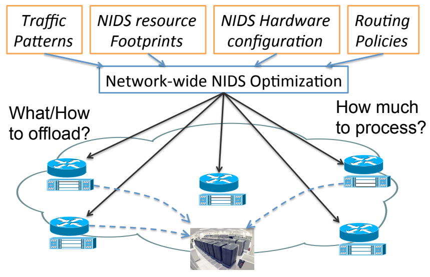

Figure 6: Network-wide framework for assigning NIDS processing and replicating responsibilities.

Aggregation for fine-grained splitting: As we saw earlier, prior work requires each type of NIDS analysis to be self-contained. Consequently, analyses such as `Scan` detection are topologically constrained. Allowing the NIDS to communicate intermediate results provides further opportunities for distributing the load. Consider the setup in Figure 5. Each node on the path runs a subset of the `Scan` analysis. The nodes send their intermediate results to an aggregation node that eventually generates alerts. (In this example, the aggregation happens at the ingress, but that is not strictly necessary.) A natural constraint here is to ensure that the result generated after aggregation is semantically equivalent to a centralized analysis. We defer to Section 6 on how we achieve this in practice.

The above scenarios highlight the need to look beyond pure on-path opportunities for distributing NIDS responsibilities in a network. In the next section, we begin with a high-level system overview before delving into the specific formulations for incorporating replication and aggregation opportunities in subsequent sections.

3. SYSTEM OVERVIEW

Our goal is to optimally assign processing, aggregation, and replication responsibilities across the network. Optimality here involves a tradeoff between the compute load on the NIDS elements and the communication costs incurred. Next, we give an overview of the key entities and parameters involved in our framework (Figure 6).

In the spirit of many recent efforts, we assume a logically centralized management module that configures the NIDS elements (e.g., [12]). This module periodically collects information about the current traffic patterns and routing policies. Such data feeds are routinely collected for other network management tasks [9]. Based on these inputs, the module runs the optimization procedures (described later) to assign NIDS responsibilities. The optimization is run periodically (e.g., every 5 minutes) or triggered by routing or traffic changes to adapt to network dynamics. Note that the network administrators only need to specify high-level policy objectives (e.g., how much link capacity to allow for replication) and set up the optimization module to receive the relevant data feeds. Afterwards, the configuration is completely automated.

We briefly describe the high-level inputs that this network-wide NIDS controller needs:

1. *Traffic and routing patterns:* This categorizes the traffic into

different logical *classes*. Each such class may be identified by a source and destination prefix-pair and some application level ports (e.g., HTTP, IRC). Let $\mathcal{T}_c$ denote the set of end-to-end sessions of the traffic class c and let $|\mathcal{T}_c|$ denote the volume of traffic in terms of the number of sessions. We initially assume that each class has a unique symmetric routing path P_c (P for *Path*),[1] and then subsequently relax this assumption. We use the notation $N_j \in P_c$ to denote that NIDS node N_j is *on the routing path*. Note that some nodes (e.g., a dedicated cluster) could be completely off-path; i.e., it does not observe traffic on any end-to-end routing path unless some other node explicitly forwards traffic to it.

2. *Resource footprints:* Each class c may be subject to different types of NIDS analyses. For example, HTTP sessions may be analyzed by a payload signature engine and through application-specific rules, while all traffic (itself a class) might be subject to `Scan` analysis. We model the cost of running the NIDS for each class on a specific *resource* r (e.g., CPU cycles, resident memory) in terms of the expected per-session resource footprint F^r_c, in units suitable for that resource (F^r for *Footprint* on r). We expect these values to be relatively stable and can be obtained either via NIDS vendors' datasheets or estimated using offline benchmarks [8]. Our approach can provide significant benefits even with approximate estimates of these F^r_c values.
3. *NIDS hardware:* Each NIDS hardware device N_j is characterized by its resource capacity Cap^r_j in units suitable for the resource r. In the general case, we assume that hardware capabilities may be different across the network, e.g., because of upgraded hardware running alongside legacy equipment. $\mathcal{N}$ denotes the set of all NIDS nodes.

Communication Costs: We model communication costs in one of two ways. First, in the case of replication, we want to bound the additional *link load* imposed by the inter-NIDS communication. This addresses the concern that network administrators may have with the additional traffic introduced by replication; this ensures that we do not overload network links (and thus avoid unnecessary packet losses). Similar to the notion of a router being on the path, we use the notation $Link_l \in P_c$ to denote that the network link l is on the path P_c. Second, for aggregation, we count the total *network footprint* imposed by the inter-NIDS communication, measured in *byte-hops*. For example, if NIDS N_1 needs to send a 10KB report to NIDS N_2 four hops away, then the total footprint is $10 \times 4 = 40$ *KB-hops*.

Given this setup, we describe the formal optimization frameworks in the following sections.

4. NIDS WITH REPLICATION

As we saw in Figure 3, we can reduce the NIDS load by replicating the traffic to nodes that are off-path if they have spare resources. In this section, we provide a general framework for combining on-path distribution with off-path replication. For the discussion, we assume that the NIDS analyses run at a *session-level* granularity. This is typical of most common types of NIDS analyses in use today [24,34]. We also assume that each class has a single symmetric routing path. (We relax this in Section 5.) Figure 7 shows the LP formulation for our framework, to which we refer throughout this section.

[1]Different classes may share the same routing path; e.g., the classes corresponding to HTTP and IRC between the same pair of source and destination prefixes are distinct logical classes but still traverse the same path.

Minimize $LoadCost$ subject to

$$LoadCost = \max_{r,j}\{Load^r_j\} \tag{1}$$

$$\forall c: \sum_{j:N_j \in P_c} \Big(p_{c,j} + \sum_{\substack{j':N_{j'} \in M_j \\ N_{j'} \notin P_c}} o_{c,j,j'}\Big) = 1 \tag{2}$$

$$\forall r,j: \; Load^r_j = \sum_{\substack{c: \\ N_j \in P_c}} \frac{F^r_c \times |\mathcal{T}_c| \times p_{c,j}}{Cap^r_j} + \sum_{\substack{j',c: \\ N_j \in M_{j'};N_j \notin P_c}} \frac{F^r_c \times |\mathcal{T}_c| \times o_{c,j',j}}{Cap^r_j} \tag{3}$$

$$\forall l: \; LinkLoad_l = BG_l + \sum_{\substack{c,j,j': \\ Link_l \in P_{j,j'};N_{j'} \in M_j}} \frac{|\mathcal{T}_c| \times o_{c,j,j'} \times Size_c}{LinkCap_l} \tag{4}$$

$$\forall l: \; LinkLoad_l \le \max\{MaxLinkLoad, BG_l\} \tag{5}$$

$$\forall c,j: \; 0 \le p_{c,j} \le 1 \tag{6}$$

$$\forall c,j,j': \; 0 \le o_{c,j,j'} \le 1 \tag{7}$$

Figure 7: LP formulation for replication

For each NIDS node, N_j, we introduce the notion of a *mirror set* $M_j \subseteq \mathcal{N}$ (M for *Mirror*) that represents a candidate set of nodes to which N_j can offload some processing. This allows us to flexibly capture different replication strategies. In the most general case all nodes are candidates, i.e., $\forall j: M_j = \mathcal{N} \setminus \{N_j\}$. In case of a single datacenter, we set $\forall j: M_j = \{N_{\mathrm{DC}}\}$ where N_{DC} is the datacenter. We can also consider local offload policies where M_j is the set of N_j's one- or two-hop neighbors. Let $P_{j,j'}$ denote the routing path between N_j and the mirror node $N_{j'}$.

