Day 18: Middleboxes
The Network Operator Perspective

Ack: Vyas Sekar, Aaron Gember, Felipe Huici, Zafar Qazi, Justine Sherry
DarpaNet Design Goals
DarpaNet Design Goals

- Survivability (Robustness)
- Different services
- Different network technologies
- management of different networks
- cost effective; easy to add hosts; accountability
- **Accountability**
- **Security**
- **Performance**
Need for Network Evolution

- New applications
- New devices
- Evolving threats
- Performance, Security, Compliance, Accountability
- Policy constraints

Percentage of Methods Used to Exfiltrate Data

- Exposed Private Web Application Interface (1.5%)
- HTTP File Upload Site (1.5%)
- Malware Capability: IRC (2%)
- Malware Capability: SMTP (4%)
- SQL Injection (8%)
- Native FTP Client (10%)
- Native Remote Access/ Application (27%)
- Malware Capability: FTP (17%)
A Survey

• 57 enterprise network administrators

• Small (< 1k hosts) to XL ( >100k hosts)

• Asked about deployment size, expenses, complexity, and failures.
How many middleboxes do you deploy?

Typically on par with # routers and switches.
What kinds of middleboxes do you deploy?

Many kinds of devices, all with different functions and management expertise required.
Network Evolution today: Middleboxes!

<table>
<thead>
<tr>
<th>Type of appliance</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firewalls</td>
<td>166</td>
</tr>
<tr>
<td>NIDS</td>
<td>127</td>
</tr>
<tr>
<td>Media gateways</td>
<td>110</td>
</tr>
<tr>
<td>Load balancers</td>
<td>67</td>
</tr>
<tr>
<td>Proxies</td>
<td>66</td>
</tr>
<tr>
<td>VPN gateways</td>
<td>45</td>
</tr>
<tr>
<td>WAN Optimizers</td>
<td>44</td>
</tr>
<tr>
<td>Voice gateways</td>
<td>11</td>
</tr>
</tbody>
</table>

**Total Middleboxes** 636

**Total routers** ~900

Data from a large enterprise: >80K users across tens of sites

Just network security $10 billion
How many networking personnel are there?

Average salary for a network engineer - $60-80k USD
How do administrators spend their time?

Most administrators spent 1-5 hrs/week dealing with failures; 9% spent 6-10 hrs/week.

<table>
<thead>
<tr>
<th></th>
<th>Misconfig.</th>
<th>Overload</th>
<th>Physical/Electrical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firewalls</td>
<td>67.3%</td>
<td>16.3%</td>
<td>16.3%</td>
</tr>
<tr>
<td>Proxies</td>
<td>63.2%</td>
<td>15.7%</td>
<td>21.1%</td>
</tr>
<tr>
<td>IDS</td>
<td>54.45%</td>
<td>11.4%</td>
<td>34%</td>
</tr>
</tbody>
</table>
Why are Middleboxes So Complex?
Example: Detecting Network Attacks

- Inspect all DNS traffic with a DPI device
- If suspicious lookup takes place, send to traffic scrubber
Example: Detecting Network Attacks

- Inspect all DNS traffic with a DPI device
- If suspicious lookup takes place, send to traffic scrubber
Example: Detecting Network Attacks

- Inspect all DNS traffic with a DPI device
- If suspicious lookup takes place, send to traffic scrubber
Example: Detecting Network Attacks

- Inspect all DNS traffic with a DPI device
- If suspicious lookup takes place, send to traffic scrubber
Why are Middleboxes So Complex?

• Dynamic Network Policies
Example: Detecting Network Attacks

- Hard to manually configure routing to ensure Middlebox traversal
Example: Detecting Network Attacks

- Hard to manually configure routing to ensure Middlebox traversal
- Force Middlebox traversal by creating chokepoints
  - Makes the network brittle. If link fails → network is down
  - All traffic goes through the Middleboxes → easy to overload
Why are Middleboxes So Complex?

• Dynamic Network Policies

• Hard to Ensure Middlebox Traversal
  – Inflexible Network

• Inefficient resource utilization
  – MB gets overloaded
Consolidation reduces CapEx

Now that each Middlebox Processes all traffic you need to worry about utilization

Different Middleboxes have different peaks, lost opportunity to multiplex!!!!
How Can We fix Middleboxes?

• Move them to the Cloud?

