Network Topology Offload with Intelligent NICs

Rolf Neugebauer
(rolf.neugebauer@netronome.com)

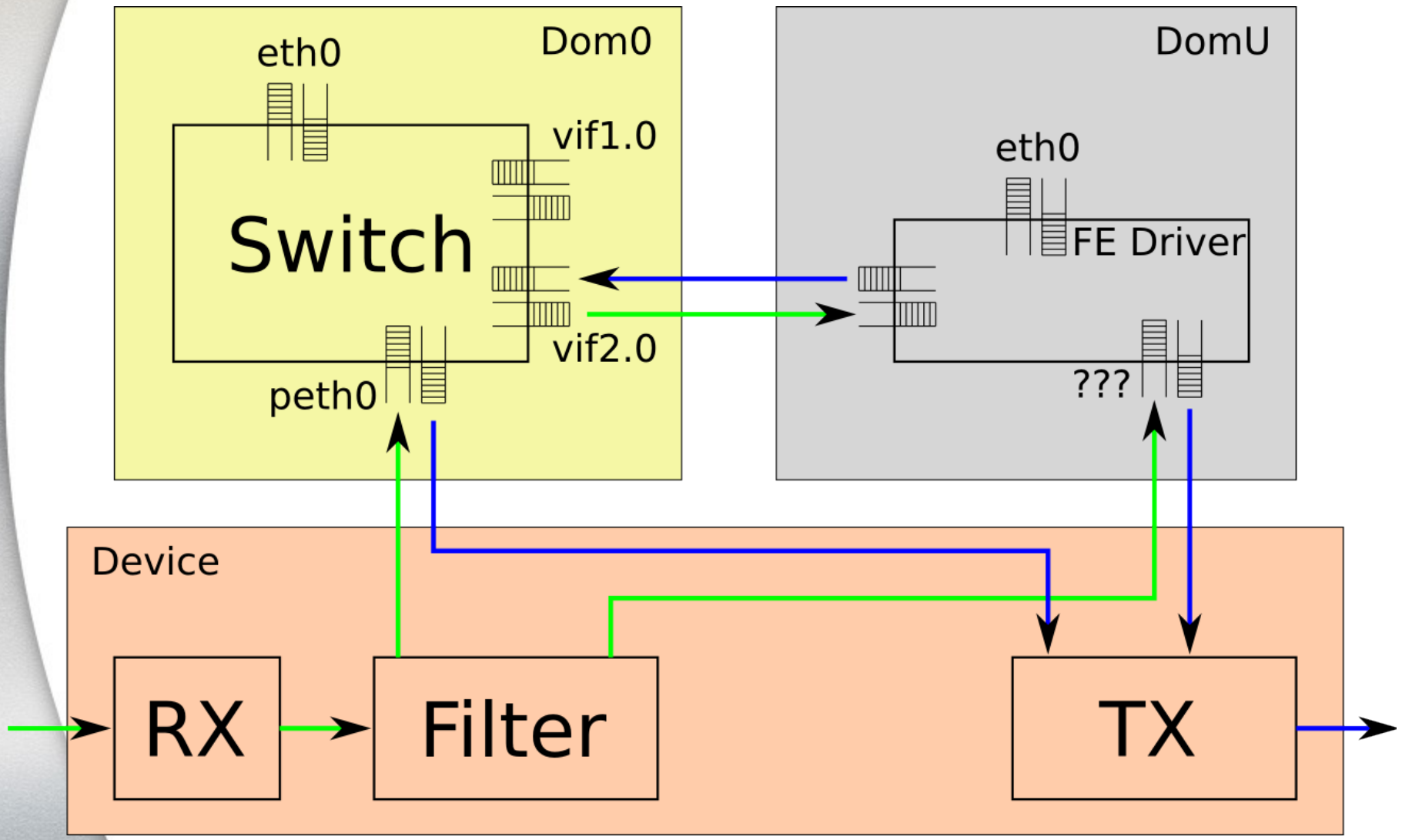
XenSummit June 2008

Motivation

- NICs with multiple queues and Virtualization support:
  - VMDq, Solarflare, Neterion, PCI-SIG SR-IOV etc
- Several physical ports (separate PCI functions)
- Larger number of virtual endpoints: 16, 32, ...
- How are these connected? Typically:
  - Ingress: perform simple classification to determine queue: MAC, VLAN, IP header fields, ...
  - Egress: basic scheduling

NETRONOME

Current Xen support



Current Xen support (cont.)