At a high-level, we need to decide if a given NIDS node is going to *process* a given session or *replicate* that traffic to one of its candidate mirror nodes (or neither). We capture these determinations with two control variables. First, $p_{c,j}$ (p for *process*) specifies the fraction of traffic on the path P_c of class c that the node N_j *processes* itself. To capture offloading via replication, we have an additional control variable: $o_{c,j,j'}$ (o for *offload*) which represents the fraction of traffic on the path P_c that N_j *offloads* to its mirror node $N_{j'}$. Note that there is no need to replicate traffic to elements that are already on-path; if $N_{j'} \in P_c$ then the variables $o_{c,j,j'}$ will not appear in the formulation. The bounds in Eq (6) and (7) ensure that these variables can only take fractional values between zero and one.

Recall that our goal is to assign processing and offloading responsibilities across the network to balance the tradeoff between the *computation load* and the *communication cost*. Here, we focus on the communication cost as a given constraint on the maximum allowed link load $MaxLinkLoad$ imposed by the replicated traffic. For example, network administrators typically want to keep links at around 30–50% utilization in order to absorb sudden bursts of traffic [10].

Our main constraint is a *coverage requirement*; we want to ensure that for each class, the traffic is processed by some node either on- or off-path. Eq (2) models this by considering the sum of the locally processed fractions $p_{c,j}$ and the offloaded fractions $o_{c,j,j'}$ and setting it to 1 for full coverage.

Eq (3) captures the stress on each resource for each node. There are two sources of load on each node: the traffic it needs to process locally from on-path responsibilities (as captured by $p_{c,j}$ values) and the total traffic it processes as a consequence of other nodes offloading traffic (as captured by $o_{c,j',j}$ values) to it. The inversion in the indices for the o contribution is because the load on N_j is a function of what other $N_{j'}$s offload to it.

Then, Eq (4) models the link load on the link l imposed by the traffic between every pair of N_j and its mirror nodes. Because $|\mathcal{T}_c|$ only captures the number of sessions, we introduce an extra multiplicative factor $Size_c$ to capture the average size (in bytes) of each session of class c. We also have an additive term BG_l to capture the current load on the link due to the normal traffic traversing it before replication (BG for *BackGround*). These additive terms can be directly computed given the traffic patterns and routing policy, and as such we treat them as constant inputs in the formulation.

As discussed earlier, we bound the communication cost in terms of the maximum link load in Eq (5). The max is needed because the background load may itself exceed the given constraint $MaxLinkLoad$; in this case, Eq (5) ensures that no new traffic is induced on such overloaded links.

Here, we focus on a specific load balancing objective to minimize the maximum load across all node-resource pairs. Surveys show that overload is a common cause of appliance failure especially for NIDS and is a key cause of concern for network operators [32]. Our min-max objective will improve the robustness of the system, allowing for sudden bursts of traffic to be handled if necessary with little or no penalty. We use standard LP solvers to obtain the optimal $p_{c,j}$ and $o_{c,j,j'}$ settings which we convert into per-node processing configurations (see Section 7).

Extensions: Our framework can accommodate more general policies for capturing the stress on the links and NIDS locations. Instead of an upper bound on each $LinkLoad_l$ as the max of the two terms as shown, we can model the aggregate link utilization cost incurred across all links in terms of a piece-wise linear cost function that penalizes higher values of the incurred link load [10]. This provides a more graceful tradeoff rather than a tight upperbound of $MaxLinkLoad$ on the link load costs. Similarly, instead of capturing $LoadCost$ as the max over all $Load_j^r$, we can model it as a cost function that penalizes higher values of load or as weighted combinations of the $Load_j^r$ values.

5. SPLIT TRAFFIC ANALYSIS

Next, we focus on the scenario from Figure 4 in which we need to replicate traffic because the forward and reverse paths are asymmetric. For simplicity, we focus on the case where there is one datacenter node, rather than generalized mirror sets. Thus, we use $o_{c,j}$ instead of $o_{c,j,j'}$, implicitly fixing a single mirror node N_{DC} for all N_j.

To model this scenario, we modify how the routing paths for each class $\mathcal{T}_c$ are specified. In the previous section, we assumed that the forward and reverse paths are symmetric and thus each $\mathcal{T}_c$ has a unique path P_c. In the case where these paths are asymmetric or non-overlapping, instead of defining a single set of eligible NIDS nodes P_c, we define three types of nodes:

1. P_c^{fwd} that can observe the "forward" direction;[2]
2. P_c^{rev} that can observe the "reverse" direction; and
3. $P_c^{common} = P_c^{fwd} \cap P_c^{rev}$, which may be empty.

We assume here that these types of nodes can be identified from the network's routing policy [31]. Having identified these common, forward, and reverse nodes, we split the coverage constraint in Eq (2) into two separate equations:

$$\forall c : cov_c^{fwd} = \sum_{j:N_j \in P_c^{common}} p_{c,j} + \sum_{j:N_j \in P_c^{fwd}} o_{c,j}^{fwd} \tag{8}$$

$$\forall c : cov_c^{rev} = \sum_{j:N_j \in P_c^{common}} p_{c,j} + \sum_{j:N_j \in P_c^{rev}} o_{c,j}^{rev} \tag{9}$$

Now, for stateful analysis, the notion of coverage is meaningful only if both sides of the session have been monitored. Thus, we model the effective coverage as the minimum of the forward and reverse coverage values:

$$\forall c : cov_c = \min\{cov_c^{fwd}, cov_c^{rev}, 1\} \tag{10}$$

We make three observations here. First, the locally processed fraction $p_{c,j}$ only applies for nodes in P_c^{common}. Second, we separately specify the coverage guarantee for the forward and reverse directions for each $\mathcal{T}_c$ and cap the effective coverage at 1. Third, we also allow the nodes in P_c^{common} to offload processing to the datacenter. (Because the nodes in P_c^{common} also appear on P_c^{fwd} and P_c^{rev}, they have corresponding $o_{c,j}^{fwd}$ and $o_{c,j}^{rev}$ variables.)

