The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment

Vyas Sekar*, Sylvia Ratnasamy†, Michael K. Reiter⋆, Norbert Egi††, Guangyu Shi ††
* Intel Labs, † UC Berkeley, ⋆ UNC Chapel Hill, †† Huawei

ABSTRACT

Most network deployments respond to changing application, workload, and policy requirements via the deployment of specialized network appliances or "middleboxes". Despite the critical role that middleboxes play in introducing new network functionality, they have been surprisingly ignored in recent efforts for designing networks that are amenable to innovation. We make the case that enabling innovation in middleboxes is at least as important, if not more important, as that for traditional switches and routers. To this end, our vision is a world with software-centric middlebox implementations running on general-purpose hardware platforms that are managed via open and extensible management APIs. While these principles have been applied in other contexts, they introduce unique opportunities and challenges in the context of middleboxes that we highlight in this paper.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management; C.2.1 [**Network Architecture and Design**]: Centralized networks

General Terms

Design, Measurement, Management

Keywords

Middlebox, consolidation, network management

1. INTRODUCTION

A growing body of research focuses on designing networks that are amenable to innovation. Broadly, these pursue two complementary approaches. The first tackles the high cost, limited flexibility, and long development cycles typically associated with routers and switches. They propose alternative programmable device architectures using low-cost hardware such as x86 CPUs [20, 22, 10], GPUs [15], FPGAs [13] and merchant switch silicon [14]. The second tackles the limited flexibility caused by the narrow or closed interfaces found in network devices. These focus not so much on the internals of a device but on the abstractions that devices expose to management applications [24, 9, 23].

Together these efforts offer promising approaches to improve network extensibility and have received widespread attention. To date, however, the focus has been almost exclusively on traditional Layer 2/3 functions such as forwarding and routing. While these are no doubt fundamental tasks that merit attention, we argue that this focus overlooks a key reality in how network deployments evolve in response to changing application, workload, and policy requirements. In current networks, the de-facto approach to introduce new functionality is often not through modification to switches and routers, but rather is through the deployment of specialized network appliances or "middleboxes".

Modern networks deploy a range of middleboxes such as WAN optimizers, proxies, intrusion detection and prevention systems, network- and application-level firewalls, and application-specific gateways. Several studies report on the rapid growth of this market; the market for network security appliances alone was estimated to be 6 billion dollars in 2010 and expected to rise to 10 billion in 2016 [8]. In other words, middleboxes are critical part of today's networks and it is reasonable to expect that they will remain so for the foreseeable future.

It is troubling then that current middlebox architectures suffer to an equal degree from the same barriers to innovation as switches and routers. Today's middleboxes are typically expensive and closed systems, with little or no hooks and APIs for extension or experimentation. Each middlebox typically supports a narrow specialized function (e.g., IDS, WAN optimization) and is built on a particular choice of hardware platform. A further exacerbation is that middleboxes are acquired from independent vendors and deployed as standalone devices with little uniformity in their management APIs, or cohesiveness in how the ensemble of middleboxes is managed.

Given the above, we argue that the case for enabling innovation in middleboxes is just as important as that for switches

Hotnets '11, November 14–15, 2011, Cambridge, MA, USA.

and routers. Rather than take a philosophical stance on the merits of middleboxes, we propose to embrace deployment realities and focus instead on how we can architect middleboxes for extensibility. To this end, we present a research agenda that targets extensibility both at the level of a single middlebox, and the management of an ensemble of middleboxes. Specifically, we explore an approach that seeks extensibility through three strategies:

- *software-centric* implementations of middlebox applications that decouple middlebox hardware and software.
- the *consolidation* of multiple (software-based) middlebox applications on a shared hardware platform, and,
- *logically centralized management* with common APIs for a unified network-wide view in provisioning and managing middlebox deployments.

In a general context, the above strategies are not new. In fact, the first and last are inspired by the above-cited work on enabling innovation in switches and routers, building on similar reasoning regarding the benefits of standardized APIs, software-centric implementations and commodity hardware. Similarly, consolidation is commonly used to reduce cost and device sprawl in data centers.

