OpenNetVM Hands-On

ONVM Basics

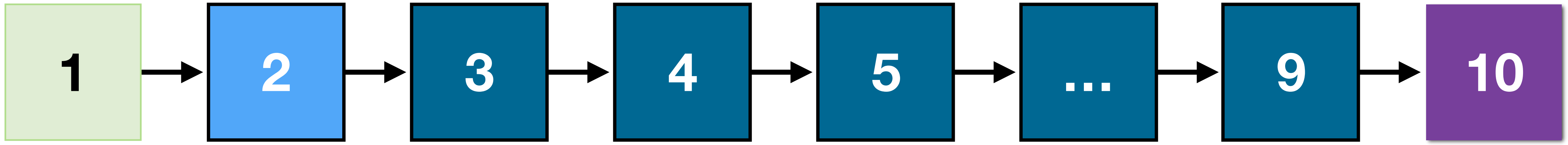
- Clone the repository: http://github.com/sdnfv/openNetVM
- Build and install DPDK
  - Included with the repository; may work with other versions
- Build ONVM manager and examples
- Run the NF Manager
- Run one or more NFs
- Send some packets!

Testbed Setup

Find info at: https://github.com/sdnfv/onvm-tutorial

10 servers connected in a chain

Already have ONVM/DPDK installed and configured



Our goal:

- Send traffic from node 1 to node 10
- Forward all packets through the switches using **OpenNetVM**
- Measure performance of NF communication

Setting up Environment

(All of this has been done already)

Export shell variables for important paths

Allocate memory for huge page region

Bind network interfaces to the DPDK driver

ONVM provides a **`setup_environment.sh`** script to automate most of these tasks

ONVM Code Structure

`openNetVM/onvm/onvm_mgr`

- NF Manager code

`openNetVM/onvm/onvm_nf`

- NFLib API used by NFs

`openNetVM/onvm/shared`

- Code shared by both Manager and NFs

`openNetVM/dpdk/`

- Git submodule with DPDK library

`openNetVM/examples/`

- Sample NF code

NF Manager

3

NF Manager

- Receives packets using DPDK (RX threads)
- Distributes packets based on a flow table (RX/TX threads)
- Tracks active NFs (Stats thread)

```
cd $ONVM_HOME/onvm
./go.sh  0,1,2 3 -s stdout
# usage: ./go.sh CORE_LIST PORT_LIST
```

Core list: set of cores used for manager threads

- **0**: stats/managerment, **1**: TX thread, **2**: RX thread
- Current release assumes all cores are on same CPU socket
- Can adjust number of RX threads with compile time macro

Port list: bitmap specifying ports (e.g., 3=0b11=ports 0 and 1)

Speed Tester NF

4

A simple throughput tester NF

Creates a batch of packets and sends them to an NF

```
cd $ONVM_HOME/examples/speed_tester
./go.sh 4 1 1
# usage: ./go.sh CORE_LIST NF_ID DEST_ID
# (be sure the manager is already running)
Total packets: 170000000
TX pkts per second:  21526355
Packets per group: 128
```

Send to self to benchmark manager's TX threads

(Doesn't use networking, just local packet processing)

Performance Characteristics

What affects NF performance?

Amount of computation performed per packet

- Hopefully very very little

RX threads - read packets from the NIC

- By default only use one RX thread
- One thread can handle ~7Gbps of 64 byte packets
- Two RX threads can handle ~70Gbps of "real" traffic

NFs - Process packets and pass to next NF or to TX thread

- Speed tester is best case: ~50 Million packets per second

TX threads - send packets out NIC

Lifecycle of an NF

onvm_nf_init - register NF with manager

- NF specifies its Service ID
- Manager returns an Instance ID

onvm_nf_run - start packet processing loop

- NF will poll a shared ring buffer until an RX or TX thread gives it a batch of packets

packet_handler - custom packet processing

- Called for each incoming packet
- Implements the application logic for the NF

Packet Handler

Read or write packet data

Read or write packet meta data

- Action, service chain position, flags, user data

Packet actions - specified by NF for each packet

- **TONF**: send to another NF
- **OUT**: send out a NIC port
- **NEXT**: use action in flow table
- **DROP**: discard packet

Service Types

Each NF starts with a user specified Service ID

- e.g., 1=Firewall, 2=Proxy, 3=...