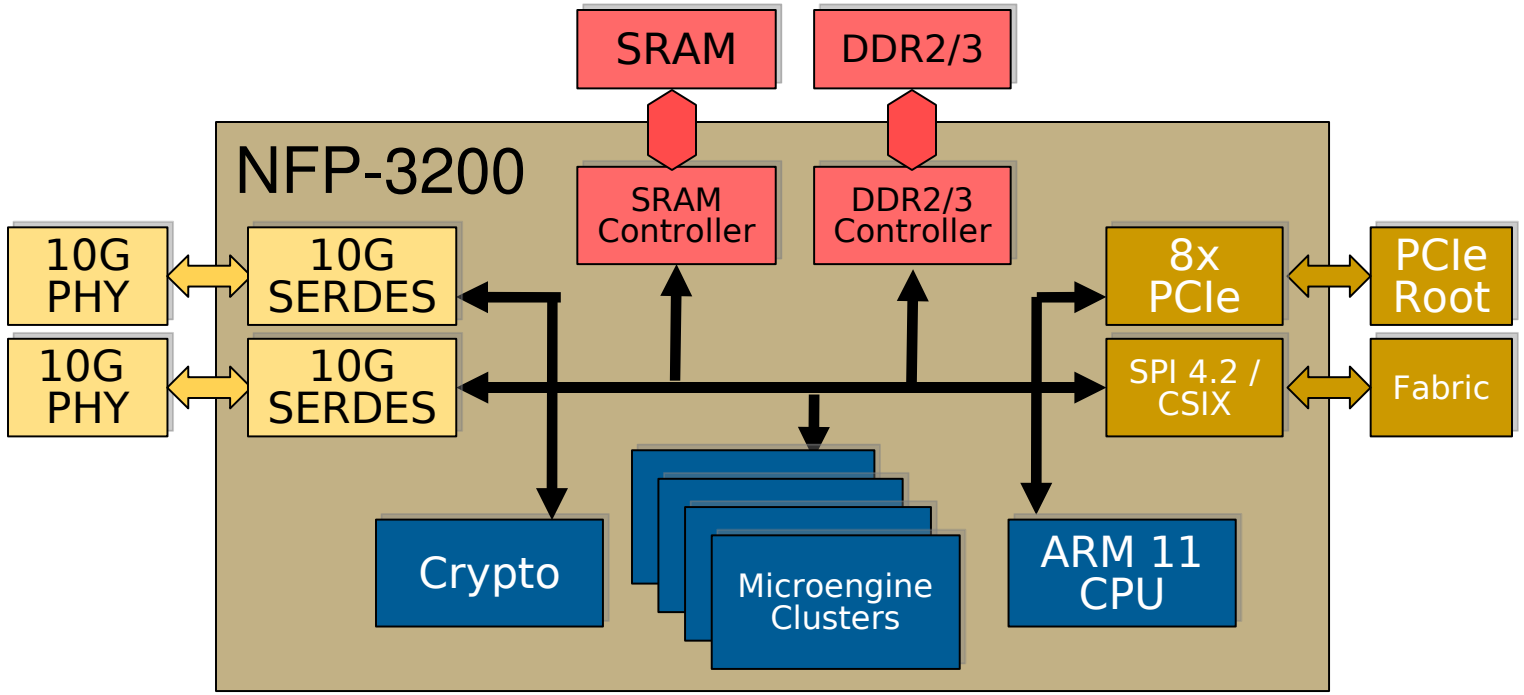
- Complex and difficult to comprehend
  - e.g. TX switch in FE
- Management of endpoints? SR-IOV?
  - (see other talks in this session)

- Support for different network setup?
  - Firewall, NAT, Load Balancing, Routing
- Support for VLANs?
- Multiple physical ports?
- Different types of endpoints?
  - TAP, (iSCSI), ...
- More complex “flow processing”
  - Cut-Through, early drop, ...

- Intelligent, programmable NPU can help

NETRONOME

Netronome NFP3200



- NFP-3200 evolved from Intel® IXP 28xx family
  - 40 RISC microengines for increased performance and functionality
  - Cost-effective DDR2/3 memory
  - PCIe/SPI for dataplane applications
  - Line interfaces with support for 10G Ethernet