However, applying these ideas to middleboxes raises unique challenges and opportunities because middleboxes differ from routers and switches on three fronts:

- *Heterogeneity:* Middleboxes cater to a wide range of requirements including security (e.g., IDS, firewall), performance acceleration (e.g., WAN optimizer), and supporting new applications (e.g., media gateways).
- *Deeper processing:* Middleboxes embed application-specific semantics and perform more complex per-packet processing (e.g., deep packet inspection, maintaining per-flow or per-session state).
- *Loose physical coupling:* Routers/switches process *every packet* they see, and each packet is processed at *every hop*. A middlebox, however, processes only a subset of packets pertinent to its application. Further, each middlebox function occurs at a small set (1 or 2) of the hops that a packet traverses.

For example, consolidating heterogeneous workloads on a shared platform raises new challenges for resource allocation and scheduling to retain the performance of standalone devices. At the same time, heterogeneity offers the opportunity to reduce hardware costs by leveraging *multiplexing* benefits arising from variability in the workload across different applications. Deeper processing raises the concern whether a general-purpose solution can provide high performance. Here, a software-centric solution also offers the possibility of *reuse* to avoid duplicating expensive lower-layer tasks such as session reconstruction or protocol parsing. Similarly, loosely coupled computation implies that configuring middleboxes to enforce high-level policies (e.g., all port 80 traffic goes through an IDS) requires more expressive interfaces compared to routing. But, loose coupling also provides an opportunity to *spatially distribute* middlebox applications to use spare resources elsewhere in the network.

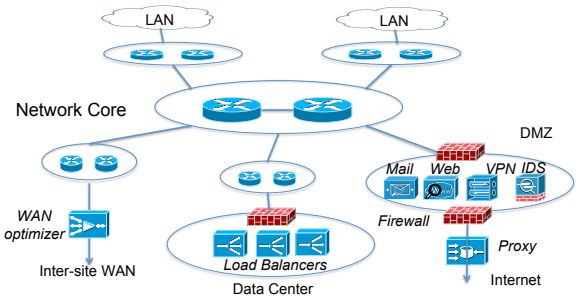

Figure 1: A typical site in the enterprise network

Our focus in this paper is on highlighting these challenges and opportunities and we leave it to future work to fully weigh the pros and cons of this vision.

Roadmap: We present evidence from a real enterprise network that motivates our proposed middlebox architecture in §2. We describe new opportunities for resource savings this architecture enables in §3; system challenges in §4 and finally conclude in §5. We discuss open questions and related work inline throughout the paper.

2. MOTIVATION

We begin with anecdotal evidence in support of our claim that middlebox deployments constitute a vital component in modern networks and the challenges that arise therein. Our observations are based on a study of middlebox deployment in a large enterprise network and discussions with the enterprise's network administrators.

The enterprise spans tens of sites across several geographical regions and serves more than 80K users. Each major site has a typical structure shown in Figure 1 with end hosts connected by a simple hierarchy of switches to a load balanced core of routers. Traffic to/from end-hosts leads to one of three destinations: enterprise hosts in other sites (WAN), external hosts on the public (Internet), or servers in local datacenters. Middleboxes are deployed along the path to each of these destinations; the figure shows the types of middleboxes in each segment. For example, we see that all inter-site traffic goes through a WAN optimizer and all public Internet traffic goes via a proxy. We also see many application gateways (*e.g.*, mail, web, VPN) in the DMZ connected to the public Internet.

Table 1 summarizes the types and numbers of different middleboxes in the enterprise. We see that the total number of middleboxes, is comparable to the number of routers! Middleboxes are thus a vital portion of the enterprise's network infrastructure. We further see a large diversity in the type of middleboxes; recent studies suggest similar diversity is found in ISP and datacenter networks as well [28, 30, 17].