Now, it may not always be possible to ensure complete coverage for some deployment scenarios. That is, for a particular combination of forward-reverse paths, and a given constraint on the maximum allowable link load, we may not have a feasible solution to ensure that each $cov_c = 1$.[3] In this case, we want to minimize detection misses and thus, we introduce a new term in the minimization objective to model the fraction of traffic that suffers detection misses because we cannot monitor both sides of the connection. That is,

$$MissRate = \frac{\sum_c (1 - cov_c) \times |\mathcal{T}_c|}{\sum_c |\mathcal{T}_c|} \tag{11}$$

Given the $MissRate$, we update the objective to be:

$$\text{Minimize: } LoadCost + \gamma MissRate$$

with γ set to a large value to have a very low miss rate.

In summary, the formulation to handle such split traffic is as follows. We retain the same structure for the compute load and link load equations as in Eq (3) and Eq (4) respectively. (There are small changes to incorporate the notion of o_{cj}^{fwd} and $o_{c,j}^{rev}$. We do not show these for brevity.) We replace the single coverage equation in Eq (2) with the new coverage models in Eqs (8), (9), and (10). Rather than force each coverage value to be 1, which could be infeasible to achieve, we focus instead on minimizing the effective miss rate by changing the objective function.

One subtle issue here is that we need to ensure that the nodes on the forward and reverse path act in a consistent manner. For example, we cannot have the forward direction of a session being processed locally at N_j and the reverse direction offloaded. We achieve this consistency by using bidirectional semantics when mapping the decision variables into actual per-flow actions executed by each node as described in Section 7.

Extensions: We can extend the model to quantify $MissRate$ in terms of the class c with the largest fraction of detection misses (i.e., $MissRate = \max_c\{1 - cov_c\}$), or consider a general weighted combination of these coverage values to indicate higher priority for some traffic.

[2]We assume a well-defined notion of forward and reverse directions, say based on the values of the IP addresses.

[3]Note that this is in contrast to the formulation from Section 4, where there is always a feasible solution to get full coverage by simply running the analyses locally, but potentially incurring a higher $LoadCost$.

6. NIDS WITH AGGREGATION

Next, we discuss scaling via aggregation. The high-level idea is to split a NIDS task into multiple sub-tasks that can be distributed across different locations. Each NIDS node generates *intermediate reports* that are sent to an aggregation point to generate the final analysis result. As a concrete example, we focus on the `Scan` detection module that counts the number of distinct destination IP addresses to which a given source has initiated a connection in the previous measurement epoch. The high-level approach described here can also be extended to other types of analysis amenable to such aggregation (e.g., DoS or flood detection).

For clarity, we focus on using aggregation without replication and assume a single symmetric path for each class. This means that we just need to assign the local processing responsibilities captured by the $p_{c,j}$ variables.[4]

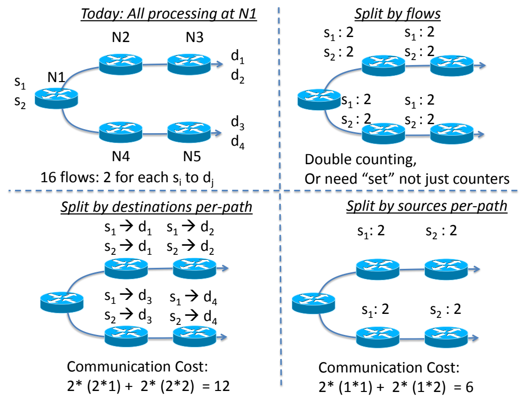
Today: All processing at N1
N2
N3
N1
s_1
s_2
d_1
d_2
d_3
d_4
N4
N5
16 flows: 2 for each s_i to d_j
Split by flows
s_1 : 2
s_2 : 2
Double counting,
Or need "set" not just counters
Split by destinations per-path
$s_1 \rightarrow d_1$ $s_1 \rightarrow d_2$
$s_2 \rightarrow d_1$ $s_2 \rightarrow d_2$
$s_1 \rightarrow d_3$ $s_1 \rightarrow d_4$
$s_2 \rightarrow d_3$ $s_2 \rightarrow d_4$
Communication Cost:
2* (2*1) + 2* (2*2) = 12
Split by sources per-path
s_1: 2
s_2 : 2
Communication Cost:
2* (1*1) + 2* (1*2) = 6

Figure 8: Different options for splitting the `Scan` detection responsibilities

Because the choice of intermediate reports and aggregation points may vary across different detection tasks, we use a general notion of *network distance* between node N_j and the location to which these reports are sent. This is captured by $D_{c,j}$ (D for *Distance*); the indices indicate that the location may depend on the specific class c. For example, in `Scan` detection, we may choose to send the reports back to the ingress for the host because it is in the best position to decide if an alert should be raised, e.g., based on past behavior observed for the hosts.

We do, however, need to be careful in choosing the granularity at which we distribute the work across nodes. Consider the example in Figure 8 where our goal is to count the number of destinations that each source contacts. Suppose there are two sources s_1, s_2 contacting four destinations d_{1-4} as shown and there are two flows for every src-dst pair. Here, each NIDS runs a per-src `Scan` counting module on its assigned subset of the traffic. Then, each node sends these local per-src counters to the aggregation point, which outputs the final result of suspicious sources. Now, we could choose three different strategies to split the monitoring responsibilities:

1. *Flow-level:* The nodes on a path split the traffic traversing that path on a per-flow basis, run a local `Scan` detection module on the set of observed flows and send intermediate reports back to the ingress.

[4]We retain the c subscript for notational consistency; for `Scan` detection the classes simply correspond to end-to-end paths rather than application-level classes.

Minimize $LoadCost + \beta \times CommCost$ subject to

$$LoadCost = \max_{r,j}\{Load_j^r\} \tag{12}$$

$$CommCost = \sum_{c,j}(|\mathcal{T}_c| \times p_{c,j}) \times Rec_c \times D_{c,j} \tag{13}$$

$$\forall c: \sum_{j: N_j \in P_c} p_{c,j} = 1 \tag{14}$$

$$\forall r,j: \; Load_j^r = \sum_{\substack{c:\\ N_j \in P_c}} \frac{F_c^r \times |\mathcal{T}_c| \times p_{c,j}}{Cap_j^r} \tag{15}$$

$$\forall c,j: \; 0 \le p_{c,j} \le 1 \tag{16}$$

Figure 9: LP formulation for aggregation

2. *Destination-level:* We do a split based on destinations for each path. In the example, node N2 checks if each source contacted d_1, node N3 for d_2, and so on.
3. *Source-level:* Each node focuses on a subset of the sources on each path; e.g., N2 and N3 monitor s_1 and s_2, respectively.

Notice that with a flow-based split, if we only report per-src counters, then we could end up overestimating the number of destinations if a particular source-destination pair has multiple flows. In this case, each node must report the full set of $\langle src, dst \rangle$ tuples, thus incurring a larger communication cost. The aggregator then has to compute the logical union of the sets of destinations reported for each source. With a destination-based split, we do not have this double counting problem. The aggregator simply adds up the number of destinations reported from each node on each path. In the worst case, however, the number of entries each node reports will be equal to the number of sources. Thus, the total communication cost could be 12 units, assuming aggregation is done at N1: each node sends a 2-row report (one row per src), the report from N2, N4 traverses one hop and those from N3, N5 take two hops. The third option of splitting based on the sources provides both a correct result without over-counting and also a lower communication cost of 6 units. Each node sends a report consisting of the number of destinations each source contacts and the aggregator can simply add up the number of destinations reported across the different paths for each source.[5] Thus, we choose the source-level split strategy since it offers both correct and communication-minimal operation in the common case. In general, we envision our NIDS controller specifying this reporting schema to a shim layer running on each node as we discuss in the next section.

