OpenNF: Enabling Innovation in Network Function Control

Aaron Gember-Jacobson, Chaithan Prakash, Raajay Viswanathan, Robert Grandl, Junaid Khalid, Sourav Das, Aditya Akella

1

Network functions (NFs)

- Perform sophisticated *stateful* actions on packets/flows
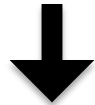

WAN optimizer

Caching proxy
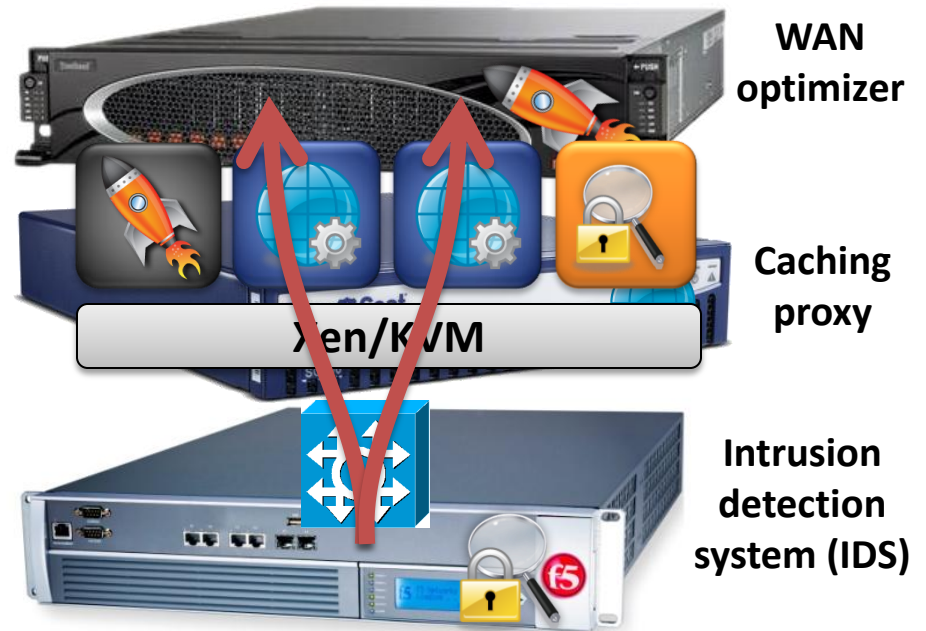
Intrusion detection system (IDS)

NF trends

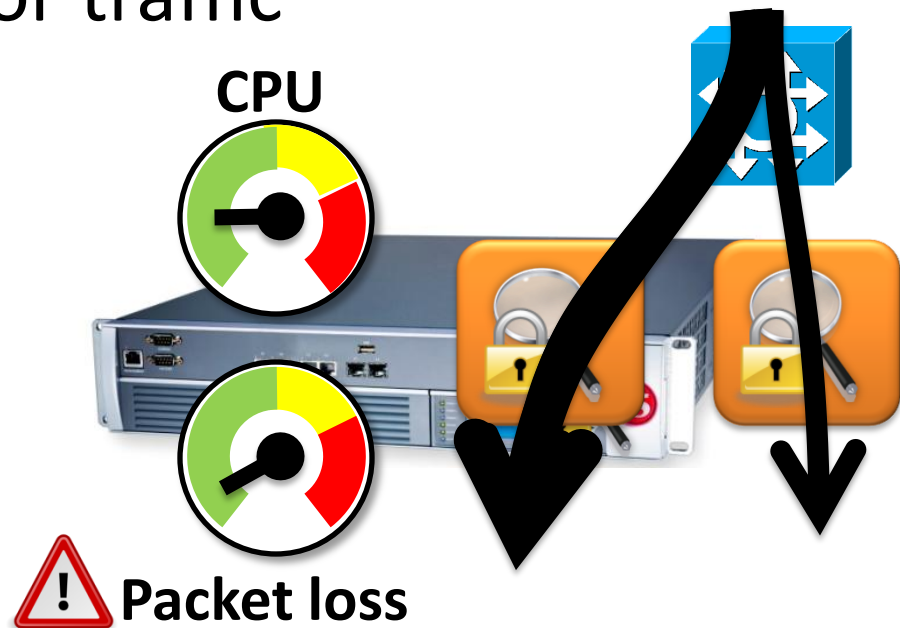- NFV → dynamically allocate NF instances

+

- SDN → dynamically reroute flows

↓

Dynamic reallocation
of packet processing

WAN
optimizer

Xen/KVM

Caching
proxy

Intrusion
detection
system (IDS)

Example: elastic NF scaling

1. Satisfy performance SLAs

2. Minimize operating costs

3. Accurately monitor traffic

CPU

Packet loss

Problem: NFV + SDN is insufficient

To **simultaneously**…

1. Satisfy performance SLAs

2. Minimize operating costs

3. Accurately monitor traffic

CPU

Cannot effectively implement
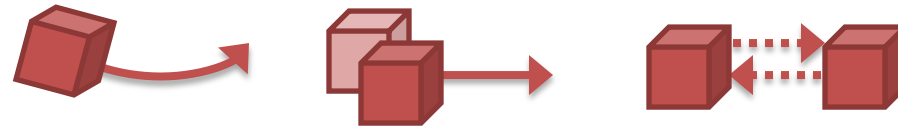new services or abstractions!