NETRONOME

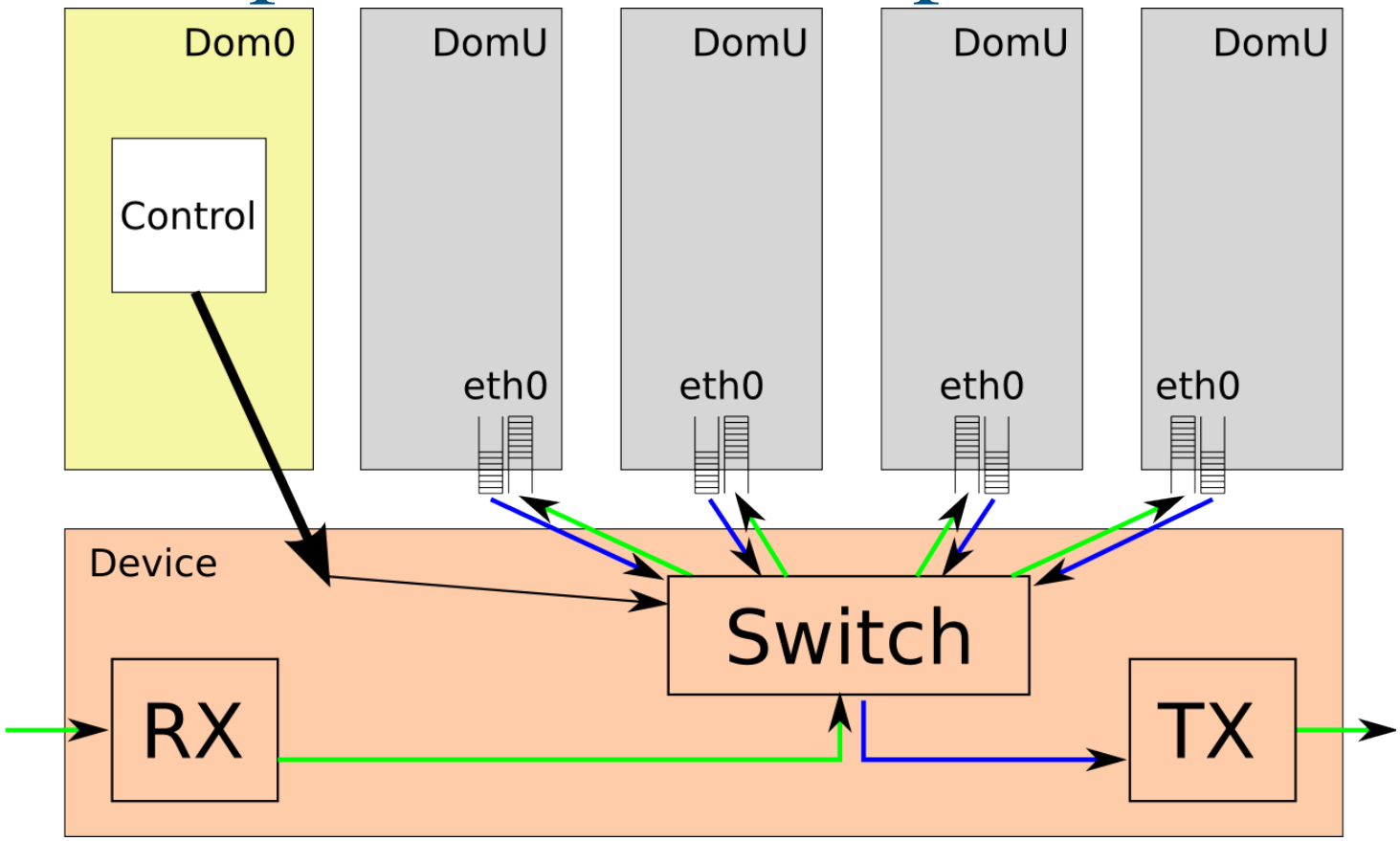
Topology offload vs acceleration

- Network Processors allows complete topology *offload*
  - Connect VMs using common network elements
  - Ethernet switch, firewall, load-balancer, ...
  - Network element's data-path entirely on NP
  - Controlled by user software in control VM

- Some functions can't be offloaded, but *accelerated*
  - “Very stateful” firewall/NAT: FTP, SIP etc
  - TCP-Splice/Flow Cut-Through for proxies
  - Approach: accelerate standard Linux functionality
    - e.g. iptables and connection tracking
    - Pass minimum number of packets to host
    - Offer API to process rest on the NP