In practice, there are a few natural cases that cover most common NIDS modules that can benefit from such aggregation (e.g., per-src, per-destination). Having chosen a suitable granularity of intermediate reports, we need as input the *per-report size* Rec_c (in bytes) for class c.[6]

As in the previous section, we want to balance the tradeoff between the computation cost and the communication cost. Because the size of the reports (at most a few MB) is unlikely to impact the link load adversely, we drop the $MaxLinkLoad$ constraints in Eqs (4), (5). Instead, we introduce a new *communication cost* term $CommCost$ in the objective, with a weight factor β, which is scaled appropriately to ensure that the load and communication cost terms are comparable. We have the familiar coverage constraint in

[5]Assuming that there is a unique and fixed path for a specific source-destination during this measurement epoch.

[6]This also depends on how these reports are encoded, e.g., key-value pairs for a source-split.

Eq (14), and the resource load model in Eq (15). (Because there is no traffic replication, the "offload" o variables do not appear in this formulation.) The additional equation required here is to model the total communication cost $CommCost$ in Eq (13). For each entry, this is simply the product of the volume of traffic, the per-unit record size, and the network distance as shown.

7. IMPLEMENTATION

We start by describing how the management engine translates the output of the LP optimizations into device configurations. Then, we describe how we implement these management decisions using a *shim* layer that allows us to run off-the-shelf NIDS software.

7.1 Optimization and configurations

We solve the LP formulations described in the previous sections using off-the-shelf LP solvers such as `CPLEX`. Given the solution to the optimization, we run a simple procedure to convert the solution into a configuration for each shim instance. The main idea is to map the decision variables—$p_{c,j}$ and $o_{c,j,j'}$ values—into non-overlapping *hash ranges*. For each c, we first run a loop over the $p_{c,j}$ values, mapping each to a hash-range, and extending the range as we move to the next j. We then run a similar loop for the $o_{c,j,j'}$. (The specific order of the NIDS indices does not matter; we only require some order to ensure the ranges are non-overlapping.) Because the optimization frameworks ensure that these $p_{c,j}$ and $o_{c,j,j'}$ add up to 1 for each c, we are guaranteed that the union of these hash ranges covers the entire range.

7.2 Shim layer

To allow network operators to run their existing NIDS software without needing significant changes, we interpose a lightweight *shim* between the network and the NIDS. We implement this using the Click modular software router [15] with a combination of default modules and a custom module (255 lines of C++ code). The shim maintains persistent tunnels with its mirror node(s) to replicate the traffic and uses a virtual TUN/TAP interface to the local NIDS process. This requires a minor change to the way the NIDS process is launched so that it reads from the virtual interface rather than a physical interface. We tested two popular NIDS: Bro [24] and Snort [34]; both had no difficulties running unmodified on top of the shim layer.

As a packet arrives, the shim computes a lightweight hash [5] of the IP 5-tuple (protocol, src/dst IPs and ports). It looks up the corresponding class (e.g., based on the port numbers and src/dst IPs) to infer the assigned hash range (from the above configuration) and decides whether to send this packet to the local NIDS process, replicate it to a mirror node, or neither. One subtle issue here is that we need to ensure that this hash is bidirectional to ensure that both directions are consistently "pinned" or offloaded to the same node. For example, we can achieve this by converting the IP 5-tuple into a canonical form such that the source IP is always less than the destination IP before computing the hash [37]. For aggregation, the hash is over the appropriate field used for splitting the task, i.e., per-source or per-destination depending on the analysis.

7.3 Aggregation

Aggregation requires two components: (1) a new shim module at each NIDS node that periodically sends reports; and (2) an aggregator to post-process these reports. As discussed earlier, the choice of reporting schema and where the aggregation runs may vary across different NIDS tasks. In the specific case of `Scan` detection, we want to report sources that contact $> k$ destinations and send these reports to the gateway nodes for each host. Now, the measured value at an individual NIDS may not exceed k, but the aggregate might. Thus, we apply the threshold k only at the aggregator and configure each individual NIDS to have a reporting threshold of $k = 0$, to retain the same detection semantics as running the scan detector at the gateway node for each host.

8. EVALUATION

We use real network topologies from educational backbones (Internet2, Geant), inferred PoP-level topologies from Rocketfuel [35], and a multi-site Enterprise topology [30]. For each topology, we construct a traffic matrix for every pair of ingress-egress PoPs using a gravity model based on city populations [27], with shortest-path routing based on hop counts. For brevity, we consider a single aggregate traffic class and do not partition traffic based on application port numbers.

8.1 System evaluation

Computation time: Table 1 shows the time to compute the optimal solution for different PoP-level topologies using an off-the-shelf LP solver (`CPLEX`). This result shows that the time to recompute optimal solutions is well within the timescales of network reconfigurations (typically on the order of few minutes).

Topology	# PoPs	Time (s)	
		Replication	Aggregation
Internet2	11	0.05	0.02
Geant	22	0.10	0.02
Enterprise	23	0.10	0.01
TiNet (AS3257)	41	0.29	0.02
Telstra (AS1221)	44	0.40	0.03
Sprint (AS1239)	52	1.30	0.05
Level3 (AS3356)	63	1.19	0.04
NTT (AS2914)	70	1.59	0.11

Table 1: Time to compute the optimal solution for the replication and aggregation formulations.

Shim overhead: The hash computations and lookups impose little overhead over the processing and packet capture that a NIDS has to run natively. In our microbenchmarks, the shim implementation does not introduce any (additional) packet drops up to an offered load of 1 Gbps for a single-threaded Bro or Snort process running on a Intel Core i5 2.5GHz machine.

Live emulation in Emulab: To investigate the benefits of off-path replication, we evaluate our system with an emulated Internet2 topology with 11 nodes using Emulab [40]. We implemented an offline traffic generator using `Scapy` [28] that takes as input the topology, traffic matrix, and template traces, and that generates a traffic trace according to these. We used real full-payload packet traces as the "seed" templates [18]. To faithfully emulate the ordering of packets within a logical session, we introduced a stateful "supernode" that is logically connected to every network ingress. This supernode injects packets within each session in order and at the appropriate ingress using the `BitTwist` tool [4]. Each NIDS node runs on a Pentium III 850 Mhz node with 512 MB of RAM[7] running Snort (version 2.9.1) using the default configuration of rules and signatures.

Figure 10 shows the total number of CPU instructions used by the Snort process, measured using the `PAPI` performance instru-

[7]The choice of low-end nodes was to ensure repeatability as it is hard to obtain a large number of high-end nodes for extended periods of time on Emulab.