Packet loss

5

Why NFV + SDN falls short

Packet loss

SLA: <1%

1. SLAs   2. Cost   3. Accuracy
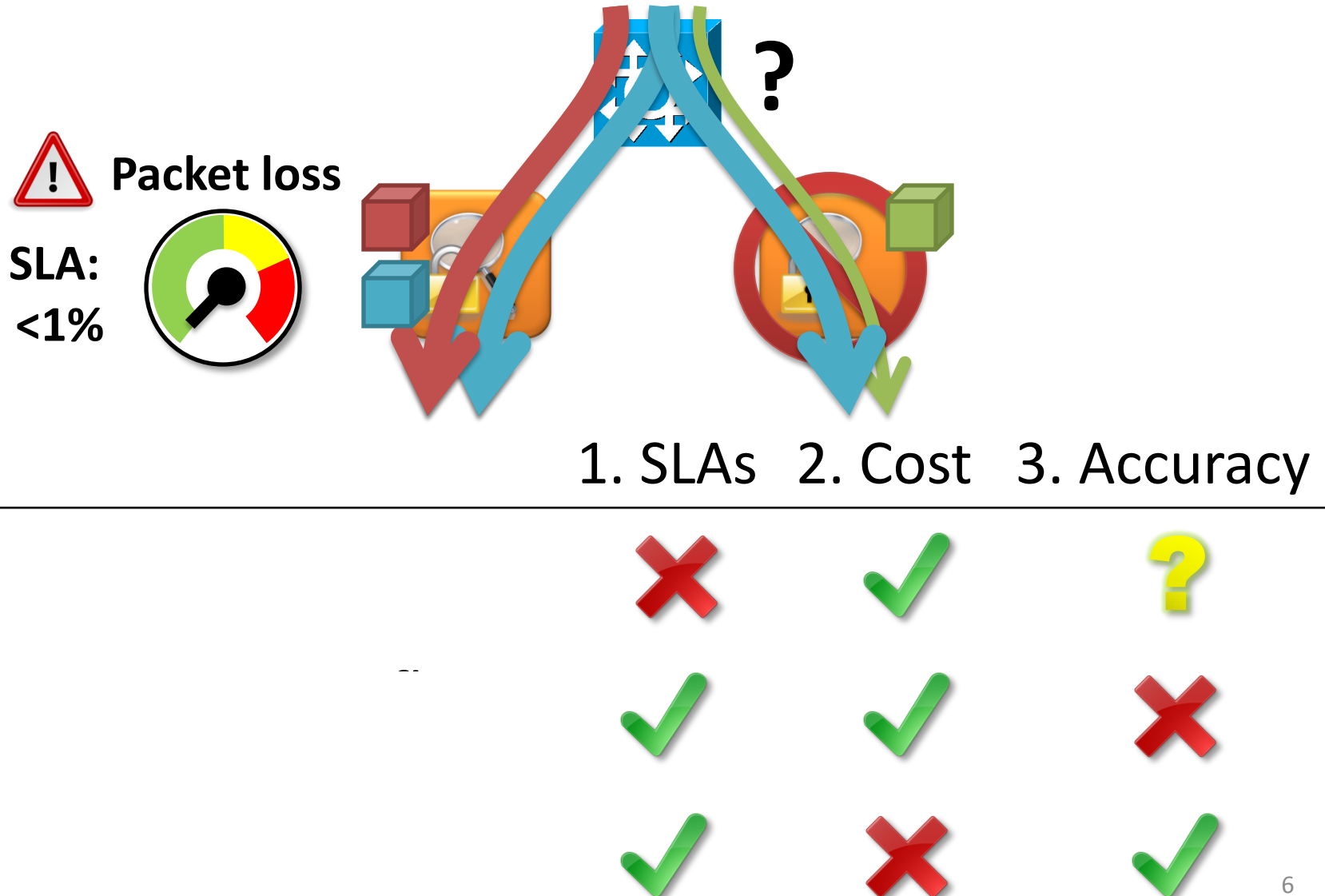
SLAs + cost + accuracy: What do we need?

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

- Guarantees: loss-free, order-preserving, …
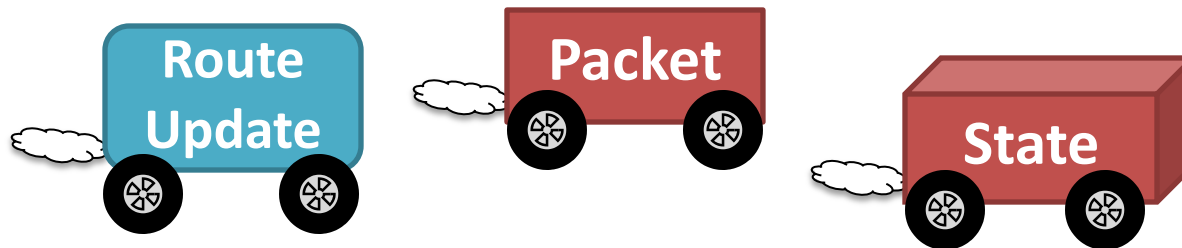
Also applies to other scenarios

Outline

- Motivation and requirements
- Challenges
- OpenNF architecture
  - State export/import
  - State operations
  - Guarantees
- Evaluation

Challenges

1. Supporting many NFs with minimal changes
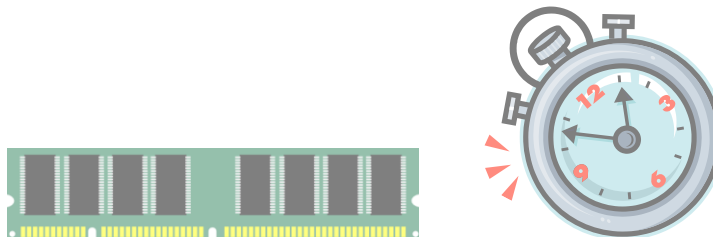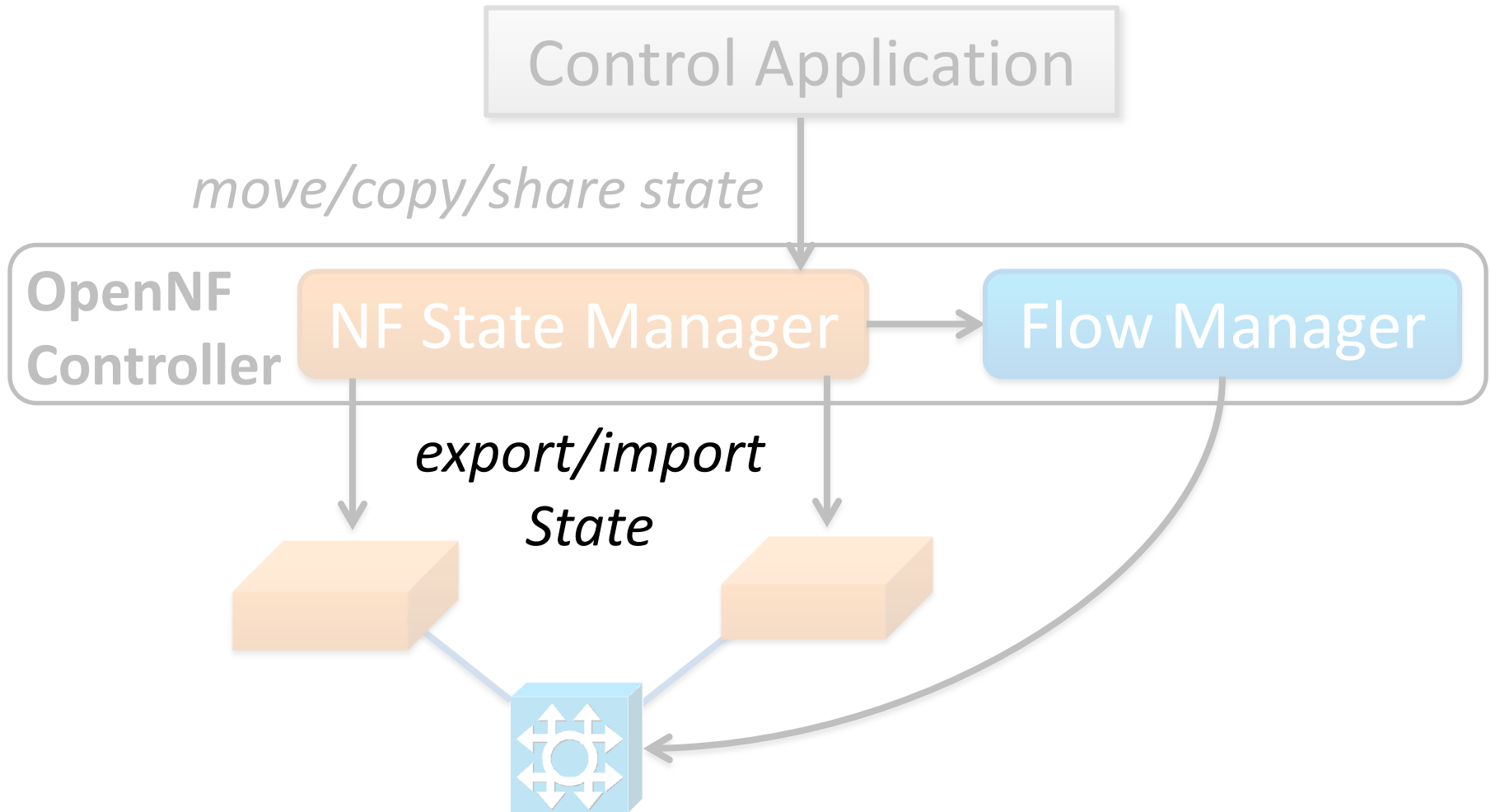
2. Dealing with race conditions