• Virtualize and Consolidate: NFV!
Proposal: Move To the Cloud
Proposal: Move To the Cloud

Make the configuration and provisioning issues someone’s problem!
Challenges

• Functional Equivalence
  – Behave exactly like it did before

• Low Performance Overhead
“Appliance for Outsourcing Middleboxes”

• Place middleboxes in the cloud.

• Use APLOMB devices and DNS to redirect traffic to and from the cloud.

• That’s it.
Outsourcing Middleboxes with APLOMB

Internet

Cloud Provider

VPN Tunnel

NAT

Internet

APLOMB Gateway (VPN)
Inbound Traffic

DNS responds with the address for a MB in the Cloud

Enterprise
Network Admin.

www.enterprise.com
192.168.1.100

98.76.54.32
Inbound Traffic

Web Server: www.enterprise.com
192.168.1.100

Enterprise
Network Admin.

Enterprise

Internet

Cloud Provider
98.76.54.32

www.enterprise.com
192.168.1.100

enterprise.com
98.76.54.32
Challenges

• Functional Equivalence
  – Behave exactly like it did before

• Low Performance Overhead??
Challenges

• Functional Equivalence
  – Behave exactly like it did before

• Low Performance Overhead
  – Little Latency Overhead
    • Going to the Cloud and back incurs huge overheads!
  – Low Bandwidth Overheads
    • Clouds charges for every byte.
Latency Optimization: Choosing a Datacenter

Route through cloud datacenter that minimizes end to end latency.

APLOMB Gateway keeps a “routing table” to select best tunnel for every Internet prefix.
Bandwidth Optimization: APLOMB+ for Compression
Add generic compression to APLOMB gateway to reduce bandwidth consumption.
APLOMB!

• Move MB to the Cloud
  – Cloud provides deals with all the complexity
• Move MB to the Cloud
  – Cloud provides deals with all the complexity

• Problems fixed?
  – Configuration issues
  – Provisioning issues

• Open Issues
  – Dynamic policies
  – Inefficient resource utilization: Monolithic Entities

• New Problems:
  – Introduces latency
How Can We fix Middleboxes?

• Move them to the Cloud?

• Virtualize and Consolidate: NFV!
Virtualize and Consolidate

Today: Independent, specialized boxes, Monolithic!!

Proxy
Firewall
IDS/IPS
AppFilter

Decouple Hardware and Software (Virtualize!!!)

Commodity hardware: e.g., PacketShader, RouteBricks, ServerSwitch, SwitchBlade

Consolidation reduces capital expenses and sprawl
Virtualize and Consolidate

Contribution of reusable modules: 30 – 80 %
How to flexibly allocate resources?

- **Recall**: Today all Middleboxes are placed at the edge!!
How to flexibly allocate resources?

Today: All processing at logical “ingress”

Overload!

Distribute Middleboxes across the Network to balance load!!!
Key idea: Consolidation

Two levels corresponding to two sources of inefficiency:

1. Consolidate Platform
2. Consolidate Management

Where to place MB? What traffic should go to MB?
Concrete System Overview: CoMb

Network-wide Controller (SDN)

Logically centralized e.g., NOX, 4D

General-purpose hardware: e.g., PacketShader, RouteBricks, ServerSwitch,

Existing work: simple, homogeneous routing-like workload

Middleboxes: complex, heterogeneous, new opportunities
Challenges

• Resource allocation problem

• Efficient Hardware implementation
Modeling Processing Coverage

HTTP: Run IDS < Proxy

<table>
<thead>
<tr>
<th>Node</th>
<th>HTTP Traffic</th>
<th>IDS &lt; Proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td>N2</td>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td>N3</td>
<td></td>
<td>0.3</td>
</tr>
</tbody>
</table>

What fraction of traffic of class HTTP from N1 → N3 should each node process?
CoMb Management Layer

Goal: Balance load across network.
Leverage multiplexing, reuse, distribution

Policy Constraints  Resource Requirements  Routing, Traffic

Network-wide Controller

Processing responsibilities
Network-wide Optimization
Minimize Maximum Load, Subject to

Processing coverage for each class of traffic
→ Fraction of processed traffic adds up to 1

Load on each node
→ sum over HyperApp responsibilities per-path

A simple, tractable linear program
Very close (< 0.1%) to theoretical optimal
Challenges

• Resource allocation problem

• Efficient Hardware implementation
CoMb Platform

Applications

IDS

... Core1 ...

... ...

