

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 RAM⁷ 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-

⁷The 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.

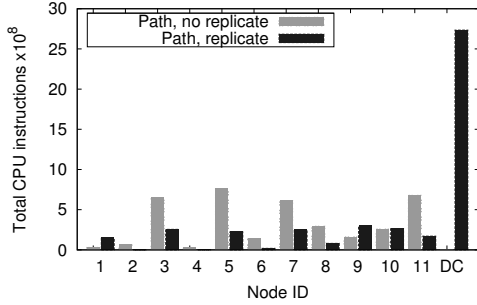


Figure 10: Maximum absolute CPU usage of each NIDS node in our Emulab experiment

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_n^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.

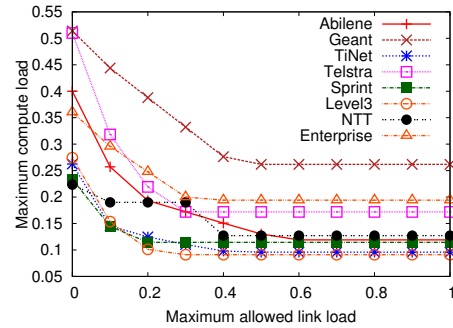


Figure 11: Varying $MaxLinkLoad$ with datacenter capacity = $10\times$.

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 $8\text{--}10\times$ 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

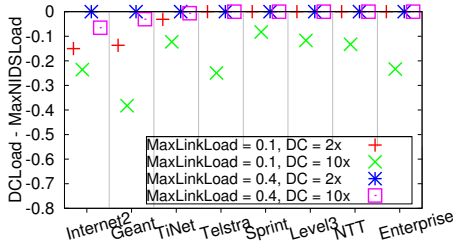


Figure 12: Comparing the compute load on the datacenter vs. maximum load on interior NIDS nodes.

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.

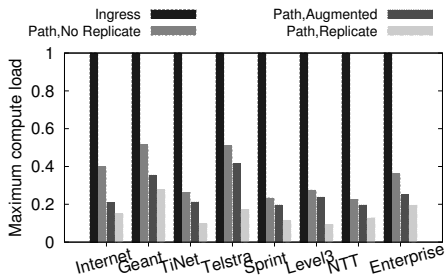


Figure 13: Maximum compute load across topologies with different NIDS architectures.

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.

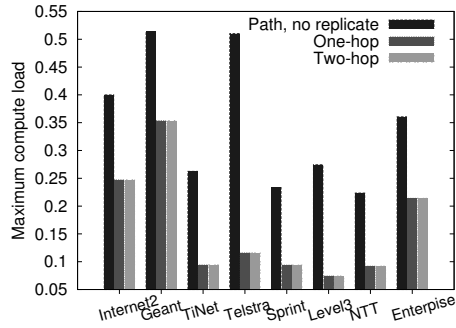


Figure 14: Local one- and two-hop replication.

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_j s) 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.

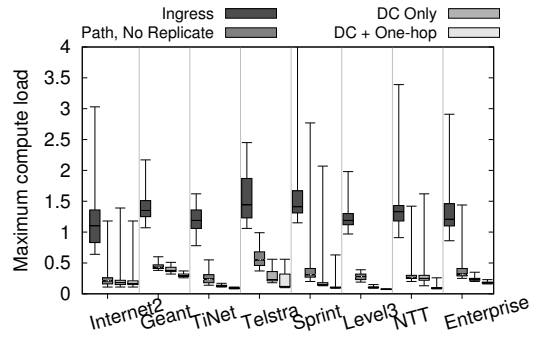


Figure 15: Comparison between NIDS architectures in the presence of traffic variability.

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 $= \theta$ and standard deviation $= \frac{\theta}{5}$), we find a path from this pre-computed 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.

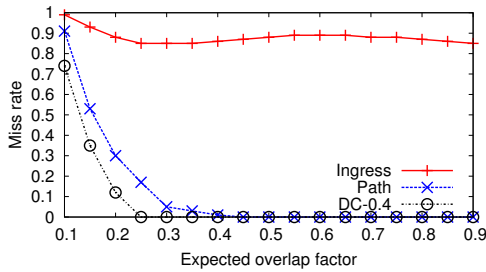


Figure 16: Detection miss rate vs. degree of overlap

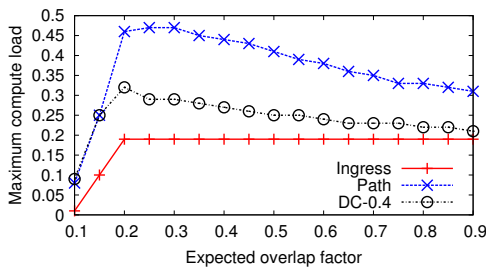


Figure 17: Maximum load vs. degree of overlap

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.

⁸The 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.

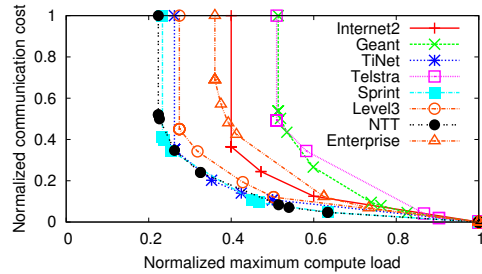


Figure 18: Tradeoff between the compute load and communication cost with aggregation as we vary β

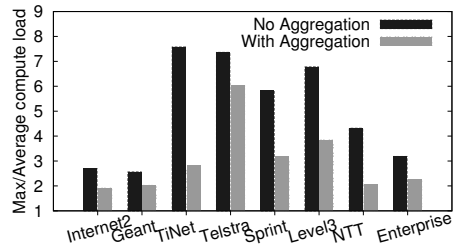


Figure 19: Ratio between maximum and average compute load with and without aggregation.

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_1 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/doors.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., Hjalmytsson, 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 traffic 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.