Toward Software-Defined Middlebox Networking

Aaron Gember, Prathmesh Prabhu, Zainab Ghadiyali, Aditya Akella
University of Wisconsin-Madison, Madison, WI, USA
{agember,pprabhu,zainab,akella}@cs.wisc.edu

ABSTRACT

Current middlebox (MB) management mechanisms are clumsy and unsuitable for taking full advantage of new MB deployment models and diverse MB functionality. Instead, we advocate for mechanisms that help exercise unified control over the key factors influencing MB operations. Our goal is to realize a *software-defined MB networking* framework to simplify management of complex, diverse functionalities and engender rich deployments. We discuss the major challenges that arise—representing, manipulating, and knowledgeably controlling MB state—and we present initial thoughts on the appropriate abstractions and interfaces to address them.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*

General Terms

Design, Management, Standardization

1. INTRODUCTION

Middleboxes (MBs) are a crucial part of many enterprise LANs, data centers, and clouds, enabling enterprises to ensure security, improve performance, and meet other sophisticated goals. MBs fill a unique and important role in the network: unlike networking equipment (e.g., switches), MBs do more than just routing[1], offering a variety of innovative functions. Yet, MBs are not as general as application servers, as MBs focus solely on examining and modifying traffic.

Recently, several new trends in MB deployment have arisen. First, SDN has enabled MBs to be deployed at arbitrary locations in LANs and data centers [2], no longer limiting placement to network choke points. In some cases, MBs can even be implemented within the SDN itself (e.g., changing IPs/ports like a NAT); although, the lack of support for more complex MBs, such as those performing deep-packet modification (e.g., WAN optimizers), means many MBs will remain separate. Second, while MBs used to be deployed as physical appliances or dedicated servers, MBs are now being deployed in a variety of additional forms: as VMs, in hypervisors, on arbitrary end-hosts [4], and as collections of processes [9]. Both of these trends lend themselves towards a more dynamic model of MB deployment, e.g., launching new MB VMs when network load is high [5].

Several factors influencing MB operations are crucial to manage in such a dynamic setting. MBs must receive the correct traffic, e.g., all inbound packets, and be configured with the correct policies, e.g., firewall accept/drop rules, to provide the intended enhancements. Additionally, changes in a deployment, e.g., switching the MB a flow traverses due to MB scaling, requires careful control over the data maintained by MBs' internal logic, e.g, the state of a TCP connection, to ensure correct MB behavior. Ignoring one or more of these aspects could have drastic consequences.

Today, each of these factors influencing the operations of a dynamic MB deployment—configuration, traffic flow, and internal data—is managed through ad hoc point mechanisms, including by-hand tweaking. This clumsy management approach primarily results from the diversity of MBs in enterprises [9] and the unique configuration and tuning each requires. Moreover, this approach makes it difficult to leverage new MB deployment models and diverse MB functionality to enrich application deployments in sophisticated ways (see examples in §2.1). Mechanisms that help exercise unified control over all the factors can rectify this key issue.

Currently, no framework for unified control exists. SDN provides a unified approach to control plane management focusing mainly on controlling traffic flow [6, 8]. Frameworks like Stratos [5] address the ordering of MBs and load distribution. However, little progress has been made with regard to controlling the state—internal data, configured policies, etc.—associated with MBs (SIMCO [1] is the closest).

Our ultimate goal is to realize *an SDN-like framework for*

[1]Modern switches tend to be powerful enough to also fulfill MB-like roles, but this is not the equipment's primary purpose.

Hotnets '12, October 29–30, 2012, Seattle, WA, USA.

MB management. Akin to how SDN has vastly simplified control plane management and led to a variety of innovative new applications for network control, a *software-defined MB networking* framework that facilitates unified control can similarly simplify management of complex, diverse functionalities and engender rich, new applications. The key issues for MBs, however, are that it is not clear what unified control entails, to what extent it is possible, and how to achieve it.

In this paper, we examine in depth the issues relating to a software-defined MB networking framework. We present several dynamic scenarios, and classify the MB state involved, to illustrate the major challenges that we believe need to be addressed: *representing, manipulating, and knowledgeably controlling MB state* (§2). We first explore the challenges in representing MB state, proposing an abstraction based on the inherent mapping of state to protocol header fields as a mechanism for dealing with internal, shared, and diverse state (§3). Next, we discuss the issue of state manipulation, highlighting the need for a broad interface that enables manipulation of most MB state while keeping a sufficient level of control in the hands of MBs themselves; we argue that a properly detailed event abstraction is a necessary complement, allowing MBs to expose their internal state changes and request missing or updated state (§4). Finally, we describe how specific control logics might utilize these interfaces and abstractions to achieve sophisticated control over MB deployments (§5).

