Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions

Guyue (Grace) Liu, Yuxin Ren, Mykola Yurchenko

K.K. Ramakrishnan, Timothy Wood
Why Improve Existing NFV Frameworks?

• Existing NFV frameworks focus on L2/L3 processing

- BESS: A Virtual Switch Tailored for NFV
- openNetVM [Hotmiddlebox’16]
- E2 [SOSP’15]
- ClickOS [NSDI’14]
- netmap - the fast packet I/O framework [Usenix ATC ‘12]
- PF_RING [SANE’04]
Why Improve Existing NFV Frameworks?

• Existing NFV frameworks are based on a packet-centric model

Diagram showing layers of network: L1 Physical, L2 Data Link, L3 Network, L4 Transport, L5-7 Application, with protocols such as UDP, TCP, Ethernet, PPP, LLDP, IP, ICMP, ARP, HTTP, DNS, FTP, SMTP, SSH, POP, Telnet, L3 Fwd, Firewall, IPsec, Shaper, L2 Fwd, NAT, Packet, Data, NF1, NF2, NF3, NFV IO, NIC.
Why Improve Existing NFV Frameworks?

- Existing NFV frameworks are based on a packet-centric model.
- Protocol processing becomes part of the NF. Repeated protocol stack processing within a chain - redundant.
Issue #1: Redundant Stack Processing

- As the chain length increases, the overhead grows significantly when going through stack processing multiple times
Idea #1: Consolidate Stack Processing

• How can we remove the redundancy within a chain?
Idea #1: Consolidate Stack Processing

• How can we remove the redundancy within a chain?
• Deploy all NFs and the stack as a single, monolithic process?
Idea #1: Consolidate Stack Processing

• How can we remove the redundancy within a chain?
• Move stack processing from NF into NFV framework
Issue #2: A Monolithic Stack is Not Efficient

- The throughput drops as the stack processing grows in functionality
Idea #2: Customizable Stack Modules

- How to avoid unnecessary processing in the stack?
Idea #2: Customizable Stack Modules

• How to avoid unnecessary processing in the stack?

• Split stack into modules based on functionality and customize processing for each NF/chain
Issue #3: Separate Stacks for NFs and Endpoint Applications

- Middlebox NFs and Endpoint applications use different underlying frameworks for protocol support

![Diagram showing separate stacks for NFs and endpoint applications]
Issue #3: Separate Stacks for NFs and Endpoint Applications

- How to transparently manage both middleboxes and endpoints?
Idea #3: Event Communication Interface

- How to transparently manage both middleboxes and endpoints?
- A flexible event interface can represent pkt., data and legacy events for a variety of services.
Microboxes $= \mu\text{Stack} + \mu\text{Event}$

- Idea #1: Consolidate Stack Processing
- Idea #2: Customizable Stack Modules
- Idea #3: Event Communication Interface
Outline

➢ Why Improve NFV Frameworks?
➢ Microboxes = µStack + µEvent
➢ µStack
  ▪ Customizable Modules
  ▪ Consistency Challenges
➢ µEvent
  ▪ Hierarchy Events
  ▪ Publish/Subscribe Interface
\(\mu\text{Stack Modules}\)

- We divide TCP processing into five basic \(\mu\)Stacks and they can be composed together to support different NFs.

![Diagram](image-url)
Layer 2/3: network layer processing to determine what flow it is associated with and maintain minimal state such as flow stats.
**μStack Modules**

- **TCP Monitor**: tracks the TCP state and reconstructs bytestream of both the client and server side of a connection.
**µStack Modules**

- **TCP Splicer**: redirects a TCP connection after establishing the handshake without support for modifying the bytestream.
**μStack Modules**

- **TCP Endpoint**: contains the full TCP logic and can terminate and respond to client requests directly.
μStack Modules

- **TCP Split Proxy**: sets up two TCP connections with client and server respectively and allows NFs to perform bytestream transformations.
Stack Consistency

- Stack and NFs are running on separate cores.
- Both Stack and NFs need to access the stack state.
Stack Consistency

- Stack state could be inconsistent when NF reads the state while stack has changed it based on new packet arrivals.

Stack Consistency: Protocol stack associated with each packet needs to be consistent when each NF processes this packet.
Stack Consistency

• Sequential processing can achieve the correctness but lead to an inefficient pipeline
Stack Consistency

- Only one core is doing useful work while others are idle
Stack Consistency: Stack Snapshots

- Take a snapshot of stack state for each packet to avoid inconsistency problem
Stack Consistency: Track Bytestream

- Store an offset instead of copying the whole bytestream
- Allow stack and NF processing to be performed asynchronously
**NF Consistency: Parallel Processing**

- Parallel processing increases core utilization and can be used for **NFs without dependencies** (NFP [SIGCOMM’17], Parabox [SOSR’17])
Flow Consistency: Parallel Stacks

- Run multiple copies of the same stack to maximize performance
- Packets are distributed at flow level to keep flow consistency
• Why Event Interface ???