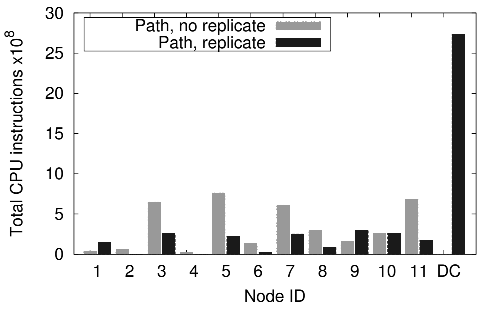

Figure 10: Maximum absolute CPU usage of each NIDS node in our Emulab experiment

mentation library [23]), on each NIDS node for the emulated Internet2 topology with 11 nodes. The result shows the configurations for two NIDS architectures: *Path, No replicate* [29] and *Path, Replicate* which represents our framework from Section 4. For our setup, we ran the formulation with a single data center (DC) with 8$\times$ the capacity of the other NIDS nodes and assuming $MaxLinkLoad = 0.4$. (We did not explicitly instantiate a 8$\times$ capacity.) Figure 10 confirms that replication provides 2$\times$ reduction in resource usage on the maximally loaded node (excepting the DC). This result is identical to that obtained using trace-driven simulations, as will be shown in Figure 13, allowing us to conclude that sensitivity analysis performed in Section 8.2 is representative of live performance.

8.2 Replication: Sensitivity analysis

Due to the difficulty of scaling our Emulab setup for larger topologies and further sensitivity analysis, we use trace-driven analysis for these evaluations.

Setup: To model the total traffic volume, we start with a baseline of 8 million sessions for the Internet2 network with 11 PoPs, and then scale the total volume for other topologies linearly proportional to the number of PoPs. We model the link capacities $LinkCap_l$ as follows. We compute the traffic volume traversing the maximum congested link (assuming the above shortest path routes). Then, we set the link capacity of each to be 3$\times$ this traffic load on the most congested link. As such, $\max_l\{BG_l\} = 0.3$; this reflects typical link utilization levels in networks today [10]. To model the node capacities Cap_j^r, we simulate the Ingress-only deployment and find the maximum resource requirement across the network, and provision each node with this inferred capacity. Thus, by construction the Ingress deployment has a maximum compute load of one. We model a single data center with $\alpha\times$ the capacity of the other NIDS nodes.

In this discussion, we examine the effects of varying the location and capacity of the data center node (Cap_{DC}^r), the maximum allowed link load with replication ($MaxLinkLoad$), alternative local replication architectures, and the impact of traffic variability.

Choice of datacenter location: The first parameter of interest is the placement of the datacenter. Here, we fix the datacenter capacity to be 10$\times$ the single NIDS capacity, but choose different locations based on four natural strategies: (1) the PoP from which most traffic originates, (2) the PoP that observes the most traffic, including traffic for which this is a transit PoP, (3) the PoP which lies on the most end-to-end shortest paths, and (4) the PoP which has the smallest average distance to every other PoP (the medoid).

We find that for most topologies the gap between the different placement strategies is very small and that placing the datacenter at the PoP that observes the most traffic works best across all topologies. (Not shown; please see our extended report [13].) Thus for the rest of the evaluation, we choose this placement strategy.

Effect of increasing allowed link load: Next, we fix the placement of the datacenter as described above and its capacity to 10$\times$, and study the impact of increasing $MaxLinkLoad$ in Figure 11. For most topologies, we see diminishing returns beyond $MaxLinkLoad = 0.4$, since at that value, the compute load on the datacenter is close to the load on the maximum NIDS node as well. This result suggests that network administrators need not be concerned about the additional load induced by the replication traffic since we can achieve near-optimal benefits at 40% link utilization.

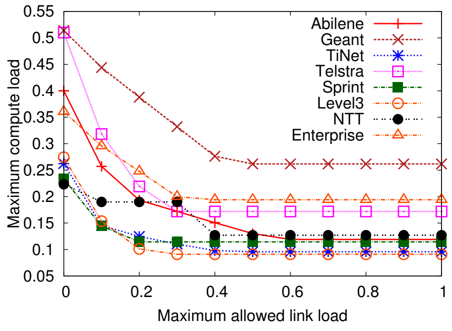

Figure 11: Varying $MaxLinkLoad$ with datacenter capacity $=$ 10$\times$.

Increasing the data center capacity: A natural question then is how much to provision the datacenter. To address this, we studied the impact of varying the datacenter capacity. Most topologies show a natural diminishing property as we increase the capacity, with the "knee" of the curve occurring earlier with lower link load. This is expected; with lower $MaxLinkLoad$, there are fewer opportunities for replicating traffic to the datacenter and thus increasing the datacenter capacity beyond 8–10$\times$ does not really help (not shown).

Visualizing maximum loads: To better understand the previous effects, we visualize a high-level summary of how the optimization allocates the compute and offload responsibilities throughout the network. We consider four configurations here: $MaxLinkLoad \in \{0.1, 0.4\}$ and a datacenter capacity Cap_{DC}^r of 2$\times$ and 10$\times$. Figure 12 shows the difference between the compute load on the datacenter node (*DCLoad*) and the maximum compute load on non-datacenter NIDS nodes (*MaxNIDSLoad*) for the different topologies. We see that at low link load and high data center capacity ($MaxLinkLoad = 0.1$ and Cap_{DC}^r of 10$\times$), the datacenter is underutilized. With larger link loads or lower link capacity, we find that the load stress on the datacenter is the same as the maximum load across the network (i.e., the gap is zero).

Comparison to alternatives: Using the previous results as guidelines, we pick a configuration with the datacenter capacity fixed at 10$\times$ the single NIDS capacity and with $MaxLinkLoad = 0.4$. Figure 13 compares this configuration (labeled *Path, Replicate*) against two alternatives: (1) today's *Ingress*-only deployment where NIDS functions run at the ingress of a path; and (2) *Path, No Replicate*, strict on-path NIDS distribution [29]. One concern is that our datacenter setup has more aggregate capacity. Thus, we also consider a *Path, Augmented* approach where each of the $|\mathcal{N}|$ NIDS

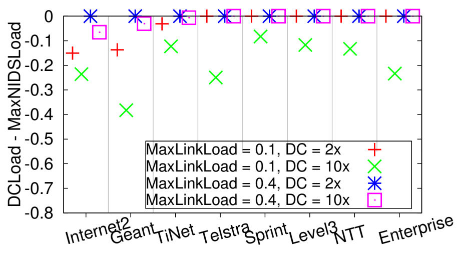

Figure 12: Comparing the compute load on the datacenter vs. maximum load on interior NIDS nodes.

nodes gets a $\frac{1}{|\mathcal{N}|}$ share of the $10\times$ additional resources. The fact that we can consider these alternative designs within our framework further confirms the *generality* of our approach.