2. BACKGROUND AND CHALLENGES

2.1 Illustrative Examples

We begin by outlining a few scenarios where the need for unified control of the state associated with MBs is evident:

Virtual Machine (VM) Provisioning. Enterprises have specific security and performance requirements for each application server. Traditionally, when a server is deployed, MBs are statically configured to meet these requirements. As data centers become entirely VM-based, server changes occur more frequently and MB management becomes a major burden. Thus, MB control should happen automatically and in concert with VM provisioning and migration: instantiating new server-specific policies, e.g., firewall rules; migrating MB state to ensure consistent, correct examination and modification of server traffic; verifying existing MB policies will not conflict with server requirements.

MB Scaling. The traffic load imposed on MBs will change over time, especially in cloud environments, motivating dynamic provisioning of MBs. While instantiating the necessary compute resources and network forwarding paths is important [5], ensuring proper handling of MB state is also crucial. MB state must be controlled at fine granularity[2] to allow rebalancing of load during scale up—e.g., policies must be replicated or split—and merging of MBs during scale down—e.g., per-flow state must be aggregated.

Middlebox	State
Firewall	Rules; Connection records {IP-tuple, seq #s, status}
NAT	Mappings {timer, internal IP-tuple, external IP-tuple}; Port forwarding rules; Timer duration; External address
Load Balancer	Mappings {timer, destination, IP-tuple}; Destinations; Balancing algorithm; Load measurements; Granularity
Redundancy Elimination	Chunk cache; Fingerprinting algorithm; Cache size; Replacement policy
Intrusion Prevention	Connection records {timer, IP-tuple, status, payloads, seq #}; Rules; Pattern-matching algorithm; Alert level

Table 1: State associated with several common types of MBs (*IP-tuple* includes addresses, ports, and protocol)

Live Network Migration. The virtualization of compute resources and networks, along with new MB deployment models, makes live migration of an entire application deployment possible. Such migrations require seamless MB operations and consistent MB performance throughout the migration (e.g., from a private data center to a public cloud). Some migrations may necessitate complex MB changes: e.g., a physical MB appliance may be split into several MB VMs or the entire network topology may change. These all require carefully dividing and migrating MB policies and internal data at the right time and between the right MBs.

Current ad hoc point mechanisms for managing dynamic MB deployments make supporting these scenarios complex and, in some cases, impossible. Moreover, deep, flexible control of layer 3-7 network services is a crucial component of an entirely software-defined data center.

2.2 Classification of MB state

Before delving further into the problem of unified control, we describe MB state in more depth.

MBs rely on complex and diverse state for proper operation. A single MB may receive dozens of "configuration" inputs, and its internal logic may maintain detailed records for thousands of flows. Moreover, the state a MB requires for operation varies significantly across MB types (and vendors), as shown in Table 1. In contrast, a network depends only on the forwarding information bases (FIBs) for proper traffic forwarding, which, in SDNs, is created and managed by a controller running SDN applications. Note that MB state will continue to increase in diversity and complexity as new MB functionality emerges and competing MB vendors try to specialize their offerings.

Pieces of MB state can be classified along several possible dimensions—source, structure, fluidity, etc.—but we believe the most salient classification is based on the role the state plays in a MB's operation. Accordingly, we divide MB state into four *classes*, listed and defined in Table 2. Each class of state has several *properties*: Some MB state is provided as *external* "configuration" input, while other state is created and manipulated by a MB's *internal* logic during operation. Some MB state pertains to *specific flows*[3], while other state

[2]Wrapping each MB, and its state, in a VM limits our choice of deployment models and the extent to which scaling is beneficial.

[3]Flow may refer to a transport connection, an application session, a source/destination pair, or any another traffic subset.

is *shared* by all, or an unknown subset of, traffic. Finally, for a specific type of MB, some classes of state have *many* possible structures and meanings while others have *few*.

We argue that it is crucial, and possible, to control two of the four classes—*action* and *supporting*—since both are critical to a MB's operation; hereafter *MB state* refers to these two classes. We don't consider designing control over *tuning* state because: (*i*) A MB could rely on a default or simpler form of tuning state—albeit at the cost of efficiency and performance—and still perform its basic operations correctly. (*ii*) Tuning state tends to be more fine-grained, vendor-specific (i.e., has many forms), and consistent over long timescales, making it more appropriate to manage this state out-of-band. Similarly, we eschew *monitoring* state because it's primarily intended for observing and tweaking MB behavior.