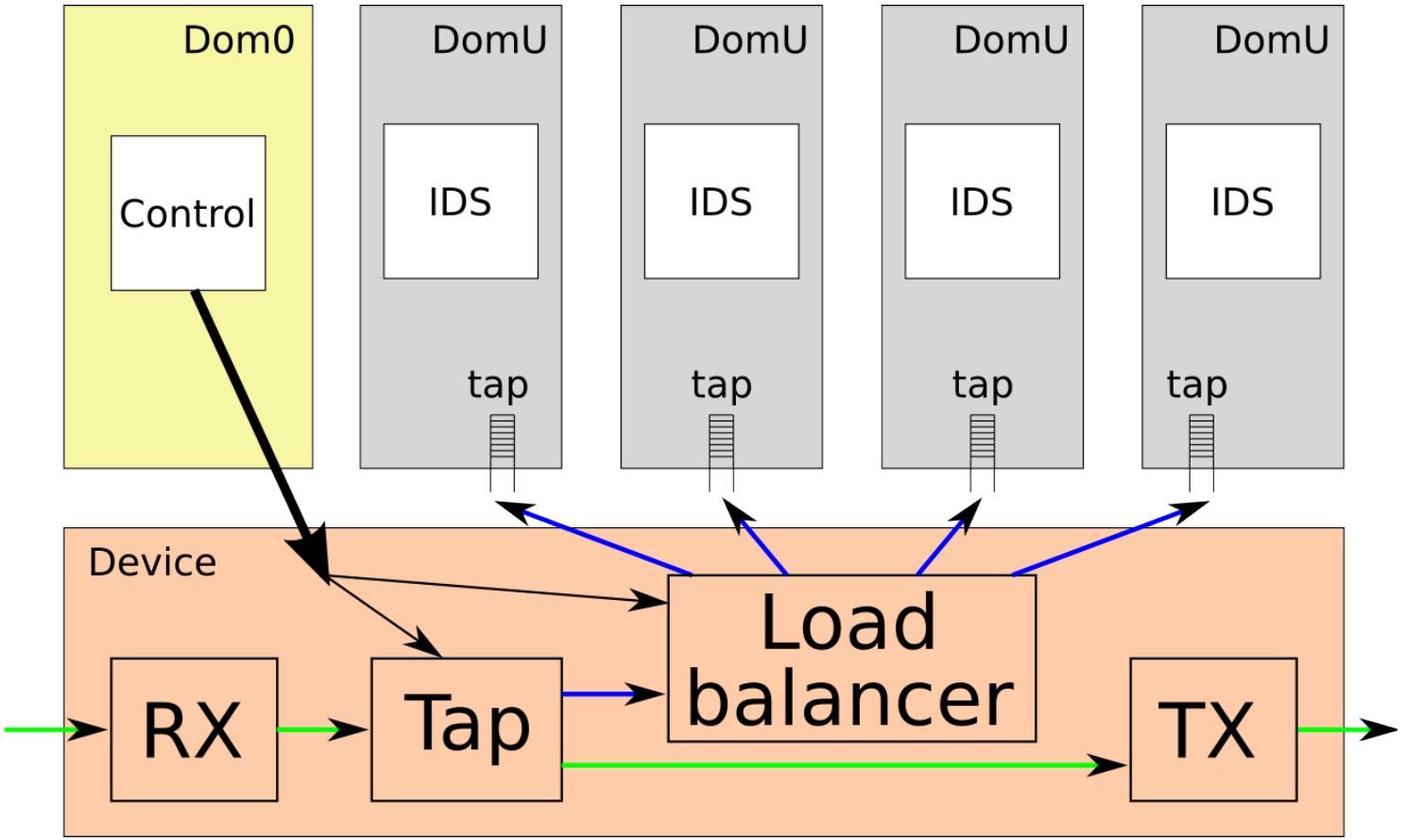
NETRONOME

A simple switch example



- InterVM traffic either via device or separate network
- NB: plenty of spare ME resources for this config