Route Update

Packet

State

3. Bounding overhead

OpenNF overview

Control Application

move/copy/share state
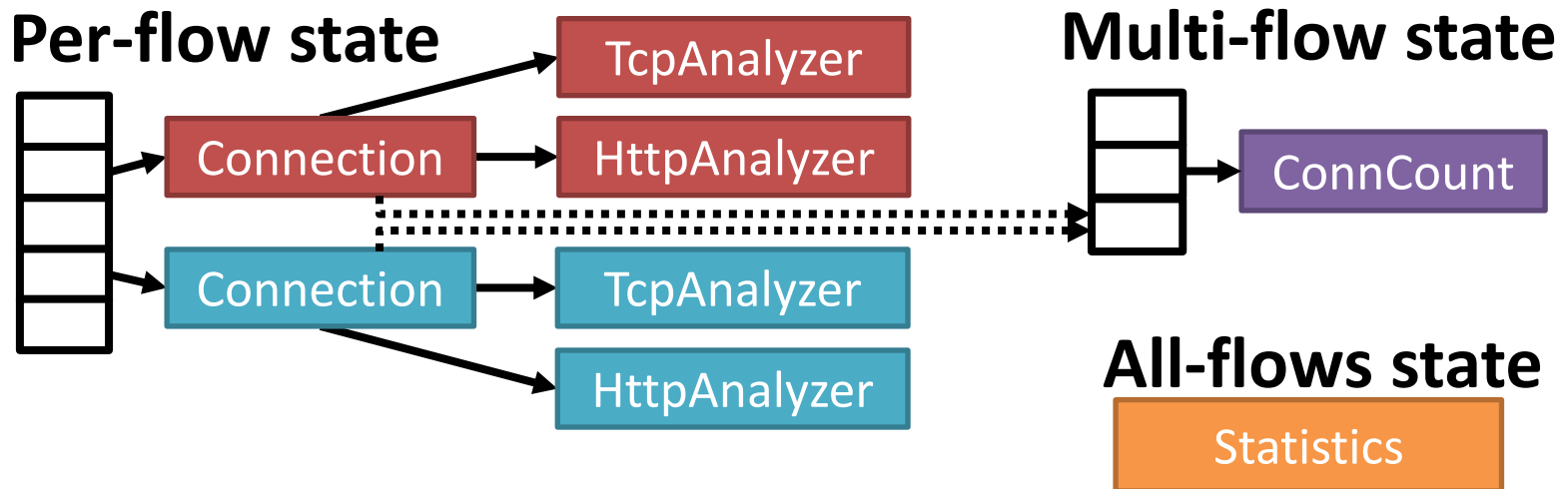
OpenNF Controller

NF State Manager

Flow Manager

export/import State

NF state taxonomy

State created or updated by an NF applies to either a **single flow** or a **collection of flows**

