Managing middleboxes

• “Middlebox manifesto” (ref. previous lecture) pointed out the need for automated middlebox management

  • Many different solutions proposed afterwards

• Typical model: middleboxes are virtualized software appliances

  • Can be deployed on general-purpose servers

  • Area typically referred to as Network Function Virtualization
Managing middleboxes - II

- Implementing middleboxes in software offers great freedom...
- Allowing things like scaling and dynamic load balancing:

![Diagram of Scaling and Load-balancing](image-url)
Managing middleboxes - III

- The ability of shuffling middleboxes around creates various challenges

- Most are related to the stateful nature of middlebox processing

- E.g., an IDS keeps some detection state for each flow it is analyzing

- What happens to the state if the IDS is terminated or the flow is remapped to another IDS node?
Example: IDS MB scaled out during portscan detection

All traffic processed by the same middlebox:

Conn attempt #1
Conn attempt #2
Conn attempt #3

IDS 1

Portscan

Scale-out mid-attack:

Conn attempt #1
Conn attempt #2
Conn attempt #3

IDS 1

IDS 2
OpenNF: Enabling Innovation in Network Function Control

Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella
University of Wisconsin-Madison
{aember,raajay,cprikash,rgrandl,junaid,souravd,akella}@cs.wisc.edu
http://opennf.cs.wisc.edu

ABSTRACT
Network functions virtualization (NFV) together with software-defined networking (SDN) has the potential to help operators satisfy tight service level agreements, accurately monitor and manipulate network traffic, and minimize operating expenses. However, in scenarios that require packet processing to be redistributed across a collection of network function (NF) instances, simultaneously achieving all three goals requires a framework that provides efficient, coordinated control of both internal NF state and network forwarding state. To this end, we design a control plane called OpenNF. We use carefully designed APIs and a clever combination of events and forwarding updates to address race conditions, bound overhead, and accommodate a variety of NFs. Our evaluation shows that OpenNF offers efficient state control without compromising flexibility, and requires modest additions to NFs.

Categories and Subject Descriptors
C.2.1 [Computer Communication Networks]: Network Architecture and Design; C.2.3 [Computer Communication Networks]: Network Operations

Keywords
Network functions, middleboxes, software-defined networking

1. INTRODUCTION
Network functions (NFs), or middleboxes, are systems that examine and modify packets and flows in sophisticated ways: e.g., intrusion detection systems (IDSs), load balancers, caching proxies, etc. NFs play a critical role in ensuring security, improving performance, and providing other novel network functionalities [37].

Recently, we have seen a growing interest in replacing dedicated NF hardware with software-based NFs running on generic compute resources—a trend known as network functions virtualization (NFV) [12]. In parallel, software-defined networking (SDN) is being used to steer flows through appropriate NFs to enforce policies and jointly manage network and NF load [17, 20, 22, 26, 32].

Together, NFV and SDN can enable an important class of management applications that need to dynamically redistribute packet processing across multiple instances of an NF—e.g., NF load balancing [32] and elastic NF scaling [21]. In the context of such applications, “NFV + SDN” can help achieve three important goals: (1) satisfy tight service level agreements (SLAs) on NF performance or availability; (2) accurately monitor and manipulate network traffic, e.g., an IDS should raise alerts for all flows containing known malware; and (3) minimize NF operating costs. However, simultaneously achieving all three goals is not possible today, and fundamentally requires more control than NFV + SDN can offer.

To see why, consider a scenario where an IDS is overloaded and must be scaled out in order to satisfy SLAs on throughput (Figure 1). With NFV we can easily launch a new IDS instance, and with SDN we can reroute some in-progress flows to the new instance [17, 32]. However, attacks may go undetected because the necessary internal NF state is unavailable at the new instance. To overcome this problem, an SDN control application can wait for existing flows to terminate and only reroute new flows [22, 38], but this delays the mitigation of overload and increases the likelihood of SLA violations. NFV accuracy may also be impacted due to some NF-internal state not being copied or shared.

