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 shared internal MB state. 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.

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.
returns successfully, the application triggers an update of MB B, which makes the necessary changes to are - process event for MB A during these operations and the processing of this state to the appropriate MBs: it issues a put(s) to MB A to flush the transferred state s which MB A no longer needs.

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.

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