Per-flow state

TcpAnalyzer

Connection

HttpAnalyzer

Connection

TcpAnalyzer

HttpAnalyzer

Multi-flow state

ConnCount

All-flows state

Statistics

NF API: export/import state

- Functions: `get`, `put`, `delete`

Scope

Per

Multi

All

put

Filter

NF

get
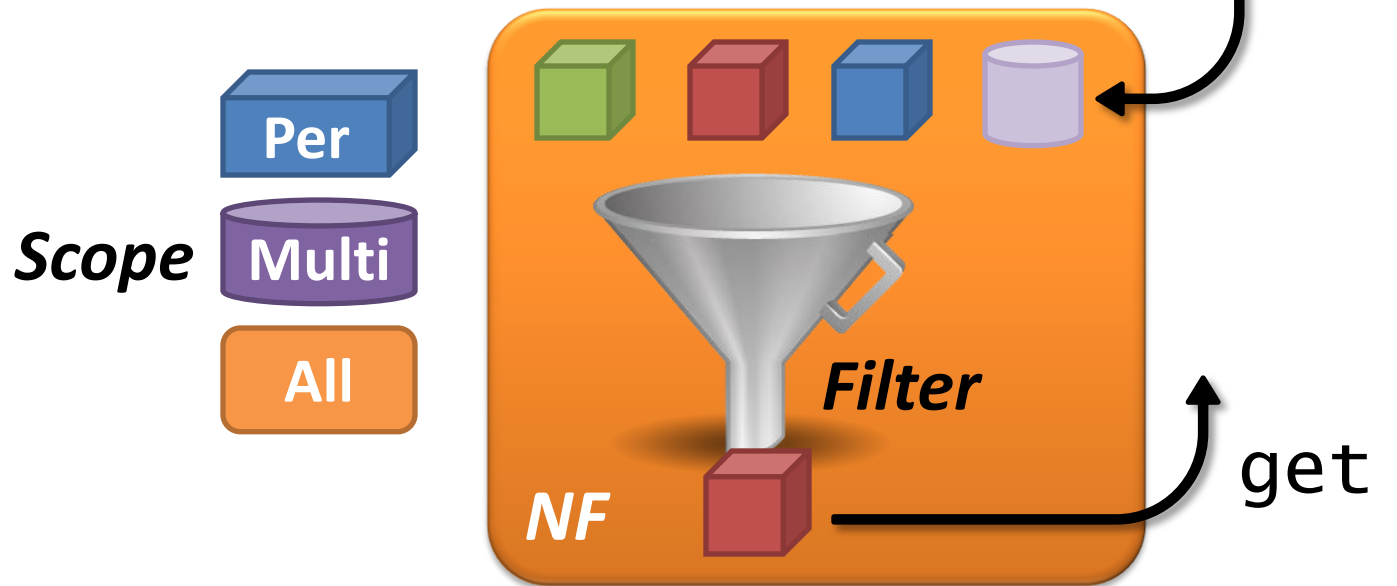
No need to expose/change internal state organization!

Control operations: move

Control Application

Flow Manager

move (port=80, Bro_1, Bro_2)

forward(port=80, Bro_2)

NF State Manager

get(per, port=80)

del(per, port=80)

[Chunk1]

[Chunk2]

put (per, Chunk1)

put (per, Chunk2)