The administrators indicated that middleboxes represent a significant fraction of their (network) capital expenses and expressed the belief that processing complexity necessitates *expensive* hardware capabilities which contributes to the high

Appliance type	Number
Firewalls	166
NIDS	127
Conferencing/Media gateways	110
Load balancers	67
Proxy caches	66
VPN devices	45
WAN optimizers	44
Voice gateways	11
Middleboxes total	636
Routers	≈ 900

Table 1: Devices in the enterprise network

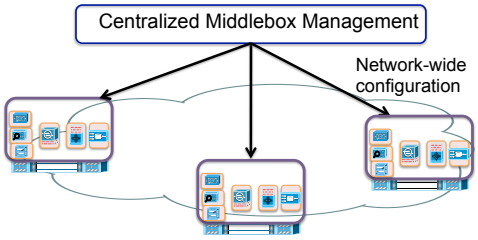

Figure 2: Each middlebox application runs as a software module on a consolidated platform with the ensemble managed by a central controller.

capital costs associated with such appliances. Two further nuggets emerged from these discussions. First, they revealed that each class of middleboxes is currently managed by a *dedicated team* of administrators; e.g., 2-3 administrators are assigned to manage just the WAN optimizer deployment, while a separate set manages all application gateways. This is in part because the enterprise uses different vendors for each application in Table 1; the vendor interaction and understanding required to manage and configure each class of middlebox leads to inefficient use of administrator expertise and significant operational expense. The lack of high-level network-wide management APIs further exacerbates the problem. For example, one administrator described how significant effort was required to manually tune what subset of traffic should be directed to the WAN optimizer to optimize the tradeoff between the bandwidth savings and appliance load. The second nugget of interest: several administrators voiced concern that market trends towards the "consumerization" of computing devices (e.g., smartphones, tablets) stands to increase the need for in-network capabilities [8]. Because middleboxes today lack *extensibility*, this would inevitably lead to further appliance sprawl, with a corresponding increase in capital and operating expenses.

Despite these concerns, the administrators reiterated the value they find in such appliances, particularly in supporting new applications (e.g., teleconferencing), increasing security (e.g., IDS), and improving performance (e.g., WAN optimizers).

An alternative architecture: We see that although middleboxes are a critical part of the network infrastructure, they are *expensive*, *closed* platforms that are *difficult to extend*, and *difficult to manage*. This motivates us to rethink how middleboxes are designed and managed. We envision an architecture in Figure 2 where *software-centric* implementations of middlebox applications are *consolidated* to run on a general-purpose shared hardware platform, managed in a *logically centralized* manner with uniform APIs for a network-wide view.

Software-based solutions reduce the cost and development cycles to build and deploy new middlebox applications (and independently argued in parallel work [12]); consolidating multiple applications on a single physical platform reduces device sprawl (with some early commercial offerings already emerging [7, 3]); and centralized management with uniform APIs simplify network-wide management (as has long been argued for routing and access control [19, 18, 9]).

We proceed to explore the unique opportunities and challenges that arise in combining these strategies and applying them in the context of middleboxes.

3. OPPORTUNITIES

While recent work shows that software implementations of network elements can deliver high performance (e.g., [20, 15, 14]), there is a concern that the extensible solutions we envision may be less resource efficient than today's specialized solutions. However, as we discuss next, our proposal introduces *new* efficiency opportunities that do not arise with today's middlebox deployments.

3.1 Application multiplexing

Consider the WAN optimizer and the IDS from Figure 1. The former optimizes file transfers between two enterprise sites and may see peak load at night when system backups are run. In contrast, the IDS may see peak load during the day because it monitors users' web traffic.

Suppose the volumes of traffic processed by the WAN optimizer and IDS at two time instants t_1, t_2 are $10, 50$ packets and $50, 10$ packets respectively. Today each application runs on separate hardware and each device must be provisioned to handle a peak load of $\max\{10, 50\} = 50$ packets. A consolidated middlebox with each application in software can flexibly allocate resources as the load varies. Thus, it needs to be provisioned to handle the peak *total* load of 60 packets or 40% fewer resources.

Measurement: Figure 3 shows a timeseries of the utilizations of four middleboxes at one enterprise site, each normalized by its maximum observed value. We see that the devices reach peak utilization at different times. If $NormUtil^t_{app}$ is the normalized utilization of the device app at time t, we compare the sum of the peak $\sum_{app} \max_t\{NormUtil^t_{app}\} = 4$, and the peak total $\max_t\{\sum_{app} NormUtil^t_{app}\} = 2.86$. Thus, in Figure 3, multiplexing will reduce the resource requirement $\frac{4-2.86}{4} = 28\%$.