Each NF is assigned a unique Instance ID

Service types are used for addressing (TONF action)

The manager uses the Instance ID to load balance across replicas with the same Service ID

- The RSS hash of the packet (based on n-tuple) is used to pick an instance in a consistent way for all packets in a flow

By default: all RX packets are delivered to Service 1

- Can be changed by modifying the default Service Chain

Bridge NF

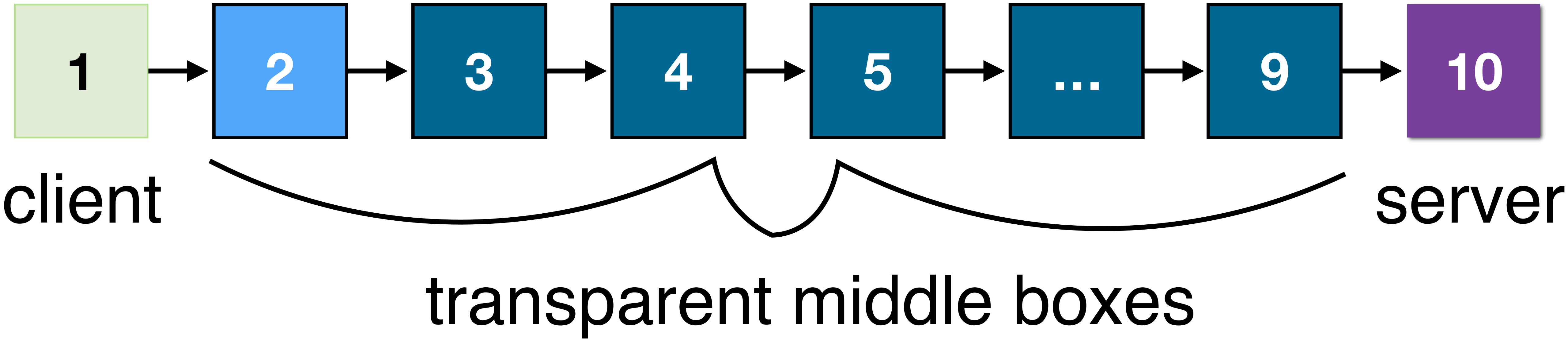
5

Bridges two NIC ports

Useful for chaining servers

```
# Terminal 1: Run manager with web console
./go.sh 0,1,2 3 -s web
```

```
# Terminal 2: Start bridge NF
cd $ONVM_HOME/examples/bridge
./go.sh 4 1
```



1

2

3

4

5

...

9

10

client

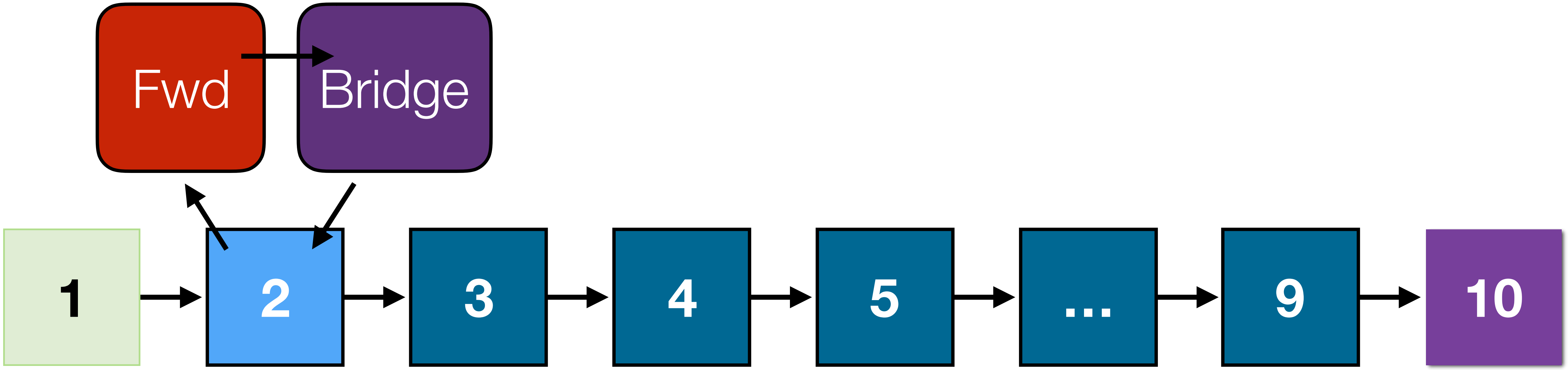
server

transparent middle boxes

NF Chains

OpenNetVM is primarily designed to facilitate service chaining **within** a server.

- NFs can specify whether packets should be sent out the NIC or delivered to another NF based on a service ID number