2.3 Three Challenges

Given an understanding of what is amenable to unified control, we argue that there are three major challenges that must be addressed in achieving this control and moving towards a model of software-defined MB networking.

1. How do you view and interpret MB state?
2. How do you manipulate what state exists and where?
3. How do you make informed state control decisions?

While these challenges are similar to those faced by SDN, addressing these challenges for MBs is harder due to the diversity and complexity of MBs. We discuss these issues in more depth in the sections that follow, and we present initial thoughts on the appropriate abstractions and interfaces that help address them. Additional challenges that require further research are discussed in §7.

3. STATE REPRESENTATION

We believe that viewing and interpreting MB state is one of the foremost issues in unified control: we need to know *what* we are controlling *before* we try to control it. However, there are several challenges in doing so:

- Some state is established and manipulated by a MB's internal control logic, causing the structure of this state to only be known to the MBs themselves. Such concealment arises from both vendors' desire to protect intellectual property and the disregard for this state in ad hoc MB management.
- Some state is shared by all (or an unknown subset of) flows and impacts the MB's operations on all of these flows. This conflicts with the per-flow operations common in applications and the network.
- There is high diversity in the structure and semantics of state across several types of MBs. This diversity leads to complexity if not properly contained.

We believe these challenges can be addressed by exposing the right abstraction for viewing and interpreting MB state. The abstraction should hide the intricacies of individual MB offerings while still allowing for specialization between MB types and vendors.

Source	Destination	Proto	Other	Action
10.10.0.0/16	*	TCP	dport 22	ACCEPT
*	10.20.1.0/24	TCP	dport 80	ACCEPT
*	*	TCP	state ESTABLISHED	ACCEPT

(a) Rules (Action)

SrcIP	DstIP	Proto	SPort	DPort	State
10.10.54.51	10.20.1.23	TCP	12983	22	ESTABLISHED
10.10.12.37	192.168.0.2	TCP	25483	22	SYN RCVD
192.168.0.1	10.20.1.73	TCP	52342	80	ESTABLISHED

(b) Connection Records (Supporting)

Table 3: Action and supporting state for a firewall

Fortunately, an inherent commonality exists in the operations of many MBs that can aid the formation of such an abstraction: many MB operations are a function of the values in packet headers. For example, a stateful firewall creates and updates connection records for each flow based on the values in the network- and transport-layer headers of received packets (Table 3b). Likewise, the decision to accept or drop a packet is based on these header values[4] (Table 3a). These are akin to SDN's use of protocol header fields to define forwarding behaviors.

3.1 View Abstraction

We propose an abstraction that leverages this *inherent mapping of state to protocol header values* to provide a myopic view of the complex and detailed state associated with a MB. The view abstraction (Figure 1) uses pertinent protocol header fields as a *key* for each distinct chunk of *action* and/or *supporting* state. Figures 2 and 3 show examples of state encoded using this abstraction.

The *key* is carefully constructed to identify exactly the traffic subsets to which a piece of state applies. A basic 5-tuple (source/destination network addresses, transport protocol, and source/destination transport ports) is a sufficient key for most state, but the protocol fields that form the key are flexible. Defining keys in this way enables us to leverage the ubiquity of common protocols to counteract MB diversity. Moreover, it provides an inherent hierarchy (e.g., IP addresses are hierarchical by design) for identifying subsets and supersets of state.

Action state is more challenging to represent because of its intrinsic tie to MBs' internal logic. For example, action state for firewalls is accept/drop rules, while action state for NATs is a mapping from public addresses/ports to private addresses/ports. We could represent action state as a binary blob. However, this would make creation and manipulation of action state complex because of the need for a deep understanding of MB semantics for the target MB. Instead, we propose to represent action state as a *transformation function* that changes specific packet header fields to

[4]Along with extra values calculated or inferred by the MB, e.g., TCP connection state.