3.2 Reusing software elements

Each middlebox needs to implement modules for packet capture, parsing headers, reconstructing flow/session state, and parsing application protocols. If the same traffic is processed by many applications (e.g., HTTP traffic is processed

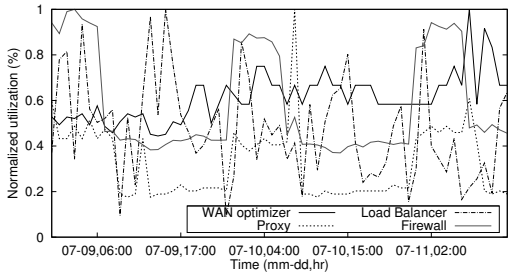

Figure 3: Middleboxes reach peak utilization levels at different times

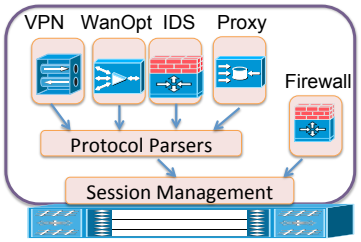

Figure 4: Reusing modules across applications

by the IDS, proxy, and a web firewall in Figure 1) each appliance repeats these common actions. Alternatively, we can *reuse* these building blocks across the middlebox applications as in Figure 4. That is, the processing logic and data structures for session reconstruction and HTTP parsing are shared across the IDS, proxy, and the web firewall. Note that this complements application multiplexing; multiplexing exploits temporal variability, whereas reuse helps when different applications act on the same packets.

Consider the IDS and proxy from Figure 1. Both reconstruct the session- and application-level state before running higher-level actions. Suppose each device needs 1 unit of processing per packet and that the common tasks contribute 50% of the processing. While both appliances process HTTP traffic, they may also process traffic unique to each context; e.g., IDS processes UDP traffic which the proxy ignores. Suppose there are 10 UDP packets and 45 HTTP packets. The total resource requirement in Figure 1 is $(IDS = 10 + 45) + (Proxy = 45) = 100$ units. The setup in Figure 4 avoids duplicating the common tasks for HTTP traffic and needs $45 * 0.5 = 22.5$ units or 22.5% fewer resources.

Measurement: The example shows we need to quantify: (1) the overlap in traffic processed by middleboxes and (2) the relative contribution of the reusable actions. First, to measure the traffic overlap, we obtain (public) configurations for Bro [26] and Snort [1] and the (private) configuration for a WAN optimizer. From these, we extract the port numbers of traffic that each processes. Then, we quantify the pairwise overlap between middleboxes using flow-level traces from Internet2. The overlap between M_1 and M_2 is the ratio of the volume of common traffic they process to the volume of traffic that at least one processes, i.e., $\frac{|M_1 \cap M_2|}{|M_1 \cup M_2|}$. Across all pairs of middleboxes and traces, this overlap is $64 - 99\%$.

Next, to quantify the relative contribution of the common

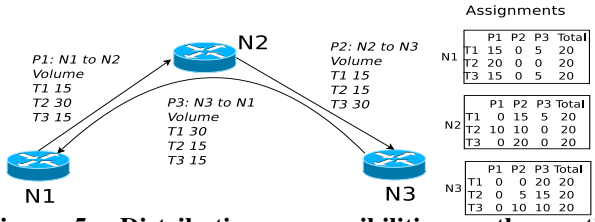

Figure 5: Distributing responsibilities as the spatial structure of the traffic changes over time

primitives, we use trace-driven benchmarks with Bro [26].[1] Bro uses a common session management layer and application-specific modules are built atop this layer. For HTTP traffic, this common layer contributes 18% and 54% of the CPU and memory footprint.

3.3 Spatial distribution

Centralizing middlebox management also provides a network-wide view to spatially distribute middlebox tasks to further reduce resource costs. Consider the topology in Figure 5 with three nodes N1–N3 and three end-to-end paths P1–P3. The traffic on these paths peaks to 30 packets at different times as shown.