Bro_1

Bro_2
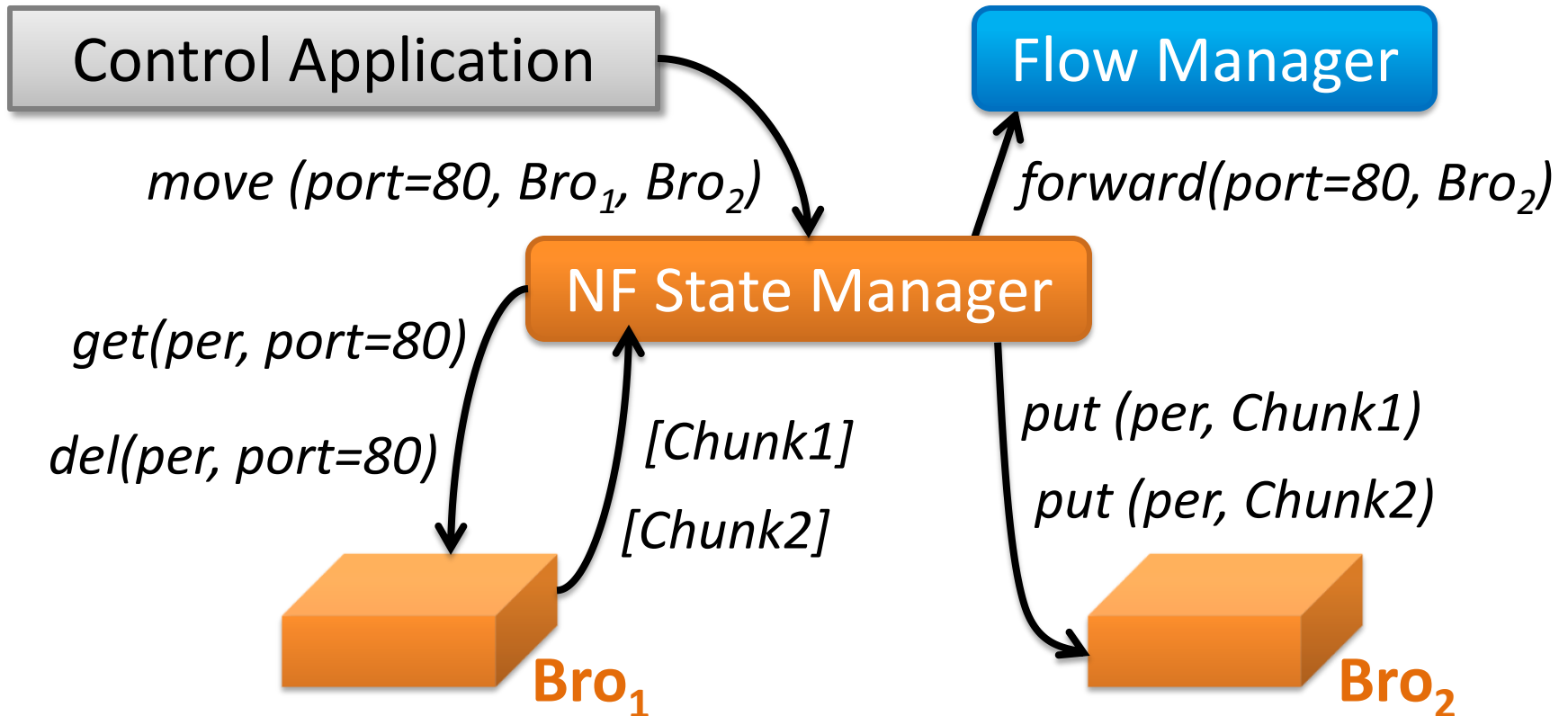
Also provide copy and share

Lost updates during move

$move(red, Bro_1, Bro_2)$

Missing state

Missing updates

R3

detect-MHR

R2

B1

Bro_1

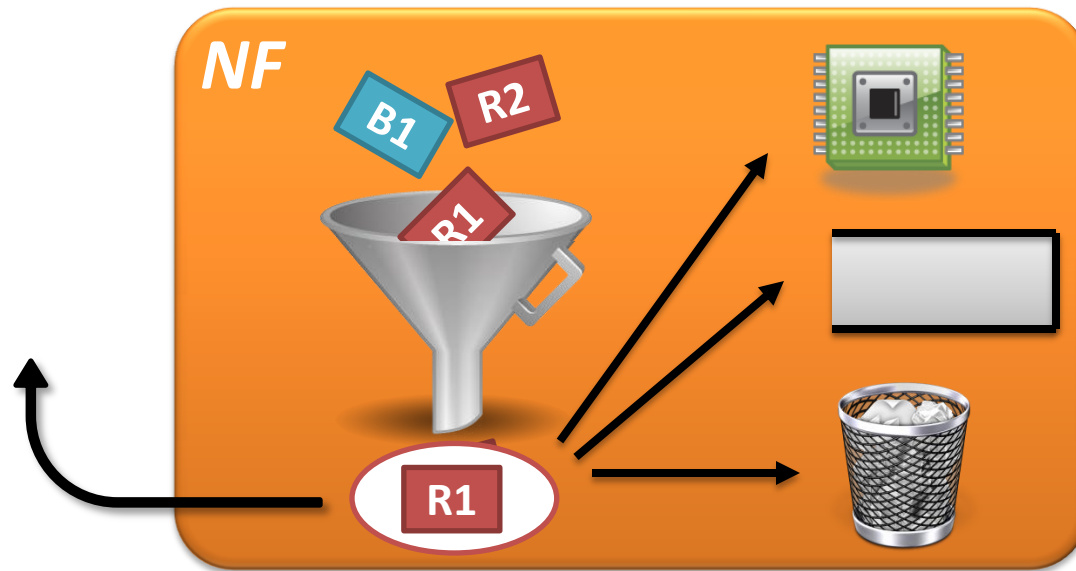Bro_2
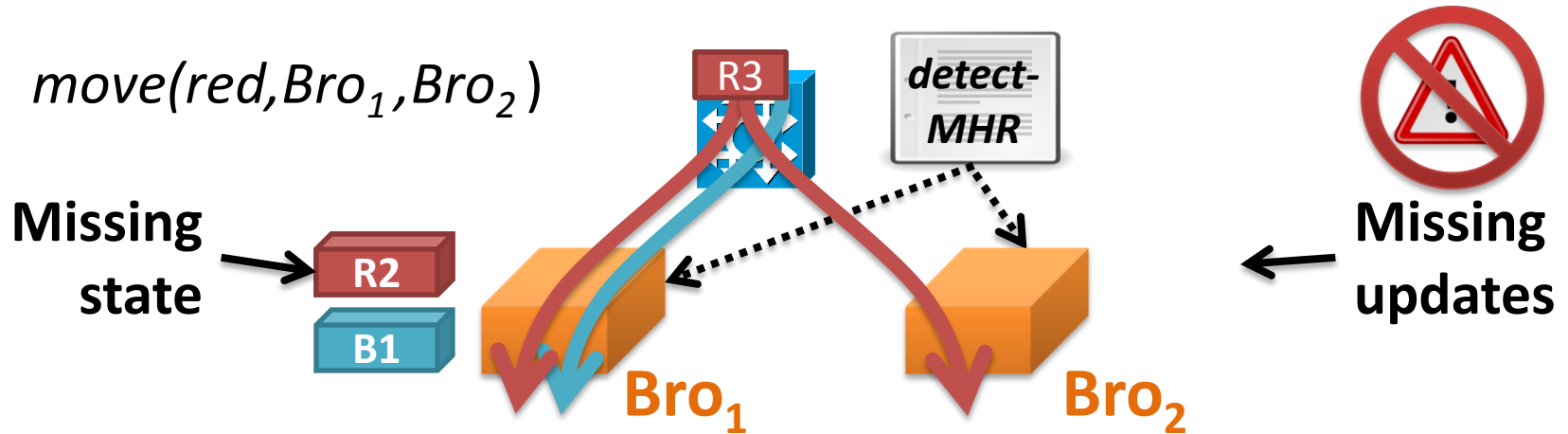
Loss-free: All state updates should be reflected in the transferred state, and all packets should be processed

❌ Split/Merge *[NSDI '13]*: pause traffic, buffer packets
  – Packets in-transit when buffering starts are dropped
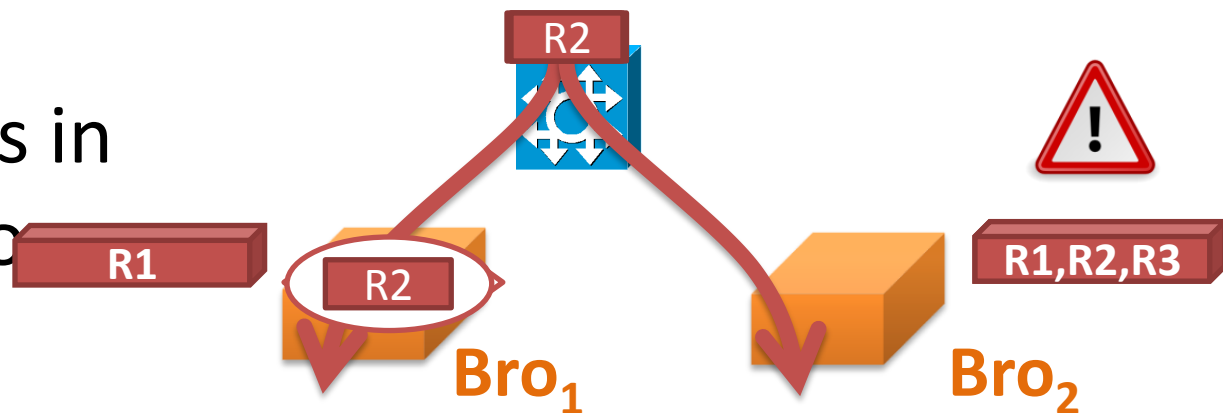
NF API: observe/prevent updates using events

Only need to change an NF's receive packet function!