Proxy

Core4

Challenges: Performance
Parallelize
Isolation

Policy Enforcer

IDS < Proxy

Policy Shim (Pshim)

Challenges: Lightweight
Parallelize

Classification:

HTTP

NIC

Traffic

Challenges: No contention
Fast classification
Parallelizing Application Instances

**App-per-core**

- Inter-core communication
- More work for PShim
- No in-core context switch

**HyperApp-per-core**

+ Keeps structures core-local
+ Better for reuse
- But incurs context-switch
- Need replicas

HyperApp-per-core is better or comparable
Contention does not seem to matter!
CoMb Platform Design

Core-local processing

Core 1
- M1
- Hyper App1
- PShim

Core 2
- M1
- Hyper App3
- PShim

Core 3
- M5
- Hyper App4
- PShim

Hyper App2

Parallel, core-local

NIC hardware

Contention-free network I/O

Workload balancing
Capturing Reuse with HyperApps

HTTP:
- 1+2 unit of CPU
- 1+3 units of mem

HyperApp: find the union of apps to run

HTTP = IDS & Proxy

UDP = IDS

NFS = Proxy

CPU
Memory

Footprint on resource

Need per-packet policy, reuse dependencies!
Capturing Reuse with HyperApps

**HTTP:**
1+2 unit of CPU
1+3 units of mem

**HyperApp:** find the *union* of apps to run

HTTP = IDS & Proxy

HTTP
UDP
NFS

CPU
Memory

NFS = Proxy

CPU
Memory

HyperApp: All MB (app) that need to process a flow.

Need per-packet policy, reuse dependencies!
CoMB

• Virtualize and Consolidate
• Problems fixed:
  – Provisioning issues
  – Inefficient resource utilization: Monolithic Entities

• Open Issues
  – Dynamic policies
  – Configuration issues

• New Problems:
  – Centralized management system → new interfaces
CoMB

• Virtualize and Consolidate
Managing Middleboxes

• Is Challenging:
  – Dynamic Network Policies
  – Hard to Ensure Middlebox Traversal
  – Inefficient resource utilization

• Existing Solutions:
  – Virtualize the Middleboxes
    • Improves Resource utilization
    • Can make configuration someone else’s policy
  – Introduce Centralized Control
    • Ensures Middlebox Traversal
How do administrators spend their time?

Most administrators spent 1-5 hrs/week dealing with failures; 9% spent 6-10 hrs/week.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Misconfig.</td>
<td>Firewalls</td>
<td>67.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proxies</td>
<td>63.2%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IDS</td>
<td>54.45%</td>
</tr>
</tbody>
</table>
SDN Stack

- **App**: Applications
- **Runtime**: Control Flow, Data Structures, etc.
- **Controller Platform**: (OpenFlow)
- **Switches**

**Controller API**

**Switch API**

**Controller Platform**

**Controller**

**Applications**
Outline

• Why middleboxes?
• SIMPLE
• OpenMB
• Slick
Can SDN simplify middlebox management?

**Centralized Controller**

Web
- Firewall
- IDS
- Proxy

OpenFlow

**Scope:** Enforce middlebox-specific steering policies

**Necessity + Opportunity:**
Incorporate functions markets views as important
What makes this problem challenging?

Middleboxes introduce new dimensions beyond L2/L3 tasks.

Achieve this with **unmodified** middleboxes and **existing** SDN APIs.
SIMPLE overview

Policy enforcement layer for middlebox-specific “traffic steering”

Legacy Middleboxes

OpenFlow capable
Challenge: Policy Composition

Policy Chain:

* → Firewall → IDS → Proxy

Oops!
Forward Pkt to IDS or Dst?

“Loops”
Traditional flow rules may not suffice!
Challenge: Resource Constraints

Can we set up “feasible” forwarding rules?
Challenge: Dynamic Modifications

User1: Proxy → Firewall
User2: Proxy

Are forwarding rules at S2 correct?

Proxy may modify flows
New dimensions beyond Layer 2-3 tasks

1) Policy Composition → Potential loops

2) Resource Constraints → Switch + Middlebox

3) Dynamic Modifications → Correctness?