Class	Definition	Intrusion Prevention (IPS) Example	Internal/ External	Shared/ Per-Flow	Many/ Few Forms	Should Manage
Action	Defines operations to apply to packets/flows	Rules	Both	?	Few	Yes
Supporting	Helps decide between multiple possible actions	Connection Records	Internal	Both	Many	Yes
Tuning	Tunes MB algorithms for performance, efficiency, etc.	Alert level	External	Shared	Many	No
Monitoring	Quantifies MB operations	Packet counters	Internal	Both	Many	No

Table 2: Classes of MB state, and its properties

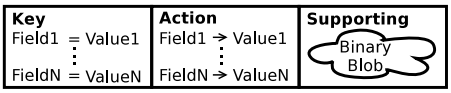


Figure 1: Abstract view of MB state

new constants (or discards the packet). This more generic representation can encode most of the actions of firewalls, NATs, load balancers, and (partially) IPSs, which are among the most common MBs [9]. MBs with different operational semantics require alternative representations of action state, which we leave for future investigation.

Supporting state is always represented as a *binary blob*, since its structure depends on a MB's deep internal logic. In special cases, where the structure of the binary blob is known, the blob may be analyzed by logic external to the MB, but the blob should never be changed by external logic to avoid inducing unintended MB behavior.

Note that by basing our view abstraction primarily on protocol header fields, we provide a mechanism for encoding only per-flow MB state. Shared MB state is much more challenging to represent because it is unclear how this state might be controlled. For example, the cache on a redundancy elimination (RE) MB is shared across all flows and synchronized between source and destination RE MBs; it is unclear how an RE MB's cache should be changed to ensure redundancy is correctly removed and restored when a flow is re-routed through a different RE MB. Therefore, our current state abstraction exposes *all shared state as a single binary blob*.

Takeaways. The internal or shared nature of some MB state makes representing MB state challenging. A view abstraction provides a uniform and well-structured representation of diverse MB state for the purposes of examining, defining, and facilitating migration of data that influences MBs' behavior. It does not entirely solve the challenge of how to manipulate what MB state exists and where; we address this challenge in the next section.

4. STATE MANIPULATION

Influencing a MB's behavior requires manipulating the state residing at the MB.[5] Today, this manipulation can only occur through narrow, MB-specific configuration interfaces. Moreover, these interfaces exclude significant subsets of state that are established and leveraged by the MB's internal logic. This limited interface severely constrains the flexibility and potential sophistication of a MB deployment: e.g., a flow's packets must traverse the same IPS for the duration of the flow because a record of the connection, required for proper attack detection, cannot be moved between IPSs. We argue that MBs should expose a broad state manipulation interface, complementary to our view abstraction. Unfortunately, this is complicated by several factors:

- Manipulation of some MB state is at the discretion of the MBs themselves: e.g., an IPS may establish a drop rule for flows believed to be malicious. This contrasts with SDNs where all state[6] is established and manipulated by the SDN controller.
- The state manipulation required to achieve a desired MB behavior varies significantly by MB type, and in some cases the objective behind the manipulation: e.g., scaling a firewall requires an interface for migrating connection records from an existing firewall and installing rules to apply to packets; scaling a load balancer requires transferring a partial list of potential destination servers. This is in contrast to SDNs, where state manipulation is restricted to forwarding entries.

These factors highlight the importance of carefully harmonizing a MB's internal manipulation and use of state with the external interfaces provided for state manipulation.

One extreme point in the design space is to create and modify *all* MB state externally. With this approach, the bulk of MBs' internal logics are reimplemented by a controller application, and MBs become nothing more than "dumb" packet modifiers, akin to SDN switches being "dumb" packet forwarders. While this very SDN-like approach provides significant flexibility, we believe it *removes too much control* from the MBs themselves, thereby *constraining* innovation.

Instead, we believe that state manipulation interfaces on MBs should be multi-faceted. First, MBs should expose a generic interface, which builds on our view abstraction, for externally accessing and updating diverse pieces of state. Second, MBs should announce internal state changes and be able to request external state changes. We present our initial thoughts on these interfaces in this section. Decisions on what the state should contain, when it should be changed, and where it should reside are best addressed by scenario-specific control logic (§5) which leverages these interfaces.

4.1 Operations

We propose a broad, generic interface, complementary to our view abstraction, for manipulating the state residing at MBs. A broad interface accommodates MB diversity and sophisticated MB control without introducing undue complexity. In contrast, narrow, state-specific interfaces, e.g., a

[5] A MBs's behavior can also be influenced through changes in its internal logic, but we assume this is fixed.

[6] With the exception of state like flow counters and timers.