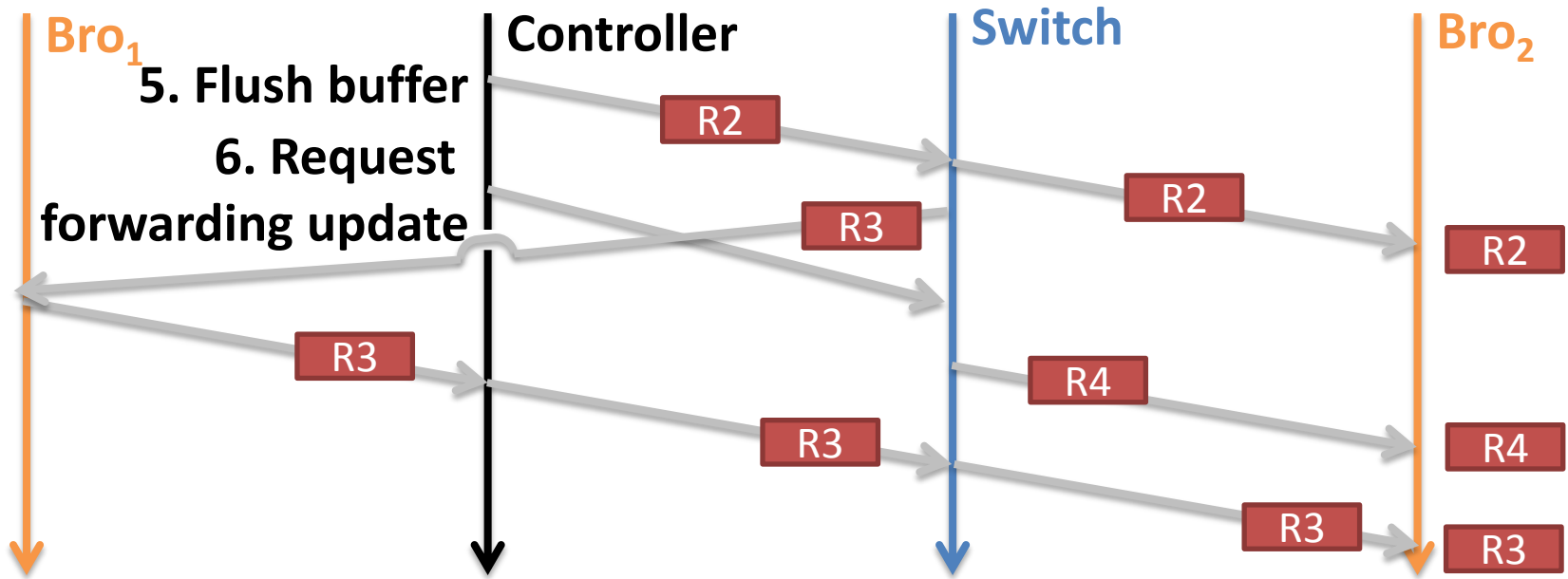# Use events for loss-free move

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

6. Update forwarding

R2

R1

R2

Bro_1

R1,R2,R3

Bro_2

Re-ordering of packets

- False positives from Bro's `weird` script

Bro₁ **Controller** **Switch** **Bro₂**

5. Flush buffer
6. Request forwarding update

R2 R2 R2
R3 R3
R3 R4 R4
R3 R3 R3

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)

**Bro IDS**   **iptables**   **Squid Cache**   **PRADS**

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]
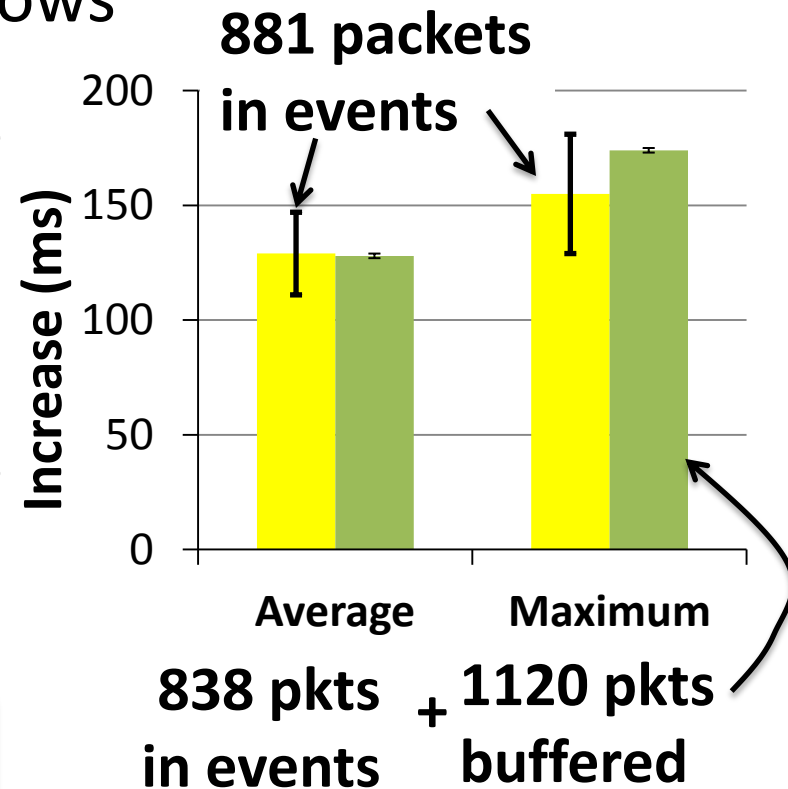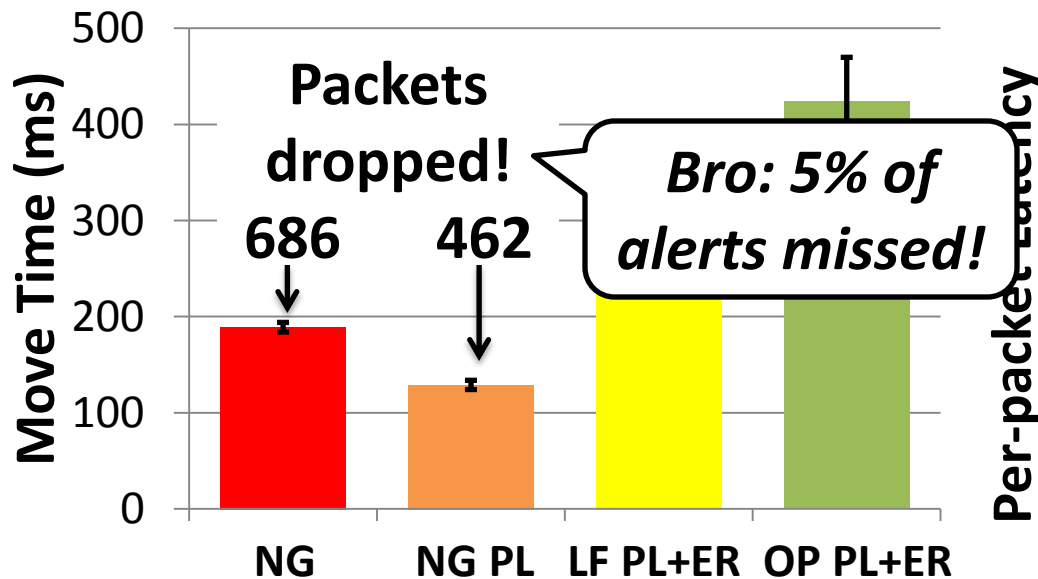
Evaluation: state export/import
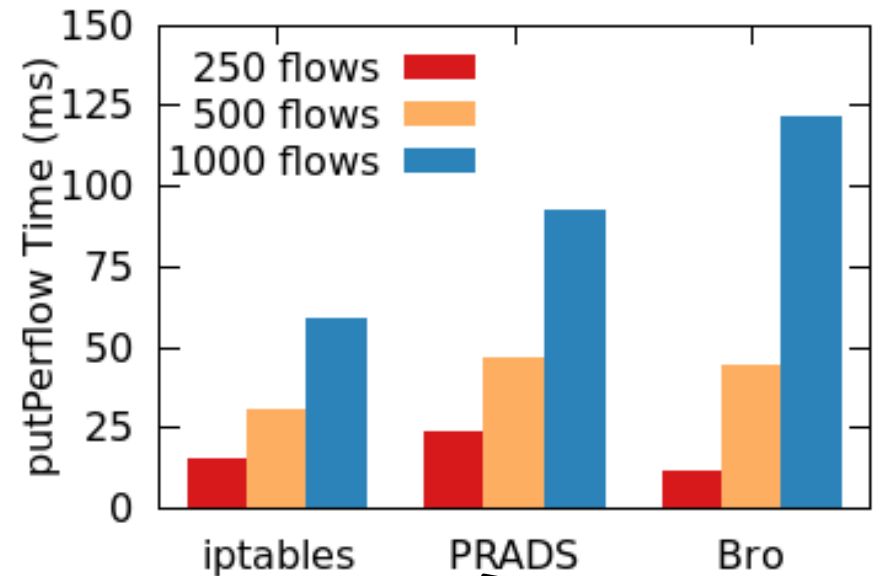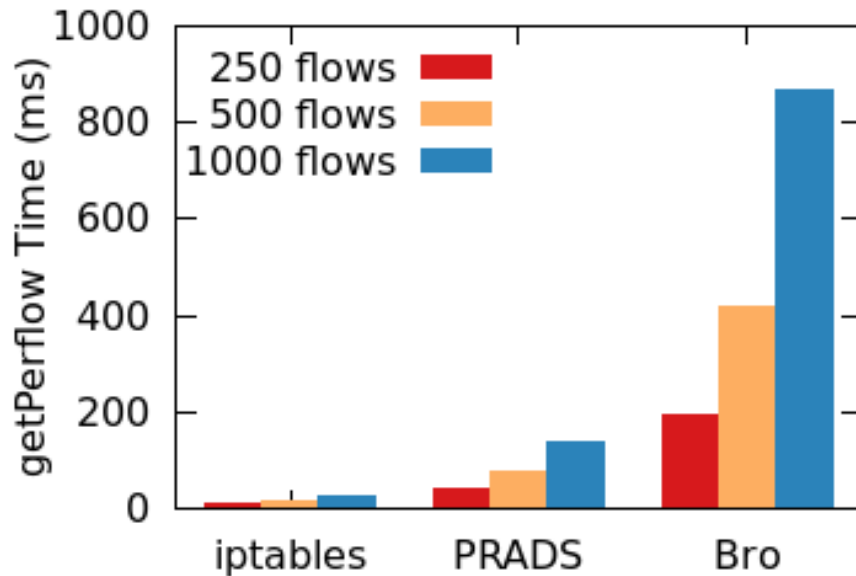
Serialization/deserialization costs dominate