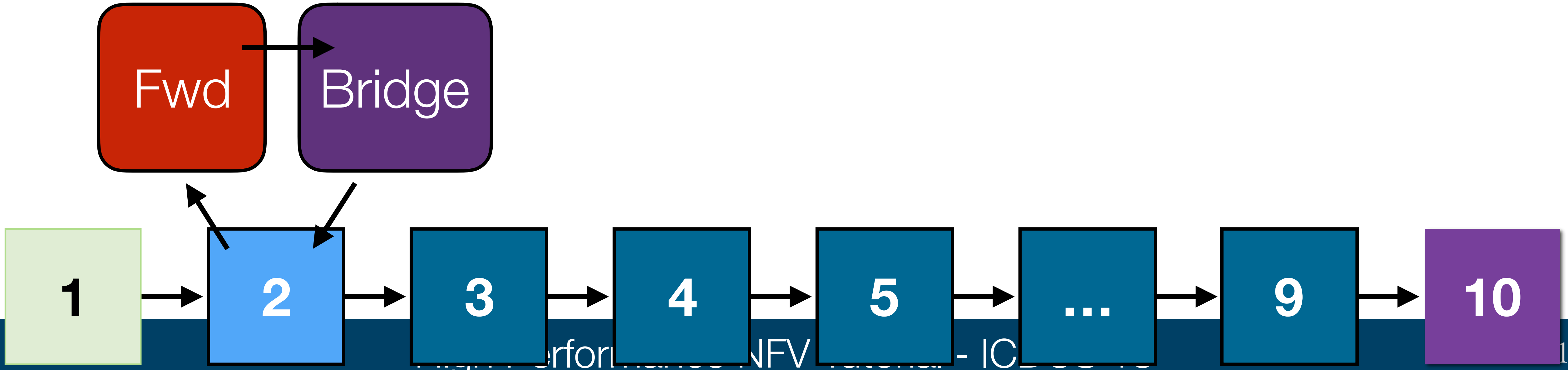


6

```
# Terminal 1: ONVM Manager (skip this if it is already running)
cd $ONVM_HOME/onvm
./go.sh  0,1,2  3 -s stdout
```

```
# Terminal 2: Simple Forward NF
cd $ONVM_HOME/examples/simple_forward
./go.sh  3 1 2
# parameters: CPU core=3, ID=1, Destination ID=2
```

```
# Terminal 3: Bridge NF
cd $ONVM_HOME/examples/bridge
./go.sh  4 2
# parameters: CPU core=4, ID=2
```



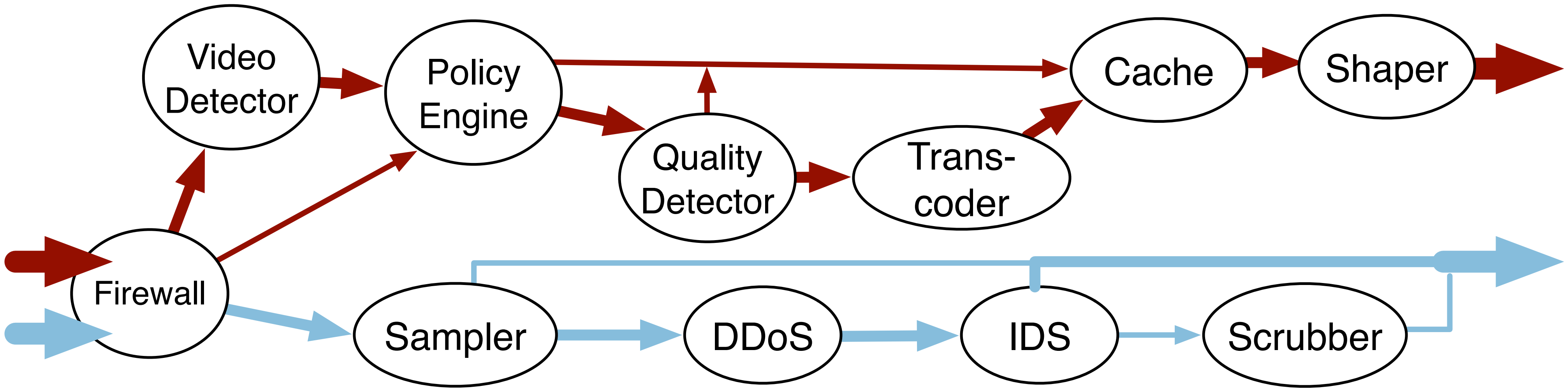
Packet Steering

NF Controlled

- NFs can specify an action (to NF, out port, drop, etc)

Manager Controlled

- Flow table in manager defines a Service Chain
- List of actions (to NF, out port, drop, etc)
- Flow table rules are installed by NFs
- SDN controller example NF uses OpenFlow to lookup service chain rules for each flow



Flow Director

Data structure and API to define service chains

- Maps a flow (5-tuple) to an array of actions

Code is in shared/onvm_flow_dir.c

- High speed concurrent hash table based on Cuckoo Hash
- Re-uses RSS packet hash calculations to lower overhead
- Manager provides flow table implementation, but relies of NFs to configure the rules

Flow	Action 1	Action 2	Action 3	...
f1	ToNF 3	ToNF 4	Out 1	
f2	ToNF 3	ToNF 4	ToNF 7	...
...				

Design Summary

NFs must be modular and easy to compose

- Isolated processes run in Docker containers
- Simple development, deployment, and chaining

Break abstractions when necessary

- Bypass OS and network stack to get raw packets
- Shared memory between processes for faster chains

Balance control across the hierarchy

- SDN controller provides flow table rules
- NFs can make custom decisions based on individual packets
- Manager can override decisions if necessary

THE GEORGE WASHINGTON UNIVERSITY

open source code at **http://sdnfv.github.io**

UCR