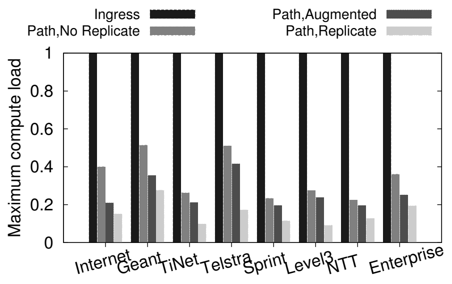

Figure 13: Maximum compute load across topologies with different NIDS architectures.

Recall that the current deployments of *Ingress*-only have a maximum compute load of one by construction. The result shows that *Path, Replicate* has the best overall performance; it can reduce the maximum compute load by $10\times$ compared to today's deployments and up to $3\times$ compared to the proposed on-path distribution schemes.

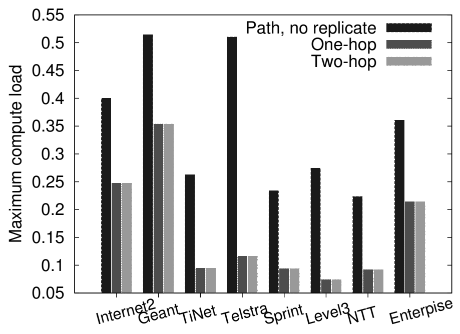

Figure 14: Local one- and two-hop replication.

Local offload: The above results consider a setup where the network administrator has added a new datacenter. Alternatively, they can use the existing NIDS infrastructure with *local* replication strategies. Specifically, we consider the mirror sets (M_js) consisting of 1-hop or 2-hop neighbors in addition to the existing on-path distribution. Figure 14 compares the maximum compute load vs. a pure on-path distribution again setting $MaxLinkLoad = 0.4$. Across all topologies, allowing replication within a one-hop radius provides up to $5\times$ reduction in the maximum load. We also see that going to two hops does not add significant value beyond one-hop offload. This suggests a replication-enhanced NIDS architecture can offer significant benefits even without needing to augment the network with additional compute resources.

Performance under traffic variability: The results so far consider a static traffic matrix. Next, we evaluate the effect of traffic variability. To obtain realistic temporal variability patterns, we use traffic matrices for Internet2 [14]. From this, we compute empirical CDFs of how each element in a traffic matrix varies (e.g., probability that the volume is between $0.6\times$ and $0.8\times$ the mean). Then, using these empirical distributions we generate 100 time-varying traffic matrices using the gravity model for the mean volume.

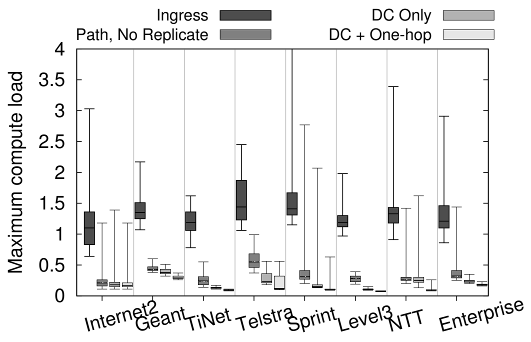

Figure 15: Comparison between NIDS architectures in the presence of traffic variability.

Figure 15 summarizes the distribution of the peak load across these 100 runs using a box-and-whiskers plot showing the minimum, 25th %ile, median, 75th %ile, and the maximum observed load. We consider four NIDS architectures: *Ingress*; *Path, No replicate*; *Path, replicate* with a datacenter node $10\times$ capacity (labeled `DC Only`); and *Path, replicate* with the flexibility to offload responsibilities to either a datacenter and within a 1-hop radius (labeled `DC + One-hop`). We find that the replication-enabled NIDS architectures outperform the non-replication strategies significantly, with the median values roughly mirroring our earlier results. The worst-case performance of the no-replication architectures can be quite poor, e.g., much larger than 1. (Ideally, we want the maximum compute load to be less than 1.) We also analyzed how the augmentation strategy from Figure 13 performs; the worst-case load with the *Path, Augmented* case is $4\times$ more than the replication enabled architecture (not shown).

8.3 Replication with routing asymmetry

In this section, we evaluate how replication is effective for scenarios where the forward and reverse flows may not traverse the same route as we saw in Section 2.

We emulate routing asymmetry as follows. For each ingress-egress pair, we assume the forward traffic traverses the expected shortest path from the ingress to the egress; i.e., P_c^{fwd} is the shortest-path route. However, we set the reverse path P_c^{rev} such that the expected *overlap* (over all ingress-egress pairs) between the forward and reverse paths reaches a target overlap ratio θ. Here, we measure the overlap between two paths P_1 and P_2 in terms of the Jaccard similarity index: $\frac{P_1 \cap P_2}{P_1 \cup P_2}$, which is maximum (= 1) when they are identical and lowest (= 0) if there is no overlap. For each end-to-end path, we precompute its overlap metric with every other path. Then, given a value of θ' (drawn from a Gaussian distribution with mean = θ and standard deviation = $\frac{\theta}{5}$), we find a path from this precomputed set that is *closest* to this target value.[8] For each target θ, we generate 50 random configurations. For each configuration, we run the extended formulation from Section 5 for the *Ingress*-only architecture, the *Path, no replicate* architecture, and our proposed framework with a datacenter. We report the *median* across the 50 runs for two metrics: the detection *miss rate* — the total fraction of traffic that could not be analyzed effectively by any NIDS node — and the compute load as in the previous evaluations.

Figure 16 shows the median miss rate as a function of the overlap factor for the different configurations. We see that the miss rate with an *Ingress*-only setup is greater than 85% even for high values of the overlap metric. The $MaxLoad$ curve in Figure 17 is interesting because *Ingress* is lower than the other configurations. The reason is that there is little useful work being done here — It ignores more than 90% of the traffic! Another curious feature is that $MaxLoad$ for the replication architecture first increases and then decreases. In this setup with low overlap, the datacenter is the most loaded node. At low θ, however, the $MaxLinkLoad$ constraint limits the amount of traffic that can be offloaded and thus the datacenter load is low.

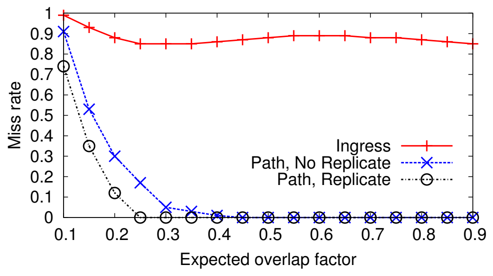

Figure 16: Detection miss rate vs. degree of overlap

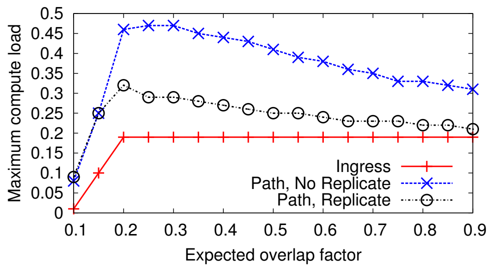

Figure 17: Maximum load vs. degree of overlap

8.4 NIDS with aggregation

In this section, we highlight the benefits of aggregation using the framework from Section 6. As discussed earlier, we focus on `Scan` detection.