Cost grows with state complexity

Evaluation: operations

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

Packets dropped!

686   462

Bro: 5% of alerts missed!

881 packets in events

838 pkts in events + **1120 pkts buffered**

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
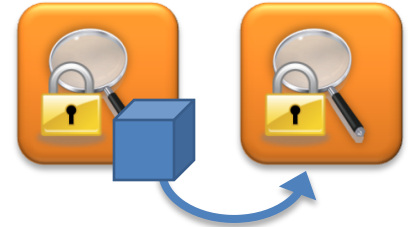
# **Backup**

- [Related work](#)
- [Copy and share](#)
- [Order-preserving move](#)
- [Bounding overhead](#)
- [Example control application](#)
- [Evaluation: controller scalability](#)
- [Evaluation: importance of guarantees](#)
- [Evaluation: benefits of granular control](#)

Existing approaches

- Virtual machine replication
  - Unneeded state → incorrect actions
  - Cannot combine → limited reallocation

- Split/Merge *[NSDI'13]*
  - State allocations and accesses occur via library
  - Addresses a specific problem → limited suitability
  - Packets may be dropped or re-ordered → wrong NF behavior
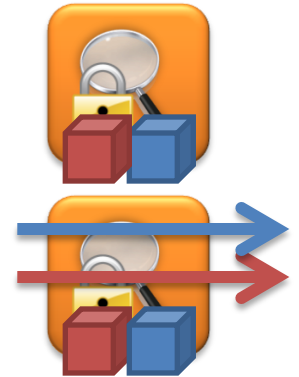
Copy and share operations

- Used when multiple instances need some state
- Copy – no or eventual consistency
  - Once, periodically, based on events, etc.
- Share – strong or strict consistency
  - Events are raised for all packets
  - Events are released one at a time
  - State is copied before releasing the next event

Copy (multi-flow): 111ms
Share (strong): 13ms/packet

Order-preserving move

- Flush packets in events to $Inst_2$

- `enableEvents(blue,buffer)` on $Inst_2$

- ***Forwarding update:*** send to $Inst_1$ & controller

- Wait for packet from switch (remember last)

- ***Forwarding update:*** send to $Inst_2$

- Wait for event for last packet from $Inst_2$

- Release buffer of packets on Inst

B4

Buf

B1,B2, B3,B4

B3

B1

B2

Bounding overhead

Applications decide (based on NF & objectives):

