OpenNF: Enabling Innovation in Network Function Control

Aaron Gember-Jacobson, Chaithan Prakash, Raajay Viswanathan, Robert Grandl, Junaid Khalid, Sourav Das, Aditya Akella
Network functions (NFs)

- Perform sophisticated *stateful* actions on packets/flows

- Intrusion detection system (IDS)
- Caching proxy
- WAN optimizer
- Intrusion detection system (IDS)
NF trends

• NFV → dynamically allocate NF instances
NF trends

• NFV → dynamically allocate NF instances
NF trends

- NFV → dynamically allocate NF instances
- SDN → dynamically reroute flows
NF trends

- NFV → dynamically allocate NF instances
- SDN → dynamically reroute flows
NF trends

- NFV → dynamically allocate NF instances
- SDN → dynamically reroute flows

Dynamic reallocation of packet processing
Example: elastic NF scaling
Example: elastic NF scaling

1. Satisfy performance SLAs
Example: elastic NF scaling

1. Satisfy performance SLAs
Example: elastic NF scaling

1. Satisfy performance SLAs
Example: elastic NF scaling

1. Satisfy performance SLAs
2. Minimize operating costs
Example: elastic NF scaling

1. Satisfy performance SLAs
2. Minimize operating costs

CPU
Packet loss
Example: elastic NF scaling

1. Satisfy performance SLAs
2. Minimize operating costs
3. Accurately monitor traffic
Problem: NFV+SDN is insufficient

To simultaneously...
1. Satisfy performance SLAs
2. Minimize operating costs
3. Accurately monitor traffic

Cannot effectively implement new services or abstractions!
Why NFV + SDN falls short

1. SLAs
2. Cost
3. Accuracy

Packet loss

---
Why NFV + SDN falls short

Packet loss

SLA: <1%

1. SLAs  2. Cost  3. Accuracy
Why NFV + SDN falls short

1. SLAs  2. Cost  3. Accuracy

Packet loss

SLA: <1%
Why NFV + SDN falls short

Packet loss

SLA: <1%

1. SLAs  2. Cost  3. Accuracy

Reroute new flows

[Stratos - arXiv:1305.0209]

Packet loss

Reroute existing flows

[SIMPLE - SIGCOMM '13]
Why NFV + SDN falls short

1. SLAs
2. Cost
3. Accuracy

Reroute new flows
[Stratos - arXiv:1305.0209]

Reroute existing flows
[SIMPLE - SIGCOMM ‘13]
Why NFV + SDN falls short

1. SLAs
2. Cost
3. Accuracy

Packet loss

Reroute new flows
[Stratos - arXiv:1305.0209]

Reroute existing flows
[SIMPLE - SIGCOMM '13]
Why NFV + SDN falls short

1. SLAs  2. Cost  3. Accuracy

Packet loss

Reroute new flows
[Stratos - arXiv:1305.0209]

Reroute existing flows
[SIMPLE - SIGCOMM ’13]
Why NFV + SDN falls short

1. SLAs
2. Cost
3. Accuracy

<table>
<thead>
<tr>
<th>Reroute new flows</th>
<th>✔️</th>
<th>✔️</th>
<th>❓</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Stratos - arXiv:1305.0209]</td>
<td>❌</td>
<td>✔️</td>
<td>❓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reroute existing flows</th>
<th>✔️</th>
<th>✔️</th>
<th>❌</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SIMPLE - SIGCOMM '13]</td>
<td>✔️</td>
<td>✔️</td>
<td>❌</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wait for flows to die</th>
<th>✔️</th>
<th>❌</th>
<th>✔️</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Stratos - arXiv:1305.0209]</td>
<td>✔️</td>
<td>❌</td>
<td>✔️</td>
</tr>
</tbody>
</table>
SLAs + cost + accuracy: What do we need?

- Quickly move, copy, or share internal NF state alongside updates to network forwarding state

- Guarantees: loss-free, order-preserving, ...

Also applies to other scenarios
Outline

• Motivation and requirements

• Challenges

• OpenNF architecture
  – State export/import
  – State operations
  – Guarantees

• Evaluation
Challenges

1. Supporting many NFs with minimal changes

2. Dealing with race conditions

3. Bounding overhead
OpenNF overview

Control Application

move/copy/share state

NF State Manager

export/import State

Flow Manager
NF state taxonomy

State created or updated by an NF applies to either a **single flow** or a **collection of flows**
NF API: export/import state

- Functions: get, put, delete
NF API: export/import state

- Functions: get, put, delete

Scope: Per, Multi, All

Filter

NF
NF API: export/import state

- Functions: get, put, delete
NF API: export/import state

- Functions: get, put, delete

No need to expose/change internal state organization!
Control operations: move

move \( (\text{port}=80, \text{Bro}_1, \text{Bro}_2) \)

Control Application \hspace{2cm} \text{Flow Manager}

NF State Manager

\text{Bro}_1 \hspace{2cm} \text{Bro}_2
Control operations: move

**Control Application**

- `move (port=80, Bro₁, Bro₂)`

**NF State Manager**

- `get(per, port=80)`

**Flow Manager**

- Bro₁
- Bro₂
Control operations: move

Control Application

Flow Manager

move (port=80, Bro₁, Bro₂)

get(per, port=80)

[Chunk1]

[Chunk2]

Bro₁

Bro₂
Control operations: move

- **move** \( (port=80, \text{Bro}_1, \text{Bro}_2) \)
- **get** \( (per, port=80) \)
- **del** \( (per, port=80) \)

[Chunk1] [Chunk2]
Control operations: move

Control Application

move (port=80, Bro₁, Bro₂)

get(per, port=80)

del(per, port=80)

