Design and Implementation of a Framework for Software-Defined Middlebox Networking

Aaron Gember, Robert Grandl, Junaid Khalid, Aditya Akella
University of Wisconsin-Madison, Madison, WI, USA
{agember,rgrandl,junaid,akella}@cs.wisc.edu

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management

Keywords

Middlebox; Software-defined networking

1. MOTIVATION

Middleboxes (MBs) are used widely to ensure security (e.g., intrusion detection systems), improve performance (e.g., WAN optimizers), and provide other novel network functionality [4, 6]. Recently, researchers have proposed several new architectures for MB deployment, including Stratos [2], CoMb [4], and APLOMB [6]. These frameworks all advocate dynamic deployment of software-based MBs with the goal of increasing flexibility, improving efficiency, and reducing management overhead.

However, approaches for controlling the behavior of MBs (i.e., how MBs examine and modify network traffic) remain limited. Today, configuration policies and parameters are manipulated using narrow, MB-specific configuration interfaces, while internal algorithms and state are completely inaccessible and unmodifiable. This apparent lack of fine-grained control over MBs and their state precludes correct and performant implementation of control scenarios that involve re-allocating live flows across MBs: e.g., server migration, scale up/down of MBs to meet cost-performance trade-offs, recovery from network or MB failures, etc.

Several key requirements must be satisfied to effectively support the above scenarios. To illustrate these requirements, we consider a scenario where MB instances are added and removed based on current network load [2] (Figure 1). When scaling up, some in-progress flows may need to be moved to a new MB instance to reduce the load on the original instance. To preserve the correctness and fidelity of MB operations, the new instance must receive the internal MB state associated with the moved flows, while the old instance still has the internal state associated with the remaining flows. For some MBs (e.g., an intrusion prevention system (IPS) this means we need the ability to *move internal MB state at fine granularity*. For other MBs (e.g., a redundancy elimination (RE) system) we instead need the ability to *clone shared internal MB state*. Regardless of the type of MB being scaled, we want the new MB instance to behave the same as the original, requiring the ability to *clone and dynamically modify MB configurations*. When scaling down, we need to consolidate several MB instances into fewer instances, requiring the ability to *merge internal MB state from multiple MBs*. Finally, we need the ability to *coordinate MB state changes with network routing changes*; this ensures flows aren't directed to MB instances until they have the necessary state.

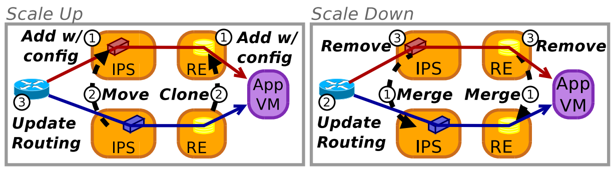

Figure 1: Horizontal scaling of MBs

Existing techniques—e.g., virtual machine snapshots, joint control of MB configuration and network routing [5], and application-level libraries [3]—can address some of these requirements, but these approaches have limited applicability and tend to reduce performance or cause correctness issues.

2. OBJECTIVE & CHALLENGES

Inspired by software-defined networking (SDN), we advocate for the development of a *software-defined middlebox networking* (SDMBN) framework to address the above requirements. An ideal SDMBN framework offers useful abstractions for fine-grained, software-driven control of MB internals without wresting too much control away from the MBs themselves. Such a carefully balanced framework can simplify management of complex MB deployments and engender a variety of rich dynamic MB control scenarios.

Designing an SDMBN framework requires addressing two key roadblocks. First, compared to switch forwarding state, MB state is highly diverse. A single MB may receive dozens of configuration inputs and its internal logic may establish and manipulate hundreds of pieces of in-depth state whose structure and semantics varies significantly across MB types and vendors. Second, internal MB logic is complex. Each MB features intricate and unique packet processing logic that is closely tied to internal state; unlike network switches, there is not a clean separation between control and data planes.

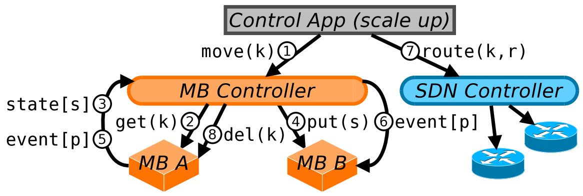

Figure 2: OpenMB architecture and example

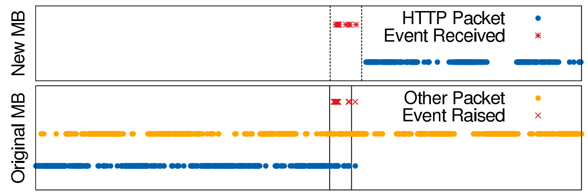

Figure 3: Actions/events during *scale up* scenario

3. FRAMEWORK DESIGN

We design and implement OpenMB, an exemplar SDMBN framework that overcomes the above challenges. OpenMB represents one point in the SDMBN design space, carefully trading-off some opportunities for vendor optimizations in exchange for increased control application flexibility. Its design is based on: (*i*) our observation that different MBs have commonalities in the role (configuring, supporting, or reporting) and partitioning (per-flow or shared) of pieces of MB state; and (*ii*) a careful division of responsibility for state changes—MBs are responsible for creating and modifying supporting and reporting state, as they do today, and control applications are responsible for manipulating where specific pieces of supporting and reporting state reside, as well as creating and updating all configuration state.