1. Granularity of operations

Filter

Per

Multi

All   **Scope**
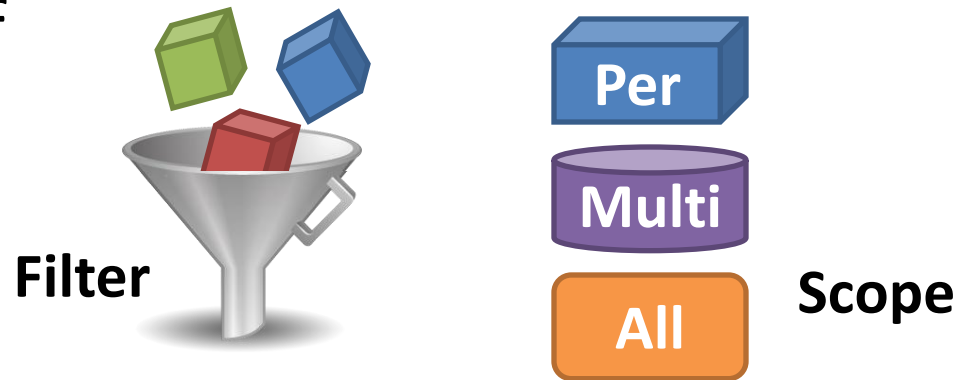
2. Guarantees desired

None

LF
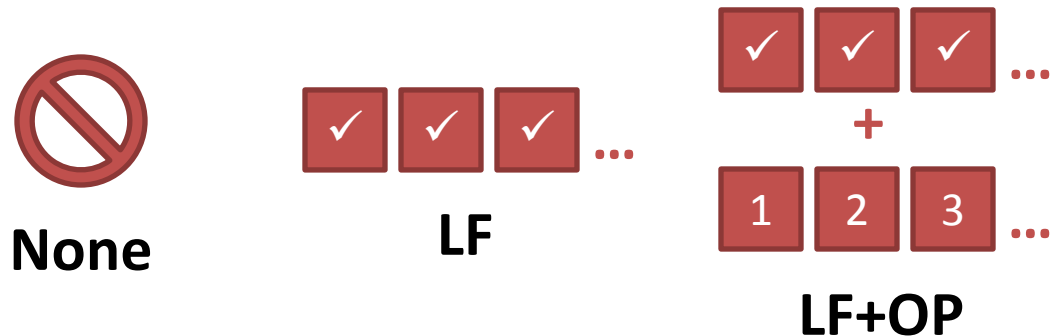
LF+OP

Example app: elastic NF scaling

scan.bro
vulnerable.bro
weird.bro
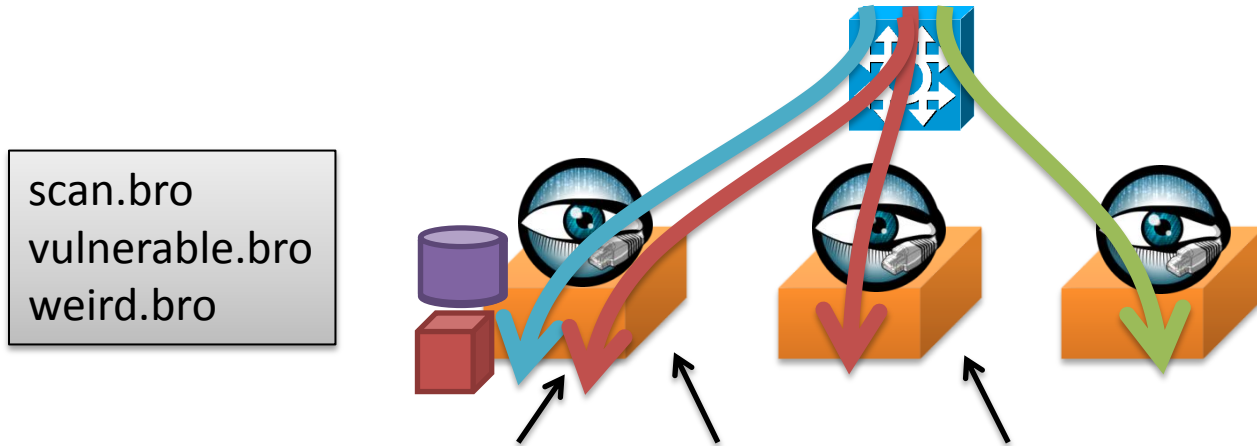
```
movePrefix(prefix,oldInst,newInst):
  copy(oldInst,newInst,{nw_src:prefix},multi)
  move(oldInst,newInst,{nw_src:prefix},per,LF+OP)
  while (true):
    sleep(60)
    copy(oldInst,newInst,{nw_src:prefix},multi)
    copy(newInst,oldInst,{nw_src:prefix},multi)
```

Evaluation: controller scalability
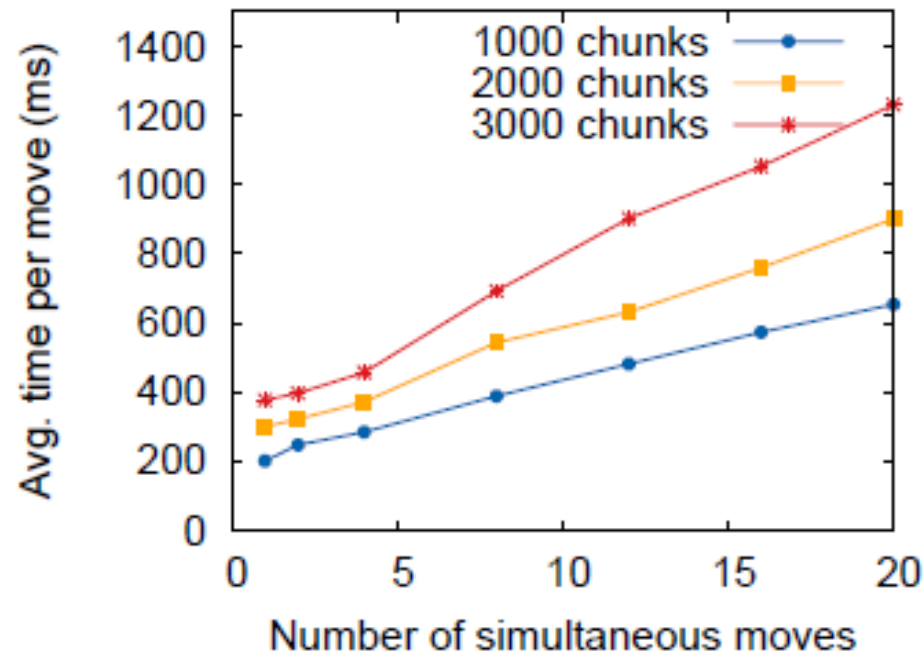
Improve scalability with P2P state transfers

Evaluation: importance of guarantees

- Bro$_1$ processing malicious trace @ 1K pkts/sec
- After 14K packets: move active flows to Bro$_2$

Alert	Baseline	NF	LF	LF+OP
Incorrect file type	26	25	24	26
MHR Match	31	28	27	31
MD5	116	111	106	116
Total	173	164	157	173

↵

Evaluation: benefits of granular control

- HTTP requests from 2 clients (40 unique URLs)
- Initially: both go to $Squid_1$
- 20s later: reassign $Client_1$ to $Squid_2$

	Ignore	Copy-client	Copy-all
Hits @ $Squid_1$	117	117	117
Hits @ $Squid_2$	Crash!	39	50
State transferred	0 MB	4 MB	54 MB

↩