Can we address these with *unmodified* middleboxes and *existing* SDN APIs?
Composition ➞ Tag Processing State

Policy Chain: * → Firewall → IDS → Proxy

Insight: Distinguish different instances of the same packet
SIMPLE System Overview

- Resource Manager
- Modifications Handler
- Rule Generator

Legacy Middleboxes

OpenFlow capable
Resource Constraints $\rightarrow$ Joint Optimization

Theoretically hard!
Not obvious if some configuration is feasible!
Offline + Online Decomposition

Policy Spec
Network Topology
Switch TCAM
Mbox Capacity + Footprints
Traffic Matrix

Resource Manager

Offline Stage
Deals with Switch constraints

Online Step
Deals with only load balancing
Offline Stage: ILP based pruning

Set of all possible middlebox load distributions

Pruned set

- Feasible
- Sufficient freedom

Balance the middlebox load
SIMPLE System Overview

- Web
- FW
- IDS
- Proxy

Resource Manager

Rule Generator

Modifications Handler

Legacy Middleboxes

OpenFlow capable
Modifications $\rightarrow$ Infer flow correlations

Correlate flows $\rightarrow$ Install rules

Payload Similarity

User 1: Proxy $\rightarrow$ Firewall
User 2: Proxy
SIMPLE Implementation

- Resource Manager *(Resource Constraint)*
- CPLEX
- Modifications Handler *(Dynamic modifications)*
- Rule Generator *(Policy Composition)*
- POX extensions

OpenFlow 1.0

```
Flow  Tag/Tunnel  Action
---   ----------  -----
      |           |
      |           |
      |           |
      |           |
      |           |
      |           |
      |           |
```

```
Flow  Tag/Tunnel  Action
---   ----------  -----
      |           |
      |           |
      |           |
      |           |
      |           |
      |           |
      |           |
```
Evaluation and Methodology

• What benefits SIMPLE offers? load balancing?
• How scalable is the SIMPLE optimizer?
• How close is the SIMPLE optimizer to the optimal?
• How accurate is the dynamic inference?

Methodology

– Small-scale real test bed experiments (Emulab)
– Evaluation over Mininet (with up to 60 nodes)
– Large-scale trace driven simulations (for convergence times)
Summary of SIMPLE

- Middleboxes: Necessity and opportunity for SDN

- Goal: Simplify middlebox-specific policy enforcement

- Challenges: Composition, resource constraints, modifications

- SIMPLE: policy enforcement layer
  - Does not modify middleboxes
  - No changes to SDN APIs
  - No visibility required into the internal of middleboxes

- Scalable and offers 4-7X improvement in load balancing
Outline

• Why middleboxes?

• SIMPLE

• OpenMB

• Slick
Middlebox Deployment Models

- Arbitrary middlebox placement
- New forms of middlebox deployment (VMs, ETTM \textit{[NSDI 2011]}, CoMB \textit{[NSDI 2012]})
Live Data Center Migration

• Move between software-defined data centers

• Existing VM and network migration methods
  – Unsuitable for changing underlying substrate

Programmatic control over middlebox state
Middlebox Scaling

- Add or remove middlebox VMs based on load

- Clone VM (logic, policy, and internal state)
  - Unsuitable for scaling down or some scaling up

Fine-grained control
Contributions

• Classify middlebox state, and discuss what should be controlled

• Abstractions and interfaces
  – Representing state
  – Manipulating where state resides
  – Announcing state-related events

• Control logic design sketches
Software-Defined Middlebox Networking

Today

SDN-like Middleboxes

Controller

App

App

Middlebox

Middlebox

IPS
Key Issues

1. How is the logic divided?

Diagram:
- App
- Controller
- Middlebox
Middlebox State

- Configuration input + detailed internal records

- **Src**: HostA
  - **Server**: B
  - **Proto**: TCP
  - **Port**: 22

- **Server**: B
  - **CPU**: 50%

- **State**: ESTAB
  - **Seq #**: 3423

- **Hash**: 34225
  - **Content**: ABCDE

- **Balance Method**: Round Robin

- **Cache size**: 100

Significant state diversity
Classification of State

Action
- Src: HostA
- Server: B
- Proto: TCP
- Port: 22

Supporting
- Server: B
- CPU: 50%
- State: ESTAB
- Seq #: 3423
- Hash: 34225
- Content: ABCDE

Tuning
- Balance Method: Round Robin
- Only affects performance, not correctness
- Cache size: 100

Internal & dynamic

Many forms
How to Represent State?

