Reusing Network Services Logic to Improve Network Performance

Yotam Harchol
vmware® Research

(This work was done while at the Hebrew University)

Joint work with Anat Bremler-Barr and David Hay

Appeared in ACM SIGCOMM 2016

This research was supported by the European Research Council ERC Grant agreement no 259085, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), and the Neptune Consortium.
Network Functions (Middleboxes)

- Monolithic **closed** black-boxes
  - ✗ High *cost*
  - ✗ Limited *provisioning* and *scalability*

Network Function Virtualization (NFV):
- ✓ Reduce *cost* (by moving to software)
- ✓ Improve *provisioning* and *scalability* (by virtualizing software NFs)

At the cost of:
- ✗ Reduced *performance* (mainly *latency*)
Network Functions (Middleboxes)

- High **cost**
- Limited **provisioning** and **scalability**
- Limited and separate **management**
  - Different vendors
  - No standards
  - Separate control plane
Network Functions (Middleboxes)

- Actually, many of these black-boxes are very modular

- High cost
- Limited provisioning and scalability
- Limited and separate management

- Limited *functionality* and limited *innovation* (High entry barriers)

- Similar complex processing steps, *no re-use*
OpenBox

• **OpenBox**: A new software-defined framework for network functions
• Decouples network function control from their data plane
• Unifies data plane of multiple network functions

Benefits:
✓ Easier, unified control
✓ Better performance (improved latency)
✓ Scalability
✓ Flexible deployment
✓ Inter-tenant isolation
✓ Innovation

[www.openboxproject.org](http://www.openboxproject.org)
github.com/OpenBoxProject
Software Defined Networking

- High cost of middleboxes switches
- Limited provisioning and scalability of middleboxes switches
- Limited management of middleboxes switches
- Limited functionality and limited innovation
- Complex processing steps
- Distributed algorithms

40%-60% of the appliances in large-scale networks are middleboxes!
[Sherry & Ratnasamy, ‘12]
The OpenBox Framework

- Network Functions: OpenBox Applications
- Logically-Centralized OpenBox Controller
- Northbound API
- OpenBox Protocol
- OpenBox Service Instances

Additionally:
- Isolation between NFs / multiple tenants
- Support for hardware accelerators
- Dynamically extend the protocol
Observation:
Most network functions do very similar processing steps
But there is no re-use…

The design the OpenBox framework is based on this observation
Network Function Decomposition

Firewall:
- Read Packets
- Header Classifier
- Drop
- Alert
- Output

Load Balancer:
- Read Packets
- Header Classifier
- Rewrite Header
- Output

Intrusion Prevention System:
- Read Packets
- Header Classifier
- DPI
- DPI
- DPI
- Drop
- Alert
- Output
Northbound API

Specifying processing graph and block configuration

Events, Load information

OpenBox Protocol

OpenBox Service Instances

OpenBox Controller

Control Plane

Data Plane

Intrusion Prevention System

Load Balancer

Firewall

OpenBox Applications

Read Packets
Header Classifier
Alert
Drop
Output

Read Packets
Header Classifier
Rewrite Header
Output

Read Packets
Header Classifier
Drop
Alert
Output
Logically-Centralized Controller

Multiple tenants run multiple applications for multiple policies in the same network.

Isolation between applications and tenants enforced by NB API.

Network-wide view for automatic scaling, provisioning, placement, and steering.
Naïve Graph Merge

Firewall:

1. Read Packets
2. Header Classifier
3. Drop
4. Alert
5. Output

Intrusion Prevention System:

1. Read Packets
2. Header Classifier
3. Drop
4. Alert (Firewall)
5. Alert (IPS)
6. DPI
7. DPI
8. DPI
9. Drop
10. Output

Concatenated Processing Graph:

1. Read Packets
2. Header Classifier
3. DPI
4. Header Classifier
5. DPI
6. DPI
7. Alert
8. Output

Performance ≈ Diameter of Graph (# of classifiers)

Total: 134μs
Graph Merge Algorithm

**Input Graphs:**

**Firewall:**
- Read Packets
- Header Classifier
- Drop
- Alert
- Output

**Intrusion Prevention System:**
- Read Packets
- Header Classifier
- DPI
- DPI
- DPI
- Drop
- Alert
- Output
Graph Merge Algorithm

Step 1: Normalize graphs to trees

**Firewall:**

1. **Read Packets** → **Header Classifier** → **Drop** → **Output**
   - **Alert** → **Output**

**Intrusion Prevention System:**

1. **Read Packets** → **Header Classifier** → **DPI** → **Drop** → **Output**
   - **Alert** → **Output**
   - **DPI** → **Alert** → **Output**
   - **DPI** → **Drop** → **Output**
   - **Output** → **Output**
Graph Merge Algorithm

Step 2: Concatenate graphs
Graph Merge Algorithm

Step 3: Merge classifiers
Graph Merge Algorithm

Step 3: Merge classifiers

Can we change block order?
Graph Merge Algorithm

Step 3: Merge classifiers

Read Packets → Header Classifier

Drop → DPI → Output

Drop → DPI → Alert (IPS) → Output

Drop → DPI → Alert (IPS) → Output

Alert (Firewall) → DPI → Alert (Firewall) → DPI → Alert (Firewall) → Output

Alert (Firewall) → DPI → Alert (Firewall) → DPI → Alert (Firewall) → Output

Alert (Firewall) → DPI → Alert (Firewall) → DPI → Alert (Firewall) → Output

Alert (Firewall) → DPI → Alert (Firewall) → DPI → Alert (Firewall) → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output

Alert (IPS) → Drop → Output
Step 4: Remove redundant block copies
(and rewire connectors accordingly)
Graph Merge Algorithm

Merged Processing Graph:

- Read Packets: 2μs
- Header Classifier: 30μs
- Alert (Firewall): 10μs
- DPI: 50μs
- Alert (Firewall): 10μs
- Alert (Firewall): 50μs
- DPI: 10μs
- Alert (IPS): 2μs
- Output: 2μs

Shorter Diameter (less classifiers)

Total: 104μs (22% improvement)
When NOT to Merge?