[8]The exact details of how these paths are chosen or the distribution of the θ values are not the key focus of this evaluation. We just need some mechanism to generate paths with a target overlap ratio.

Figure 18 shows how varying β trades off the communication cost ($CommCost$) and compute cost ($LoadCost$) in the resulting solution, for the different topologies. Because different topologies differ in size and structure, we normalize the x- and y-axes using the maximum observed $LoadCost$ and $CommCost$ respectively over the range of β for each topology. As such, the point closest to the origin can be viewed as the best choice of β for the corresponding topology. This figure shows that for many topologies, there are choices of β that yield relatively low $CommCost$ and $LoadCost$ simultaneously, e.g., both being less than 40% of their maximums.

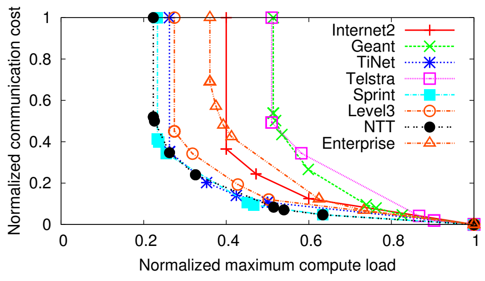

Figure 18: Tradeoff between the compute load and communication cost with aggregation as we vary β

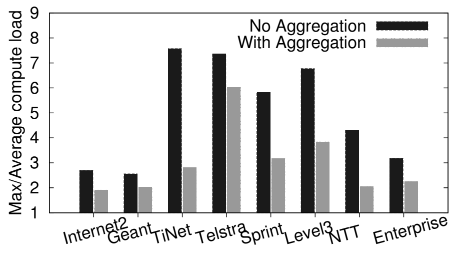

Figure 19: Ratio between maximum and average compute load with and without aggregation.

To illustrate the load balancing benefits of aggregation, Figure 19 shows the ratio of the compute load of the most loaded node to the average compute load. Here, for each topology, we pick the value of β that yields the point closest to the origin in Figure 18. A higher number represents a larger variance or imbalance in the load. Figure 19 compares this ratio to the same ratio when no aggregation is used. As we can see, aggregation reduces the load imbalance substantially (up to $2.7\times$) for many topologies.

8.5 Summary of key results

Our main results are:

- The optimization step and shim impose low overhead.
- Administrators need not worry about optimal choice of data center location, capacity, or the maximum link load. Our approach provides benefits over a range of practical and intuitive choices.
- Replication reduced the maximum compute load by up to $10\times$ when we add a NIDS cluster or up to $5\times$ with one-hop offload.

- In the presence of traffic dynamics, replication provided up to an order of magnitude reduction in maximum load.
- Replication reduced the detection miss rate from 90% to zero in the presence of partially overlapping routes.
- Aggregation reduced the load imbalance by up to $2.7\times$.

9. DISCUSSION

Consistent configurations: One concern with distribution is ensuring consistency when configurations are recomputed. We could use standard techniques from the distributed systems literature (e.g., two-phase commit [21]). We can also use simpler domain-specific solutions; e.g., whenever new configurations are pushed out, the NIDS nodes continue to honor both the previous and new configurations during the transient period. This may potentially duplicate some work, but ensures correctness of operation.

Shim for higher line-rates: Our current shim implementation imposes close to zero overhead for a single-threaded NIDS running on a single core machine for traffic up to 1 Gbps. We plan to extend our implementation using recent advances in packet capture [25, 26].

Robustness to dynamics: A sudden, significant shift in traffic patterns (adversarial or otherwise) could render the current distribution strategies ineffective. One approach to counter this is to allow for some "slack" (e.g., using the 80-th percentile values instead of the mean) in the input traffic matrices to tolerate such sudden bursts.

Extending to NIPS and active monitoring: Our approach can be generally applied to any *passive* traffic monitoring system without affecting the forwarding paths or latency of traffic. Our framework can also be extended to the case of intrusion prevention systems (NIPS), though unlike NIDS, NIPS are on the critical forwarding path which raises two additional issues that we need to handle. These arise from the fact that we are not actually replicating traffic in this case; rather, we are *rerouting* it. First, we can no longer treat the BG_l as a constant in the formulation. Second, we need to ensure that the latency penalty for legitimate traffic due to rerouting is low.

Combining aggregation and replication: As future work we plan to explore if a unified formulation that combines both opportunities offers further improvements. For example, we might be able to use replication to reduce the communication cost of aggregation. One challenge is that the analyses benefiting from aggregation may need to split the traffic at a different granularity (e.g., per source) vs. those exploiting replication (e.g., stateful signature matching on a per-session basis). Thus, we need a more careful shim design to avoid duplicating the effort in packet capture across different nodes in order to combine these ideas.

10. RELATED WORK

Scaling NIDS hardware: NIDS involve computationally intensive tasks (e.g., string and regular-expression matching). There are many proposals for better algorithms for such tasks (e.g., [33]), using specialized hardware such as TCAMs (e.g., [20, 42]), FPGAs (e.g., [17]), or GPUs (e.g., [38]). The dependence on specialized hardware increases deployment costs. To address this cost challenge, there are ongoing efforts to build scalable NIDS on commodity hardware to exploit data-level parallelism in NIDS workloads (e.g., [37,39]). These efforts focus on scaling *single-vantage-point* implementations and are thus complementary to our work. Our framework allows administrators to optimally use their existing hardware or selectively add NIDS clusters.

NIDS management: Our use of centralized optimization to assign NIDS responsibilities follows in the spirit of our prior work [29]. The approach we propose here extends our prior work in three key ways. First, we generalize on-path distribution to include replication and aggregation. Second, this previous framework cannot handle the types of split-traffic analysis with asymmetric routes as we showed in Figure 16. Third, on a practical note, this past approach requires source-level changes to the NIDS software. In contrast, our implementation allows administrators to run off-the-shelf NIDS software.

Offloading NIDS: A recent proposal makes a case for outsourcing all network processing functionality including NIDS to cloud providers [32]. While this may work for small businesses and enterprises, larger enterprises and ISPs would likely retain in-house infrastructure due to security and policy considerations. Furthermore, this proposal does not focus on computation-communication tradeoffs. Our approach can also incorporate a cloud datacenter and can offer ways to augment existing infrastructure instead of getting rid of it altogether.

Distributed NIDS: Prior work makes the case for network-wide visibility and distributed views in detecting anomalous behaviors (e.g., [16]). These focus primarily on algorithms for combining observations from multiple vantage points. Furthermore, specific attacks (e.g., DDoS attacks, stepping stones) and network scenarios (e.g., asymmetric routing as in Section 2) inherently require an aggregate view. Our focus is not on the algorithms for combining observations; rather, we build a framework for enabling such aggregated analysis.