Suppose we want all traffic to be monitored by IDSes. The default deployment is an IDS at each *ingress* N1, N2, and N3 for monitoring traffic on P1, P2, and P3 respectively. Each such IDS needs to be provisioned to handle the peak volume of 30 units with a total network-wide cost of 90 units. With a network-wide view, however, we can potentially *distribute* the IDS responsibilities by allowing each IDS at N1–N3 to monitor some fraction of the traffic on the paths traversing it (e.g., [29]). (Here, we assume that IDSes are "on-path" or their upstream routers redirect packets to them [11].) In this case, we can provision each IDS to handle 20 packets and yet provide full monitoring coverage. For example, at time T1, N1 uses 15 units for P1 and 5 for P3; N2 uses 15 units for P2 and 5 P3; and N3 devotes all 20 units to P3. (We can generate similar configurations for the other times as shown.) Thus, distribution reduces the total provisioning cost $\frac{90-60}{90} = 33\%$ compared to an ingress-only deployment. Again, note that this is orthogonal to application multiplexing and software reuse.

Measurement: To quantify the benefit of spatial distribution, we use the topology and two weeks of per-5-minute traffic matrices from the enterprise network (site-level) and Internet2 (PoP-level). Let V_i be the volume of traffic that each IDS N_i is provisioned to handle, such that this configuration can fully monitor the traffic for each 5-minute interval over the two-week period. We find that spatial distribution reduces the total provisioning cost, $\sum_i V_i$, 33% for Internet2 and 55% for the enterprise network.

4. DESIGN CHALLENGES

Next, we discuss the system challenges in realizing the architecture and opportunities described in the previous sec-

[1]We are not aware of vendors with reusable software modules and data on their software design is hard to obtain.

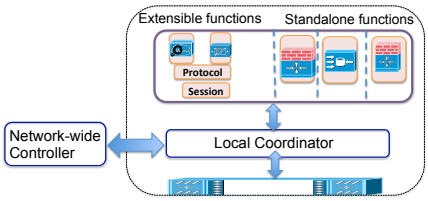

Figure 6: The different components and interfaces in our proposed middlebox architecture

tions. We envision the system architecture shown in Figure 6. A centralized *controller* determines how traffic should be processed by each middlebox to achieve network-wide objectives. Each middlebox implements a local *coordinator* responsible for resource management and executing the controller's decisions.

4.1 Network-wide challenges

The controller takes as input: (i) the network topology, (ii) the network-wide traffic workload in the form of a traffic matrix for different classes of traffic, (iii) for each middlebox, a description of its hardware resources such as CPU, memory, presence/absence of specialized hardware (*e.g.*, accelerators for crypto or reg-ex operations [21, 4, 2]) and, (iv) for each middlebox application A_i, a policy specification that describes the application's network-wide objectives and constraints.

The controller solves an optimization problem that determines how the processing should be split across middleboxes to minimize network-wide resource consumption, while respecting resource limits and policy requirements. The controller's output is a set of per-middlebox configurations. This raises three natural questions.

Q: What constitutes the policy specifications?

We envisage three high-level components here:
(1) A description of *what* traffic the application acts on—*e.g.*, 'all port 80 traffic', 'all traffic between border routers A and B', 'all traffic to prefix p', and so forth.
(2) There are natural dependencies in the *order* in which middlebox applications should be applied. For example, an IDS must inspect payloads before they are modified by compression or encryption. These can be expressed as pairwise *precedence* constraints $A_i \prec A_j$ between the applications [16].
(3) The application should convey what specialized hardware it requires. Some applications may be written to assume an accelerator (*e.g.*, a DPI ASIC for an exfiltration engine [4]) while others may opportunistically use such capabilities (*e.g.*, special encryption instructions in some CPUs [21]). Thus, each requirement also specifies if it is strict or loose.

Q: Is the optimization tractable?

Traditional optimization in monitoring and traffic engineering treats different network functions in isolation (e.g., [27]). In our context, reuse and policy dependencies between middlebox applications means that these models no longer apply. In theory, such dependencies can be captured with fine-grained models that track each action on a packet as it traverses the network; i.e., does middlebox M_k run application A_i on packet p. However, such discrete models are computationally intractable. The challenge here is to find a practical *relaxation* that it is both tractable and near-optimal.

Q: What is the controller →coordinator interface?