Per flow

Policy Language

1010110

1111000

Significant diversity

May be shared

Unknown structure

Commonality among middlebox operations
## State Representation

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
<th>Supporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field1 = Value1</td>
<td>Offset1 → Const1</td>
<td>Binary Blob</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>FieldN = ValueN</td>
<td>OffsetN → ConstN</td>
<td></td>
</tr>
</tbody>
</table>

- **Key:** protocol header field/value pairs identify traffic subsets to which state applies
- **Action:** transformation function to change parts of packet to new constants
- **Supporting:** binary blob

- Only suitable for per-flow state
- Not fully vendor independent
How to Manipulate State?

• Today: only control some state
  – Constrains flexibility and sophistication

• Manipulate all state at controller
  – Removes too much functionality from middleboxes
State Manipulation

- Control over state placement
  1. Broad operations interface
  2. Expose state-related events
Operations Interface

- Need atomic blocks of operations
- Potential for invalid manipulations of state

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Proto</th>
<th>Other</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>10.20.1.0/24</td>
<td>TCP</td>
<td>*</td>
<td>DROP</td>
</tr>
</tbody>
</table>

Key
SrclIP = 10.10.0.0/16
DPort = 22

Action
*
Events Interface

- Triggers
  - Created/updated state
  - Require state to complete operation
- Contents
  - Key
  - Copy of packet?
  - Copy of new state?

Balance visibility and overhead
Conclusion

• Need fine-grained, centralized control over middlebox state to support rich scenarios

• Challenges: state diversity, unknown semantics

get/add/remove ( ... , )
Open Questions

• Encoding supporting state/other action state?
• Preventing invalid state manipulations?
• Exposing events with sufficient detail?
• Maintaining operation during state changes?
• Designing a variety of control logics?
• Providing middlebox fault tolerance?
Outline

• Why middleboxes?

• SIMPLE

• OpenMB

• Slick
Network Policies

• Reachability
  – Alice can not send packets to Bob

• Application classification
  – Place Skype traffic in the gold queue
Limitations of SDN Data Plane

- Limited actions and matching
  - Match: Ethernet, IP, TCP/UDP port numbers
  - Action: forward, drop, rewrite header, etc.

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2.3.4:10.2.3.3</td>
<td>Fwd Port 1</td>
</tr>
<tr>
<td>A2:e3:f1:ba:ea:23:*</td>
<td>Drop</td>
</tr>
</tbody>
</table>
Extending SDN’s Data Plane

• Expand the OpenFlow standards
  – Requires hardware support

• Implement richer data plane in controller
  – Introduces additional latency to packets

• Add new devices (Middleboxes)
Challenges

• Specify network policies across middleboxes
  – Difficult to automatically react to middlebox events

• Dynamically place sophisticated middleboxes
  – Difficult to determine efficient placement
  – Difficult to adjust placement to traffic patterns

• Support for arbitrary middlebox functionality
  – Difficult to capture hardware requirements
Slick Contributions

• Abstraction for programming middleboxes
  – Simplifies the development of network policies
  – Separates specification of intent from implementation

• Dynamic placement of middlebox functionality
  – Online resource allocation algorithm

• Support for heterogeneous devices
  – Maintains performance profiles of middlebox
Slick Architecture

- **Application**
  - Encodes network policy
  - Provides handlers for triggers

- **Slick Controller**
  - Piece of code encapsulating middlebox functions

- **Middlebox Element**

- **Virtual Switch**
  - Programmable device: NetFPGA, x86 server

- **3rd party element developers**

- **Your network operator**

- **Programmable device:** NetFPGA, x86 server
Slick Architecture

- **Application**: Runs applications
- **Slick Controller**: Runs resource allocation algo.
  - Places middlebox elements
  - Steers traffic through middleboxes
  - Configures switches
- **Middlebox Element**: Installs/uninstalls middlebox functions
- **Virtual Switch**: NetFPGA, x86 server
- **Programmable device**: Virtual Switch
- **Deploy Middlebox code**: Deploy middlebox code
Resource Allocation Heuristic

Objective: minimize latency (path lengths)

Traffic matrix
And topology

Network policies in applications

Middlebox perf profile

Hardware constraints

Traffic Steering

Placement Decisions

OpenFlow Controller

Virtual Switch
Programmable device

Virtual Switch
Programmable device

Programmable device
Summary

• **Slick**: control plane for middleboxes
  – Presented an initial architecture
  – Discussed algorithmic challenge

• **Slick** is implemented in python
  – Slick controller as a module on NoX 0.5.0
  – Developed 2 applications and 3 middlebox elements

• **Open questions**
  – How can developers help guide placement?
  – What is the optimal solution for resource allocation?
Discussion: Likes/dislikes?