NF State Manager

[Chunk1]

[Chunk2]

Flow Manager

put (per, Chunk1)

put (per, Chunk2)
Control operations: move

Control Application

move (port=80, Bro_1, Bro_2)
get(per, port=80)
del(per, port=80)

NF State Manager

[Chunk1]
[Chunk2]

[Flow Manager]

forward(port=80, Bro_2)
put (per, Chunk1)
put (per, Chunk2)

Bro_1

Bro_2
Control operations: move

Control Application

move (port=80, Bro₁, Bro₂)

get(per, port=80)

del(per, port=80)

Flow Manager

forward(port=80, Bro₂)

NF State Manager

[Chunk1]

[Chunk2]

Bro₁

Bro₂

Also provide copy and share
Lost updates during move
Lost updates during move
Lost updates during move

\[ \text{move}(\text{red}, \text{Bro}_1, \text{Bro}_2) \]
Lost updates during move

$move(red,Bro_1,Bro_2)$
Lost updates during move

\[ \text{move(} \text{red}, \text{Bro}_1, \text{Bro}_2 \text{)} \]
Lost updates during move

\[ \text{move}(\text{red, } \text{Bro}_1, \text{Bro}_2) \]
Lost updates during move

\[\text{move}(\text{red}, \text{Bro}_1, \text{Bro}_2)\]

Missing state
Lost updates during move

\[ \text{move}(\text{red}, \text{Bro}_1, \text{Bro}_2) \]

Missing state

R2 → Bro_1

R1 ← Bro_2

Missing updates

detect-MHR
Lost updates during move

move(red, Bro₁₁, Bro₂₂)

Missing state

B1 → Bro₁ → Bro₂ → R1

Loss-free: All state updates should be reflected in the transferred state, and all packets should be processed

Split/Merge [NSDI ’13]: pause traffic, buffer packets

– Packets in-transit when buffering starts are dropped
NF API: observe/prevent updates using events
NF API: observe/prevent updates using events
NF API: observe/prevent updates using events
NF API: observe/prevent updates using events
NF API: observe/prevent updates using events
NF API: observe/prevent updates using events

Only need to change an NF’s receive packet function!
Use events for loss-free move
Use events for loss-free move

1. `enableEvents(red, drop)` on `Bro_1`
Use events for loss-free move

1. `enableEvents(red, drop)` on Bro$_1$
2. `get/delete` on Bro$_1$
Use events for loss-free move

1. `enableEvents(red, drop) on Bro_1`
2. `get/delete on Bro_1`
Use events for loss-free move

1. enableEvents(red, drop) on Bro₁
2. get/delete on Bro₁
3. Buffer events at controller
Use events for loss-free move

1. `enableEvents(red, drop)` on Bro₁
2. `get/delete` on Bro₁
3. Buffer events at controller
4. `put` on Bro₂
Use events for loss-free move

1. `enableEvents(red, drop)` on $\text{Bro}_1$
2. `get/delete` on $\text{Bro}_1$
3. Buffer events at controller
4. `put` on $\text{Bro}_2$
5. Flush packets in events to $\text{Bro}_2$
Use events for loss-free move

1. `enableEvents(red, drop)` on `Bro_1`
2. `get/delete` on `Bro_1`
3. Buffer events at controller
4. `put` on `Bro_2`
5. Flush packets in events to `Bro_2`
6. Update forwarding
Use events for loss-free move

1. `enableEvents(red, drop) on Bro_1`
2. `get/delete` on `Bro_1`
3. Buffer events at controller
4. `put` on `Bro_2`
5. Flush packets in events to `Bro_2`
6. Update forwarding
Re-ordering of packets

- False positives from Bro’s weird script
Re-ordering of packets

• False positives from Bro’s weird script

5. Flush buffer
Re-ordering of packets

• False positives from Bro’s weird script

Bro\textsubscript{1}

5. Flush buffer
6. Request forwarding update

Bro\textsubscript{2}

Controller

Switch

R2

R2

R2
Re-ordering of packets

• False positives from Bro’s weird script

5. Flush buffer
6. Request forwarding update
Re-ordering of packets

- False positives from Bro’s weird script

5. Flush buffer
6. Request forwarding update
Re-ordering of packets

- False positives from Bro’s weird script

5. Flush buffer
6. Request forwarding update
Re-ordering of packets

- False positives from Bro’s weird script

Order-preserving: All packets should be processed in the order they were forwarded by the switch
OpenNF: SLAs + cost + accuracy

1. Dealing with diversity

Export/import state based on its association with flows

2. Dealing with race conditions

Events + Lock-step forwarding updates
Implementation

• Controller (3.8K lines of Java)
• Communication library (2.6K lines of C)
• Modified NFs (3-8% increase in code)
Overall benefits for elastic scaling

• Bro IDS processing 10K pkts/sec
  – At 180 sec: move HTTP flows (489) to new IDS
  – At 360 sec: move back to old IDS
• SLAs: 260ms to move (loss-free)
• Accuracy: same log entries as using one IDS
  – VM replication: incorrect log entries
• Cost: scale down after state is moved
  – Stratos: scale down delayed 25+ minutes

[arXiv:1305.0209]
Evaluation: state export/import

Serialization/deserialization costs dominate

Cost grows with state complexity
Evaluation: operations

- PRADS asset detector processing 5K pkts/sec
- Move per-flow state for 500 flows

Operations are efficient, but guarantees come at a cost!
Conclusion

• Dynamic reallocation of packet processing enables new services

• Realizing SLAs + cost + accuracy requires quick, safe control of internal NF state

• OpenNF provides flexible and efficient control with few NF modifications

http://opennf.cs.wisc.edu