In this example, the only way to avoid a trade-off between NF accuracy and performance is to allow a control application to quickly and safely move the internal IDS state for some flows from the original instance to the new instance, and update network forwarding state alongside. Similar needs arise in the context of other applications that rely on dynamic reallocation of packet processing—e.g., rapid NF upgrades and dynamic invocation of remote processing.

In this paper, we present OpenNF, a control plane architecture that provides efficient, coordinated control of both internal NF state and network forwarding state to allow quick, safe, and fine-grained reallocation of flows across NF instances. Using OpenNF, operators can create rich control applications that redistribute processing to optimally meet their performance, availability, security and cost objectives, thus avoiding the need to make undesirable trade-offs.

We address three major challenges in designing OpenNF:

1. Addressing race conditions. This is the most basic issue that arises when reallocating in-progress flows: When some internal NF state is being moved, packets may arrive at the source instance after the move starts, or at the destination instance before the state transfer finishes. Unless care is taken, updates to NF state due to such packets may either be lost or happen out of order, violating move safety. Similarly, when state is copied across NF instances, updates occurring contemporaneously may cause state to become inconsistent. Depending on the NF, these issues may hurt its accuracy.

To account for race conditions, we introduce two novel constructs: (1) an event abstraction to externally observe and prevent...
Core problems

- Middleboxes tend to maintain state pertaining to the traffic they are processing

- Contrast to routers, switches etc. which do not need to “remember” anything besides the packet they are processing

- Problem: what happens to this state if processing switches to another middlebox?
OpenNF contributions

- **Mechanism** to coordinate scaling and load-balancing of middleboxes without service interruption
  - Based on state-transfer mechanism which can provide various useful properties (no packet lost/reordered)
  - Helps ensuring correctness (no state is lost/corrupted due to scaling/load-balancing)

- **Architecture** to coordinate state transfer operations
Relocating network functions

- Transferring network functions w/o service interruption requires coordination of:
  - **Application-level state transfer**: NF controller transfers state related to flows being redirected by the network level
  - **Network-level flow-steering**: SDN controller redirects flows from source to destination middlebox (already provided by standard OpenFlow network substrate)
OpenNF high-level architecture

Initiates/manages high-level operations (scaling, load-balancing, replication)

Control Application

NF State Manager

Flow Manager

Northbound API

Southbound API

Performs NF state transfer

Performs flow steering

Vendor-supplied controllers [4, 14] that move, copy, and share state for non-flow-based state, making it difficult to know the exact location of state.

API NFs must use to create and access states uses nondescript keys.

An NF to manage the flow of packets in and out of each instance.

Based on NF output or external input, control applications:

- Selectively invoking advanced remote processing.
- Fast failure recovery with low resource footprint.
- Combining existing control planes with techniques for VM migration.
- Fast elastic scale-down.

Further, to avoid race conditions: (1) an IDS may generate false alerts, but this can take a long time.

Based on preliminary observations made by a local NF, an enterprise may want to back up pieces of NF state as they are updated.

When rebalancing load, we must also account for the fact that original processing of the flow is later asked.

An IDS maintains connection counters for each end-host. If the processing of the flow is later terminated, but this can take a long time.

Figure 2: OpenNF architecture

Initiates/manages high-level operations (scaling, load-balancing, replication)
High-level functions to state manipulation operations

• Goal: **perform load-balancing by relocating a set of flows S from middlebox A to B:**
  • Move state for flows in S from A to B
  • Steer flows in S so that they traverse B instead of A

• Goal: **perform scaling by terminating middlebox A and relocating its processing task to B**
  • Same as above...
  • …but at the end A is terminated

• Goal: **replicate a middlebox A₁ to another middlebox A₂ to provide a “hot spare”**
  • Instantiate A₂
  • Periodically copy state for the set of flows S managed by A₁ to A₂