The controller communicates the per-middlebox configuration output by the global optimization to each middlebox. Conceptually, this is a set of rules, each mapping from a *packet filter* to an ordering of middlebox functions. These packet filters identify both a class of traffic (defined on packet header fields as combinations of source/destination IP prefixes and port ranges) and what fraction of this traffic class this middlebox should process. Each middlebox can then use techniques such as hash-based sampling to process the appropriate fraction of traffic while still ensuring a particular flow/session is pinned to the same node [29]. For example, a configuration of the form:

$$SrcIP = *, DstIP = *, Port = 80, Fraction \in [0, 0.5]$$
$$Firewall \prec Proxy; IDS \prec Proxy; Firewall \prec FlowMon$$

specifies that packets on port 80 with the hash of the session identifier in the range $[0, 0.5]$ should be processed by the firewall, IDS, proxy, and a flow-level monitor. Note that ordering specifies a *partial order* as there are no dependencies between firewall-IDS and IDS-Flowmon.

Even this simple example highlights that middlebox management requires greater expressiveness compared to current interfaces such as OpenFlow [24]. For example, the "action" on a packet is not a simple forward/drop but specifies a *partial order* of functions to be applied.

Different deployments (small vs. large enterprise, ISP, datacenter) may also vary in what this management interface needs to be. For example, spatial distribution may not apply to a small enterprise with 1-2 locations. An open question in this regard is whether our proposed configuration parameters are either complete or minimal.

4.2 Middlebox architecture

Figure 6 shows the three components in each middlebox platform. Between the hardware and the applications implemented in software, we introduce a 'local coordinator' that interacts with the network-wide controller and is responsible for two tasks. First, it *steers* packets between applications. Recall that each packet may be processed by multiple applications with a partial precedence order between them. Since the applications could be obtained from different vendors and unaware of each other, the coordinator is responsible for steering a packet between applications and eventually forwarding the packet. Second, the coordinator implements the resource allocation and scheduling required to efficiently multiplex different applications in a shared platform.
This raises many system implementation challenges:

Q: What is the appropriate hardware platform?

A platform of multiple general-purpose cores (x86+GPU) is a good baseline since this offers portability, ease of development, and is already a popular platform of choice [5].

At the same time, we do not want to mandate this choice since it is unclear that general-purpose cores will suffice; *e.g.* for DPI or encryption. Moreover, hardware vendors will also want to offer innovative, differentiated solutions. Rather than completely custom hardware platforms for complex applications, we aim for one in which the general-purpose cores are augmented with a small set of specialized hardware functions. This leads to many sub-questions, including: What is a modular hardware design that allows us to incrementally add accelerators? Is there a small set of accelerators that can meet most application demands? How are these accelerators best exposed to the higher-layer software?

A second form of specialized hardware is for high-speed packet classification. Prior work [20] shows that achieving high rate packet forwarding requires NIC support for packet classification and multiple hardware queues. Again, hardware vendors might choose to innovate here; *e.g.*, offering TCAM-based classification, or larger numbers of queues. This again raises several sub-questions, including: Is specialized hardware for classification required? What is the right hardware design that offers flexible classification at high speed and low cost? What API should such classification engines expose to the coordinator?

Q: How do we parallelize the coordinator?

The coordinator itself must be parallelized to run on multiple cores so that it does not become a processing bottleneck.

Q: How are resources scheduled across applications?

Prior work [20, 15, 14] describes approaches to parallelizing traffic processing across multiple cores. However they consider homogeneous workloads (e.g., routing) where the per-packet resource consumption is uniform. These assumptions may not hold in a consolidated middlebox and hence we need solutions to adaptively schedule resources across a set of heterogeneous applications.

Q: How do we refactor middlebox applications for reuse?

Today vendors sell *monolithic* solutions that tightly couple both hardware and software. A first step toward extensibility is to decouple the hardware and software; some vendors already offer 'unbundled' software-only versions (e.g., [6]). A further step is a *modular* software architecture, allowing reuse of common low-level primitives as in §3.2. Vendors may prefer to supply standalone or modular application implementations, and hence we envision the platform supporting both. This also raises the question of identifying primitives that simplify application development and provide sufficient reuse. As a starting point, modules for a few common tasks such as packet capture, session reconstruction, and application protocol parsing will be useful; these occur in most middleboxes and have non-trivial resource footprints, but are hard to implement correctly and efficiently [25, 26].