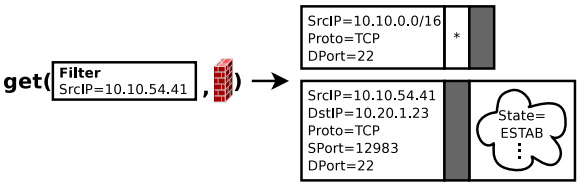
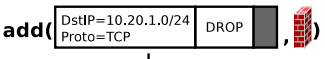


Figure 2: *get* operation applied to firewall



Source	Destination	Proto	Other	Action
*	10.20.1.0/24	TCP	*	DROP

Figure 3: *add* operation applied to firewall

firewall with one interface for updating rules and another for changing connection records, provide none of these benefits.

We believe three basic operations are sufficient:

- **get(filter, MB)** – Obtains from a MB all chunks of state whose key matches the filter and encodes the state using our view abstraction. Figure 2 shows an example *get* operation applied to the firewall state depicted in Table 3. One firewall rule applies to all traffic matching the filter, so one piece of action state is returned; one matching connection record is also returned.
- **add(state, MB)** – Adds a chunk of state, encoded using our view abstraction, to the MB. Figure 3 shows an example *add* operation that instantiates a firewall rule to drop traffic for a specific subnet.
- **remove(filter, MB)** – Removes from the MB all chunks of state whose key matches the filter.

A downside of these operations is their failure to communicate what types of state are available and required at a specific MB. For example, if a get operation applied to a firewall returns no action state, the control logic must know that some state (i.e., a rule) must be added for proper MB operation; in contrast, a NAT will automatically create an address/port mapping for new flows, so state need not preexist. This issue does not arise in SDNs because the state required is always the same: a matching forwarding entry for each flow traversing a switch.

An additional downside of this generic interface is the *potential for invalid manipulations* of MB state. For example, an outdated connection record could be obtained from one IPS and added to another; the second IPS will operate under the assumption that the connection record accurately portrays the full connection history, which it does not, allowing attacks to occur undetected. Methods for detecting such invalid state manipulations requires further research.

MBs' internal logic must be carefully enhanced to support the state manipulation operations proposed above without introducing consistency or performance issues. Add/remove operations may require manipulated, or related, state to be merged, split, or transformed in some other complex way. At a minimum, MBs need additional logic to translate between our view abstraction and their own internal structures.

4.2 Events

We believe that the operations discussed above must be complemented by an *event abstraction* to guide external state manipulation. This abstraction should expose the two types of interactions that occur between a MB's internal logic and state: (*i*) the MB's internal logic establishes or manipulates state at the MB; or (*ii*) the MB's internal logic reaches an operation that requires a piece of state. In SDNs, only the latter form of interaction occurs at network elements and is exposed via events.

The same diversity issues that plague the abstractions discussed thus far also make an event abstraction challenging to design. In SDNs, a few common events are raised by all networking equipment; MBs, in contrast, are highly specialized. We borrow ideas from our view abstraction (§3.1) and define the *scope* of an event based on protocol header fields. However, the structure used to convey the type, parameters, and semantics of an event requires further research.

One of the most difficult aspects in designing an event abstraction is exposing the right level of introspection into MBs' operations. Several distinct events could be raised during the processing of even a single packet: e.g., received first packet of flow, created connection record, updated state of TCP connection, etc. Exposing all these events could quickly overwhelm a controller. Moreover, only a subset of these events may be required in any given scenario. At the same time, exposing too few of the events could result in missing a crucial control operation: e.g., not knowing a TCP connection has been reset may result in unnecessary pinholes rules remaining on a downstream firewall.

Takeaways. MBs should expose a broad interface for manipulating both internally and externally constructed MB state. A few basic operations are sufficient, but MBs must be enhanced to support these operations without introducing consistency, performance, or state validity problems. Event abstractions serve as a necessary complement, exposing MB's internal state manipulations and allowing MBs to request missing or updated state. Operations performed in response to events are determined by the control logic, discussed next.

5. CONTROL LOGIC

Sophisticated control logics can be run atop the abstractions and interfaces discussed in the previous sections to realize rich control over dynamic MB deployments. In this section, we discuss the design of control logics for the first two scenarios presented in §2.1. These example control logics illustrate how our proposed mechanisms fit together and highlights the challenges that emerge in control logic design.

VM Provisioning. When a new VM is provisioned, the control logic holds the responsibility for identifying the MBs the VM's traffic should traverse and instantiating the required state on these MBs. This is similar to the common SDN control task of identifying switches and installing forwarding state to establish a path through the network. However, the task is more complex because MBs' capabilities and state are

not uniform. The control logic requires an *augmented network graph*, akin to those maintained by SDNs, with special MB nodes that identify each MB's capabilities (e.g., using a MB-specific modeling language [6]) and current state. The control logic must search this graph to identify MBs providing the necessary functionality, and select specific MBs based on location (e.g., select a firewall close to the server to minimize the potential for insider attacks), the presence of existing state (e.g., use the load balancer already used by related VMs), and other factors.