OpenMB's architecture consists of an MB controller, control applications, and slightly modified MBs as shown in Figure 2. Our MB-facing ("southbound") API defines how MB state is represented and how it can be access and manipulated at fine-granularity. In particular, state is represented as key-value pairs, with the key being either a string constant (for configuration state) or a flow identifier similar to the OpenFlow 10-tuple (for supporting and reporting state). State is installed in and retrieved from MBs using simple get, put, and delete calls. The southbound API also includes an event abstraction that allows MBs to notify the controller when they internally create or manipulate state. This model (unlike Split/Merge [3]) allows MBs to continue processing traffic while state is being moved or cloned and still guarantees the state is consistent and correct.

Our application-facing ("northbound") API encapsulates the intricacies of state operations on individual MBs by exposing a set of high-level operations (move, clone, merge, etc.) to control applications. The MB controller brokers these operations, issuing the appropriate southbound API calls directly to MBs, buffering and forwarding events, and dealing with operation failures. Exposing a separate API to control applications simplifies application design and limits the potential for applications to make state changes that will lead to correctness or performance issues.

Figure 2 shows the API calls invoked in the case of MB scale up. The *scale up* control application first issues the northbound API call `move(k)` to transfer a subset of state, identified by the key `k`, from MB A to MB B. The MB controller subsequently issues a series of southbound API calls to the appropriate MBs: it issues `get(k)` to MB A, receives state `s`, and issues `put(s)` to MB B. If a packet `p` is received by MB A during these operations and the processing of this packet involves a change to the state `s`, then MB A raises a re-process event for `p`; the controller passes `event[p]` to MB B, which makes the necessary changes to `s`. When `move` returns successfully, the application triggers an update of network forwarding state by issuing the call `route(k,r)` to the SDN controller to change the forwarding for flows identified by the key `k` to the route `r`. Finally, the MB controller issues a `del(k)` to MB A to flush the transfered state `s` which MB A no longer needs.

4. IMPLEMENTATION & DEMO

We have implemented a prototype of OpenMB consisting of an MB controller that implements our northbound API, four MBs—an intrusion prevention system (Bro), a traffic monitor (Prads), a redundancy elimination system (SmartRE [1]), and a network address translator (iptables)—modified to support our southbound API, and control applications for MB scaling and server migration scenarios. Our MB controller is implemented as a module ($\approx$ 1700 LOC) running atop the Floodlight OpenFlow controller. The MB controller is event driven to maximize scalability and efficiency. JSON messages are exchanged by the controller and MBs to invoke operations, send/receive state, and raise/forward events. The four modified MBs rely on a common code base ($\approx$1100 LOC) for MB-controller communications; additional MB-specific modifications are made to retrieve, insert, and remove state and to generate and process events.

Our demonstration uses this prototype to illustrate how OpenMB helps achieve dynamic fine-grained control in MB scaling and server migration scenarios. We show in real-time the sequence of actions performed by a control application, the MB controller, and MBs themselves. For example, Figure 3 shows the packet processing, API calls, and event raising/processing that occurs over a 3-second window when a Prads MB is scaled up and HTTP flows are moved to a new (top) Prads instance; the solid lines indicate the start and end of the `get` call issued to the original Prads instance, and the dashed lines indicate the start of the first and end of the last `put` call issued to the new Prads instance.

5. REFERENCES

[1] A. Anand, V. Sekar, and A. Akella. SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination. In *SIGCOMM*, 2009.

[2] A. Gember, A. Krishnamurthy, S. St. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. Stratos: A Network-Aware Orchestration Layer for Middleboxes in the Cloud. Technical Report arXiv:1305.0209, 2013.

[3] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/Merge: System Support for Elastic Execution in Virtual Middleboxes. In *NSDI*, 2013.

[4] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi. Design and Implementation of a Consolidated Middlebox Architecture. In *NSDI*, 2012.

[5] V. Sekar, R. Krishnaswamy, A. Gupta, and M. K. Reiter. Network-Wide Deployment of Intrusion Detection and Prevention Systems. In *CoNEXT*, 2010.

[6] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *SIGCOMM*, 2012.