• …
State manipulation operations
to low-level operations

Northbound API:
move(srcInst, dstInst, filter, scope, properties)
copy(srcInst, dstInst, filter, scope)
share(list<inst>, filter, scope, consistency)

NFV controller: implements copy/move/share operations by issuing appropriate sequences of low-level Southbound API calls

Southbound API:
getPerflow(), putPerflow(), delPerflow(), getMultiflow(), putMultiflow(), delMultiflow(),
getAllflows(), delAllflows(), enableEvents(), disableEvents()
Types of state

- **Per-flow state**: program state keeping track of properties of a single network flow

- **Multi-flow state**: program state keeping track of aggregate properties of multiple flows

- **All-flow state**: program state keeping track of global properties of middlebox processing (information referring to and/or affecting all flows being processed)
OpenNF: *move* operation

- Can provide various properties (loss-freeness, order preservation)
  - Trade off: correctness vs delay
- Simplest case: move w/o guarantees:

**Flow of interest**

- **OpenFlow switch**
  - 1. `getPerflow(F)`
  - 2. `delPerflow(F)`
  - 3. `putPerflow(F)`
  - 4. `Reconfigure forwarding table`

**Flow of interest**

- **SDN controller**
  - State

**Flow of interest**

- **MBSRC**
  - State
- **MBDST**
  - State
Move w/o guarantees: issues

- State created in MB_{SRC} for packets arrived after getPerflow is not moved to MB_{DST}

- **Simplest solution:** instruct MB_{SRC} (or SDN switch) to ignore (or drop) all packets received after getPerflow

- Only OK if processing is resilient to packet losses…

- …and **still suboptimal**! E.g. IDS will continue to function but may miss attacks (e.g. portscans)

- OpenNF can provide additional guarantees: **loss-freeness**, **in-order delivery**
Loss-free move

• Leverage a novel OpenNF contribution: the event management API

• Allows NFV controller to instruct middleboxes to process, buffer, or drop certain categories of packets while at the same time forwarding the same packets to the NFV controller

• In loss-free move, these primitives are used to buffer and then forward all packets received by MB_{SRC} after state has been moved, to MB_{DST}
Loss-free move - part I

1. enableEvents(F, drop)
2. getPerflow(F)
3. delPerflow(F)

Flow of interest

OpenFlow switch

Packets

SDN controller

State

MB_{SRC}

State

MB_{DST}
Loss-free move - part II

Issue: packets may still arrive out-of-order at MB_{DST} (why?)
OpenNF: copy operation

- **Copy**: used e.g. for failure recovery (copy state from an active middle box to a backup instance)
  - Uses `getPerflow/putPerflow` (no deletion)
  - Provides *eventual consistency*
OpenNF: share operation

- **Share**: used to ensure that all state updates by a middlebox are directly reflected by the state of one or more other middleboxes (distributed processing)

- **Strict consistency**: state updates to a middlebox state are immediately reflected in the state of all other middleboxes (updates to the shared state reflects the order in which packets where received by switch)

- **Strong consistency**: order of state updates may differ from the order in which packets where received by switch/but all updates performed by an individual middlebox must be “seen” by other middleboxes in the order in which they were performed
Results

![Graph showing results](image)

**Figure 10:** Efficiency of **move** with no guarantees (NG), loss-free (LF), and loss-free and order-preserving (LF+OP) with and without parallelizing (PL) and early-release (ER) optimizations; traffic rate is 2500 packets/sec; times are averaged over 5 runs and the error bars show 95% confidence intervals.
Results - II

Figure 11: Impact of packet rate and number of per-flows states on parallelized move with and without a loss-free guarantee

(a) Packet drops during a parallelized move with no guarantees loss-free move

(b) Total time for a parallelized move with no guarantees loss-free move
Other challenges in middlebox management

- Designing policy languages to describe middlebox deployments
- Offloading middleboxes to the cloud