11. CONCLUSIONS

While there are many advances in building better NIDS hardware, there is a substantial window before networks can benefit from these in practice. Our work complements existing research in scaling NIDS hardware with techniques to better utilize and augment existing NIDS deployments. To this end, we proposed a general NIDS architecture to leverage three opportunities: offloading processing to other nodes on a packet's routing path, traffic replication to off-path nodes (e.g., to NIDS clusters), and aggregation to split expensive NIDS tasks. We implemented a lightweight shim that allows networks to realize these benefits with little to no modification to existing NIDS software. Our results on many real-world topologies show that this architecture reduces the maximum compute load significantly, provides better resilience under traffic variability, and offers improved detection coverage for scenarios needing network-wide views.

Acknowledgements

We are grateful to Geoff Voelker for commenting on drafts of this paper and to Chad Spensky for initial discussions on this research. This work was supported in part by NSF grants 0831245 and 1040626, and by grant number N00014-10-1-0155 from the Office of Naval Research.

12. REFERENCES

[1] Powering virtual network services. http://embrane.com.

[2] ZScaler Cloud Security. http://www.zscaler.com.

[3] Allman, M., Kreibich, C., Paxson, V., Sommer, R., and Weaver, N. Principles for developing comprehensive network visibility. *In Proc. HOTSEC'08*, 2008.

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

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

[6] Cantieni, G. R., Iannaccone, G., Barakat, C., Diot, C., and Thiran, P. Reformulating the monitor placement problem: optimal network-wide sampling. *In Proc. CoNEXT '06*, 2006.

[7] Cisco blade servers. http://www.cisco.com/en/US/products/ps10280/index.html.

[8] Dreger, H., Feldmann, A., Paxson, V., and Sommer, R. Predicting the resource consumption of network intrusion detection systems. *In Proc. SIGMETRICS '08*, 2008.

[9] Feldmann, A., Greenberg, A., Lund, C., Reingold, N., Rexford, J., and True, F. Deriving traffic demands for operational IP networks: methodology and experience. *IEEE/ACM Trans. Netw.*, 9(3):265–280, June 2001.

[10] Fortz, B., Rexford, J., and Thorup, M. Traffic engineering with traditional IP routing protocols. *Communications Magazine, IEEE*, 40(10):118 – 124, Oct 2002.

[11] Gibb, G., Zeng, H., and McKeown, N. Outsourcing network functionality. In *Proc. HotSDN*, 2012.

[12] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. A clean slate 4D approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, Oct. 2005.

[13] Heorhiadi, V., Reiter, M. K., and Sekar, V. Balancing computation-communication tradeoffs in scaling network-wide intrusion detection systems. Technical Report TR12-001, UNC Chapel Hill, 2012.

[14] Internet2 trafficx matrices. http://www.cs.utexas.edu/~yzhang/research/AbileneTM.

[15] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.

[16] Lakhina, A., Crovella, M., and Diot, C. Diagnosing network-wide traffic anomalies. *In Proc. SIGCOMM '04*, 2004.

[17] Lee, J., Hwang, S. H., Park, N., Lee, S.-W., Jun, S., and Kim, Y. S. A high performance NIDS using FPGA-based regular expression matching. *In Proc. SAC '07*, 2007.

[18] M57 packet traces. https://domex.nps.edu/corp/scenarios/2009-m57/net/.

[19] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[20] Meiners, C. R., Patel, J., Norige, E., Torng, E., and Liu, A. X. Fast regular expression matching using small TCAMs for network intrusion detection and prevention systems. *In Proc. USENIX Security'10*, 2010.

[21] Mohan, C. and Lindsay, B. Efficient commit protocols for the tree of processes model of distributed transactions. *SIGOPS Oper. Syst. Rev.*, 19(2):40–52, Apr. 1985.

[22] Network security spending to soar in the next 5 year. http://www.v3.co.uk/v3-uk/news/1998293/network-security-spending-soar.

[23] PAPI: Performance application programming interface. http://icl.cs.utk.edu/papi/.

[24] Paxson, V. Bro: a system for detecting network intruders in real-time. *In Proc. Usenix Security'98*, 1998.

[25] Pfq homepage. http://netserv.iet.unipi.it/software/pfq/.

[26] Pf_ring. http://www.ntop.org/products/pf_ring/.

[27] Roughan, M. Simplifying the synthesis of internet traffic matrices. *SIGCOMM Comput. Commun. Rev.*, 35(5):93–96, Oct. 2005.

[28] Scapy packet manipulation toolkit. http://www.secdev.org/projects/scapy/.

[29] Sekar, V., Krishnaswamy, R., Gupta, A., and Reiter, M. K. Network-wide deployment of intrusion detection and prevention systems. *In Proc. CoNEXT '10*, 2010.

[30] Sekar, V., Ratnasamy, S., Reiter, M. K., Egi, N., and Shi, G. The middlebox manifesto: enabling innovation in middlebox deployment. *In Proc. HotNets '11*, 2011.

[31] Shaikh, A. and Greenberg, A. Ospf monitoring: architecture, design and deployment experience. *In Proc. NSDI'04*, 2004.

[32] Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., and Sekar, V. Making middleboxes someone else's problem: Network processing as a cloud service. *In Proc. SIGCOMM*, 2012.

[33] Smith, R., Estan, C., and Jha, S. XFA: Faster signature matching with extended automata. *In Proc. IEEE S&P'08*, 2008.

[34] Snort. http://www.snort.org.

[35] Spring, N., Mahajan, R., Wetherall, D., and Anderson, T. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, Feb. 2004.

[36] Teixeira, R., Shaikh, A., Griffin, T., and Rexford, J. Dynamics of hot-potato routing in IP networks. *In Proc. SIGMETRICS '04/Performance '04*, 2004.

[37] Vallentin, M., Sommer, R., Lee, J., Leres, C., Paxson, V., and Tierney, B. The NIDS cluster: scalable, stateful network intrusion detection on commodity hardware. *In Proc. RAID'07*, 2007.

[38] Vasiliadis, G., Polychronakis, M., Antonatos, S., Markatos, E. P., and Ioannidis, S. Regular expression matching on graphics hardware for intrusion detection. *In Proc. RAID '09*, 2009.

[39] Vasiliadis, G., Polychronakis, M., and Ioannidis, S. MIDeA: a multi-parallel intrusion detection architecture. *In Proc. CCS '11*, 2011.

[40] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, Dec. 2002.

[41] World intrusion detection and prevention markets. http://www-935.ibm.com/services/us/iss/pdf/esr_intrusion-detection-and-prevention-systems-markets.pdf.

[42] Yu, F., Lakshman, T. V., Motoyama, M. A., and Katz, R. H. SSA: a power and memory efficient scheme to multi-match packet classification. *In Proc. ANCS '05*, 2005.

[43] Zhang, Y. and Paxson, V. Detecting stepping stones. *In Proc. Usenix Security'00*, 2000.