Q: What are minimal APIs between the components?

We expect vendors for each component to innovate independently and to differentiate. Thus, we need a careful design of inter-component APIs that are not overly restrictive but also not complex. While we have discussed the information needed across the boundaries, distilling it into a concrete and minimal set of APIs remains open.

5. CONCLUSIONS

Middleboxes are a vital component of modern networks but have been M.I.A in recent efforts on architecting networks for innovation. We propose tackling this elephant in the room through a consolidated, software-centric architecture and highlight the unique opportunities and challenges that arise in a middlebox context. We are currently working to implement and evaluate such an architecture.

Acknowledgments

We thank Neil Doran, Patrick Egan, Sridhar Mahankali, Sanjay Rungta, Daniel Tang, and Rob Wilson for sharing their insights and feedback. This work was funded in part by ONR grant N000141010155 and by NSF grants 0831245 and 1040626.

6. REFERENCES

[1] `http://www.snort.org`.
[2] Cavium networks. `http://www.caviumnetworks.com/`.
[3] Crossbeam network consolidation. `http://bit.ly/qlotDK`.
[4] Palo alto networks. `http://www.paloaltonetworks.com/`.
[5] Riverbed Networks: WAN Optimization. `http://www.riverbed.com/solutions/optimize/`.
[6] Silver Peak software WAN optimization. `http://bit.ly/nCBRst`.
[7] Untangle. `www.untangle.com`.
[8] World enterprise network security markets. `http://bit.ly/gYW4Us`.
[9] A. Greenberg et al. A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM CCR*, 35(5), Oct. 2005.
[10] A. Greenlagh et al. Flow Processing and the Rise of Commodity Network Hardware. *ACM CCR*, Apr. 2009.
[11] A. Shieh et al. SideCar: Building Programmable Datacenter Networks without Programmable Switches. In *Proc. HotNets*, 2010.
[12] J. Anderson and A. Vahdat. xOMB: eXtensible Open MiddleBoxes. Unpublished Manuscript.
[13] B. Anwer et al. Switchblade: A platform for rapid deployment of network protocols on programmable hardware. In *SIGCOMM*, 2010.
[14] G. Lu et al. ServerSwitch: A Programmable and High Performance Platform for Data Center Networks. In *Proc. NSDI*, 2011.
[15] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-Accelerated Software Router. In *Proc. SIGCOMM*, 2010.
[16] D. Joseph and I. Stoica. Modeling middleboxes. *IEEE Network*, 2008.
[17] D. A. Joseph, A. Tavakoli, and I. Stoica. A Policy-aware Switching Layer for Data Centers. In *Proc. SIGCOMM*, 2008.
[18] M. Caesar et al. Design and implementation of a Routing Control Platform. In *Proc. of NSDI*, 2005.
[19] M. Casado et al. SANE: A Protection Architecture for Enterprise Networks. In *USENIX Security*, 2006.
[20] M. Dobrescu et al. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proc. SOSP*, 2009.
[21] M. Kounavis et al. Encrypting the Internet. In *Proc. SIGCOMM*, 2010.
[22] N. Egi et al. Towards high performance virtual routers on commodity hardware. In *Proc. CoNEXT*, 2008.
[23] N. Gude et al. NOX: Towards an Operating System for Networks. *ACM SIGCOMM CCR*, July 2008.
[24] N. McKeown et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM CCR*, 38(2), Apr. 2008.
[25] R. Pang, V. Paxson, R. Sommer, and L. Peterson. binpac: A yacc for Writing Application Protocol Parsers. In *Proc. IMC*, 2006.
[26] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proc. USENIX Security Symposium*, 1998.
[27] M. Roughan. Robust network planning. Chapter 5, Guide to Reliable Internet Services and Applications.
[28] T. Benson et al. Demystifying configuration challenges and trade-offs in network-based isp services. In *Proc. SIGCOMM*, 2011.
[29] V. Sekar et al. cSamp: A System for Network-Wide Flow Monitoring. In *Proc. of NSDI*, 2008.
[30] Z. Wang, Z. Qian, Q. Xu, Z. M. Mao, and M. Zhang. An Untold Story of Middleboxes in Cellular Networks. In *Proc. SIGCOMM*, 2011.