• NFs operating at the application level care about “Data” instead of individual packets. An event can encapsulate the data and notify the subscribers
- Event types: pkt, flow, stack and NF events organized as a hierarchy.
- Event extension X/Y: a type can be extended to create a new type by adding extra fields in addition to parent fields.
**Event Hierarchy**

- Event Hierarchy simplifies type checking and filtering
- Set up publish/subscribe system: NF subscribes to portion of the hierarchy and gets the subevents.
Event Hierarchy simplifies type checking and filtering.
Set up subscribe/publish system: NF subscribe to portion of the hierarchy and gets the subevents.
µEvent Subscription

- Controller translates pub/sub into a flow table service chain rules to allow fast event propagation
- When an event is triggered, the NF looks up the flow table for the next hop associated with this event port

Pub/Sub architecture provides convenient, higher level interfaces based on the flow of events rather than the flow of packets
## Implementation

<table>
<thead>
<tr>
<th>L4 Load Balancer</th>
<th>L7 Load Balancer</th>
<th>TCP Proxy</th>
<th>nDPI</th>
<th>SIG Match</th>
<th>Flow Stats</th>
<th>Logger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>μEvent Pub/Sub Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>μStack Modules (mOS [NSDI’17], mTCP [NSDI’14])</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chain Management + NF Communication (OpenNetVM)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>NFV IO (DPDK)</th>
</tr>
</thead>
</table>
Evaluation

• Does Microboxes improve the performance by removing redundancy?

• Can Microboxes provide customized stacks based on application requirements?

• Experiment Setup: CloudLab
  • CloudLab Servers: Xeon E5-2660 v3 @ 2.60GHz CPUs (2 *10 cores) 10Gb NIC, 160GB memory
  • Traffic Generator: mTCP web server and client; Nginx 1.4.6 and Apache Bench 2.3
Evaluation: Remove Redundancy

Throughput (Gbps) vs. Chain Length

- Throughput decreases as the chain length increases.
- The 2nd Socket shows a significant drop in throughput.
Removing redundant stack processing can improve the performance by ~2X or more.
Evaluation: Customize Stack Modules

Diagram showing the flow of traffic from Clients 4K conns through L2 FWD or HAPerxy to nginx Web Server(s) 8KB file.
Evaluation: Customize Stack Modules

- Clients 4K conns
- L2/3 µStack
- L4 NF
- L7 LB
- L7 LB + 50% Cache
- L7 LB + 100% Cache
- Web Servers
- Web Server(s)
- 8KB file
- nginix
- DPDK L2 Fwd
- HAPerxy
- L4 LB
- Latency (us)
Evaluation: Customize Stack Modules

Clients
4K conns

L7 NF

Splicer μStack

nginx
Web Server(s)
8KB file

Latency (us)

Web Servers

Splicer µStack

Clients
4K conns

DPDK L2 Fwd

HAPerxy

L4 LB

L7 LB

8KB file

4K conns

4K conns

4K conns
Evaluation: Customize Stack Modules

Clients
4K conns

L7 NF
Splicer µStack
Endpoint µStack

50% Traffic

HAProxy
L7 LB
L7 LB + 50%

50% Traffic

Clients

Latency (us)

DPDK L2 Fwd
HAPerxy
L4 LB
L7 LB
L7 LB + 50% Cache

8KB file

nginx
Web Server(s)

Guyue Liu - George Washington University
Evaluation: Customize Stack Modules

Microboxes seamlessly integrates middleboxes and endpoints to build complex network services!
Source Code

https://github.com/sdnfv/openNetVM

• OpenNetVM now integrated with mOS/mTCP endpoint stack
  • Learn more at HPNFV tutorial (Friday 9:00AM – 5:20PM)

• Microboxes with µStack and µEvents code coming soon
Conclusion

✗ Redundant Stack Processing
✗ A Monolithic Stack
✗ Separate Stacks/Interfaces
✓ Consolidate Stack Processing
✓ Customizable Stack Modules
✓ Unified Event Interface

Microboxes
= \( \mu\text{Stack} + \mu\text{Event} \)
= stack snapshot + parallel stacks + parallel events + event hierarchy + publish/subscribe interface