When cross product is too large:
- Two d-dimensional classifiers: A – n rules, B – m rules
- Classification is logarithmic with # of rules, exponential with dimension
- Serial classification time: \((\log n)^{d-1} + (\log m)^{d-1}\)
- Cross product: \(n \cdot m\) rules (worst case)
- Single classifier worst case time:

\[
\log(n \cdot m)^{d-1} = (\log n)^{d-1} + (\log m)^{d-1} + \sum_{i=1}^{d-2} \binom{d-1}{i} ((\log n)^i + (\log m)^{d-i-1}) > (\log n)^{d-1} + (\log m)^{d-1}
\]

When most packets won’t go through both classifiers:
OpenBox Data Plane Processing

Terminals
- Read Packets
- Output
- Drop

Classification
- Header Classifier
- DPI
- VLAN Push
- VLAN Pop
- Rewrite Header

Header Modification

Caching
- Store Packet
- Restore Packet

Queue Management
- FIFO Queue
- Front Drop Queue
- Leaky Bucket
- RED Queue

Transactions
- Begin Transaction
- Commit Transaction
- Rollback Transaction

Normalization
- HTML Normalizer
- JavaScript Normalizer
- XML Normalizer

De/compression
- Gzip Decompress
- Gzip Compress

Output

Reporting
- Alert
- Log

De/compression
- Gzip Decompress
- Gzip Compress

De/compression
- Gzip Decompress
OpenBox Data Plane Processing

- Provides data plane services to realize the logic of network functions
- Controlled by the logically-centralized OpenBox controller
Distributed Data Plane

OpenBox Service Instance

Hardware (TCAM)

E.g., an OpenFlow switch with encapsulation features (e.g., NSH, Geneve, FlowTags)

Alert
DPI
Rewrite Header

OpenBox Service Instance

Software

Header Classifier

Metadata
Split Processing Graph

HW Instance:

- Read Packets
- Header Classifier
- Write Metadata
- Encapsulate Metadata
- Output
- Drop

SW Instance:

- Read Packets
- Decapsulate Metadata
- Read Metadata
- DPI
- Alert
- Drop
- Output
Distributed Data Plane
Extensible Data Plane

Option 1:
New hardware implementation
Supports encapsulation

Option 2:
Software module injection

Custom software module (signed)

On the fly
No need to recompile
No need to redeploy
Scalable & Reliable Data Plane

Scalability

Provisioning

Reliability

OpenBox Controller

Hypervisor
OpenBox Protocol: Block Hierarchy

Abstract Processing Block

- HeaderClassifier
- TCAMClassifier
- TrieClassifier

Hello
...
Supported implementations:
HeaderClassifier:
[TCAMClassifier, TrieClassifier]

SetProcessingGraphRequest
...
Use TCAMClassifier in graph
Future Work: Infrastructure Support

- Infrastructure can help VNFs
  - Provide high performance (e.g., hardware accelerators)
  - Reuse processing (e.g., packet switching, “outsourced” services)

- Challenge: Design a system, define a protocol to offload processing from VNFs to infrastructure

- Gradual solution, easier to adopt for existing VNFs
Implementation

github.com/OpenBoxProject

Java-based OpenBox Controller

7500 LoCs (Java)

Control Plane

Data Plane

Software OpenBox Service Instance

REST client/server

Graph Aggregator

Network Manager

Management API

FW

IPS

Load Balancer

Northbound API

Generic wrapper for execution engines (Python)

Translation Engine

Click-based execution engine (C++)

2400 LoCs for plugin (C++)

(Plug here other execution engines. E.g., ClickNP [SIGCOMM ‘16])
Performance Improvement

Without OpenBox

With OpenBox

VM1
Firewall

VM2
IPS

OBI1: FW+IPS

OBI2: FW+IPS

Standalone VM

NF Pipeline

Latency [µs]

Throughput [Mbps]

0
10
20
30
40
50
60
70
80
900

0
100
200
300
400
500
600
700
800
900

0
10
20
30
40
50
60
70
80
900

0
10
20
30
40
50
60
70
80
900

Without OpenBox

With OpenBox

-35%

+86%
Related Work

• Orthogonal to OpenBox:
  – NF traffic steering (e.g., SIMPLE [SIGCOMM ’14])
  – NF orchestration (e.g., Stratos, OpenMano, OpenStack)
  – Runtime platforms (e.g., xOMB [ANCS ’12], ClickNP [SIGCOMM ’16])

• Similar Motivation:
  – CoMb [NSDI ‘12] – focuses on resource sharing and placement
  – Slick [SOSR ’15] – focuses on the placement of data plane units

• Only OpenBox provides:
  – Core processing decomposition and reuse
  – Standardization and full decoupling of NF control and data planes
Conclusions

• Network functions are currently a real challenge in large scale networks

• By decoupling the data plane processing of NFs their control logic we:
  – Reduce costs
  – Enhance performance
  – Improve scalability
  – Increase reliability
  – Provide inter-tenant isolation
  – Allow easier innovation

• There is still work to do...
Questions?

THANK YOU!

Play with OpenBox on a Mininet VM:
github.com/OpenBoxProject/openbox-mininet