Appropriate action state must be established at each of the selected MBs to achieve the security and performance an enterprise requires. A get operation can be used to determine if the required state already exists, e.g., because a similar VM is already using the MB. New state can be instantiated, when necessary, by defining a transformation function using the view abstraction and installing the state using the add operation. It is imperative that the state added to each MB considers the modifications made by other MBs the VM's traffic passes through: e.g., if the VM's traffic passes through a firewall and then a load balancer, the state added to the firewall must contain the IP address of the load balancer, not the IP of the VM. In some cases, the action state may not be known a priori: e.g., if outbound application flows pass through a NAT followed by a firewall, the necessary firewall pinholes are dependent on the port mapping selected by the NAT. In this case, the control logic can monitor for events raised by the NAT, e.g., indicating a new port mapping has been established, and subsequently add the corresponding state to the firewall.

Finally, to ensure traffic passes through the chosen MB sequence, the control logic (or an SDN controller) must install the appropriate forwarding rules in network switches. Again, careful attention must be paid to the manipulations MBs perform to ensure the right traffic is forwarded.

MB Scaling. The control logic for MB scaling is more complex because of the need to *migrate* state between MBs. When a MB is scaled down, all supporting state for active flows traversing that MB (*MB E*) must be moved to the remaining MBs (*MB R*) using get and add operations. The flows cannot be switched to traverse *MB R* until the state is moved. Care must be taken to ensure consistency: e.g., if a new packet for an active flow arrives (signaled by an event) at *MB E* after the flow's state has been moved, the packet must either be forwarded to *MB R* or the state from *MB E* must be moved again.

Similar design patterns and challenges arise in control applications developed for other scenarios.

6. RELATED WORK

Prior works have sought to provide specific forms of control over MBs. Sekar et. al present optimization formulations for dividing intrusion detection responsibilities based on traffic paths and IPS processing capabilities [10]. PLayer [7] passes traffic through specific MBs based on high-level policies. Both of these could be implemented as specific control logics in our framework. SIMCO [1], a protocol for dynamic configuration of NATs and firewalls, has goals similar to ours but only offers minimal control.

New MB deployment models, e.g. CoMB [9] and ETTM [4], are orthogonal to our framework's design, but unified control enables them to be better leveraged.

Our framework is grounded in the principles of SDN, which has a rich body of work ranging from specific control frameworks [2, 8] to high-level concepts [3].

7. CONCLUSION

Enriching enterprise application deployments in sophisticated ways requires taking full advantage of new MB deployment models and diverse MB functionality. Current ad hoc mechanisms for MB control, including by-hand tweaking, are clumsy and unsuitable for this task. Instead, we have advocated for the design of a *software-defined MB networking framework* capable of supporting scenarios like MB scaling and live network migration. We have examined in depth the major challenges in moving towards this control model—representing, manipulating, and knowledgeably controlling MB state—and presented our initial thoughts on the appropriate abstraction and interfaces to help address them.

Moving closer towards our goal of flexible, unified control requires further research into many additional issues, such as: representing the state associated with a broader range of MBs using well defined primitives, standardizing the representation of MB events, facilitating deep control of MB functionality, preventing invalid manipulations of MB state and ensuring consistency, augmenting MBs' internal logic to facilitate adequate manipulation of shared state, and designing control logics for a wide range of scenarios. More importantly, we believe that continued innovation in MB functionality and operation hinges on the development of SDN-like frameworks for MB management.

8. REFERENCES

[1] Rfc 4540: Nec's simple middlebox configuration (simco) protocol version 3.0. `http://tools.ietf.org/html/rfc4540`.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. In *SIGCOMM*, 2007.

[3] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. Virtualizing the network forwarding plane. In *PRESTO*, 2010.

[4] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. Ettm: A scalable fault tolerant network manager. In *NSDI*, 2011.

[5] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual middleboxes as first-class entities. Technical Report TR1771, University of Wisconsin-Madison, 2012.

[6] D. Joseph and I. Stoica. Modeling middleboxes. *IEEE Network*, 2008.

[7] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *SIGCOMM*, 2008.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM CCR*, 38(2), 2008.

[9] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *NSDI*, 2012.

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