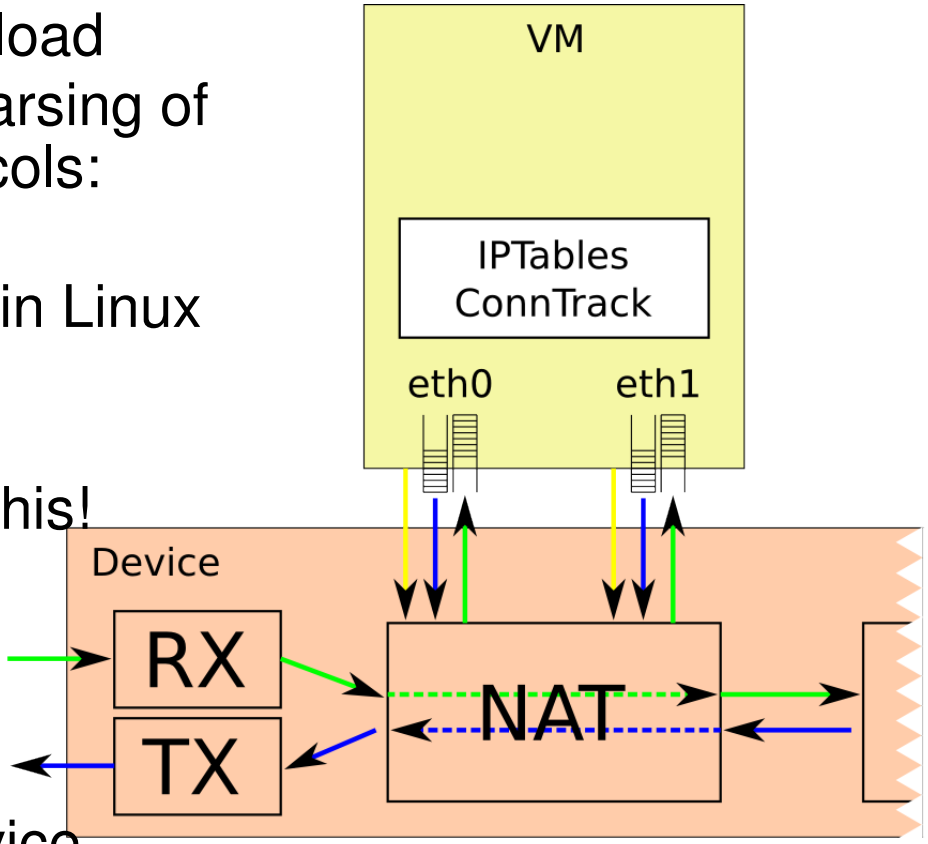
A simple IDS example



- TAP can perform filtering based on rules
- Can be extended to IPS with a TX component per VM

NAT: Topology acceleration

- Previous examples were offload
- Full stateful NAT requires parsing of flow content for some protocols:
  - FTP, SIP, etc
  - Existing implementations in Linux
  - Difficult to do on Mes
- Let Linux NAT take care of this!
  - Pass all flows up to a VM
- Once connection has been established and is deemed "safe"
  - Instruct NAT block on device to "cut-through" and modify flow
- Not new :) -> StoreLink 351x

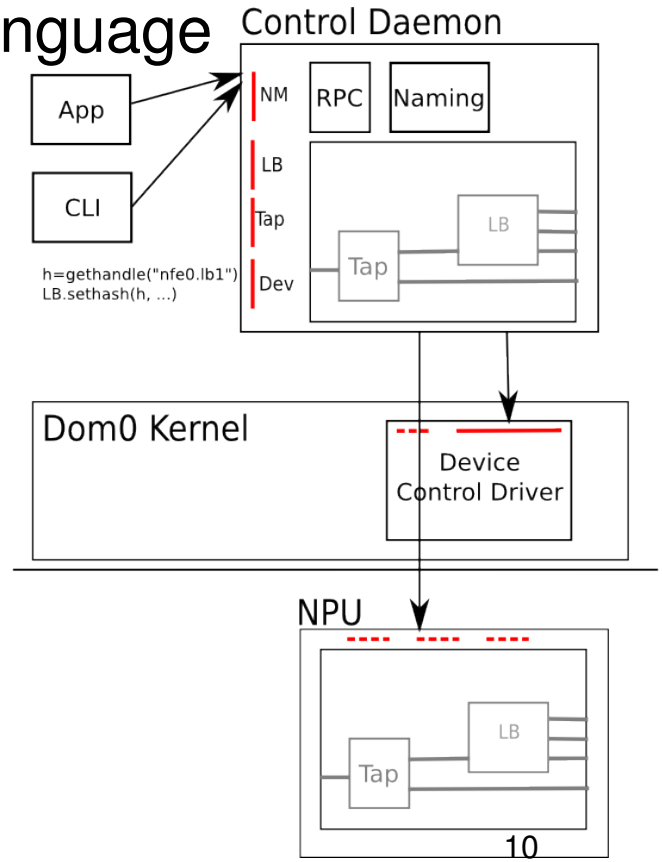


NETRONOME

Topology configuration

- Topology is modeled a Nodes connected by Links
- Potentially can implement arbitrary topologies
- Topology elements need to be configured

- Designing a topology description language
- Developing a control daemon
- Daemon has model of topology
  - Topology description
- Configures elements on the device
- API and cmdline



Conclusions

- Programmable network processors offer new functionality for network IO virtualisation
  - Complete topology offload
  - Accelerate networking features
- Existing netchannel architecture does not support these type of network devices
  - But very little support in Xen required
- NFP3200 has significant resources to implement complex topologies.
  - Will provide some example topologies
  - SDK will be available to roll your own

NETRONOME

Questions?

Also thanks to

Espen Skoglund
Joshua LeVasseur

NETRONOME

Sample topology description

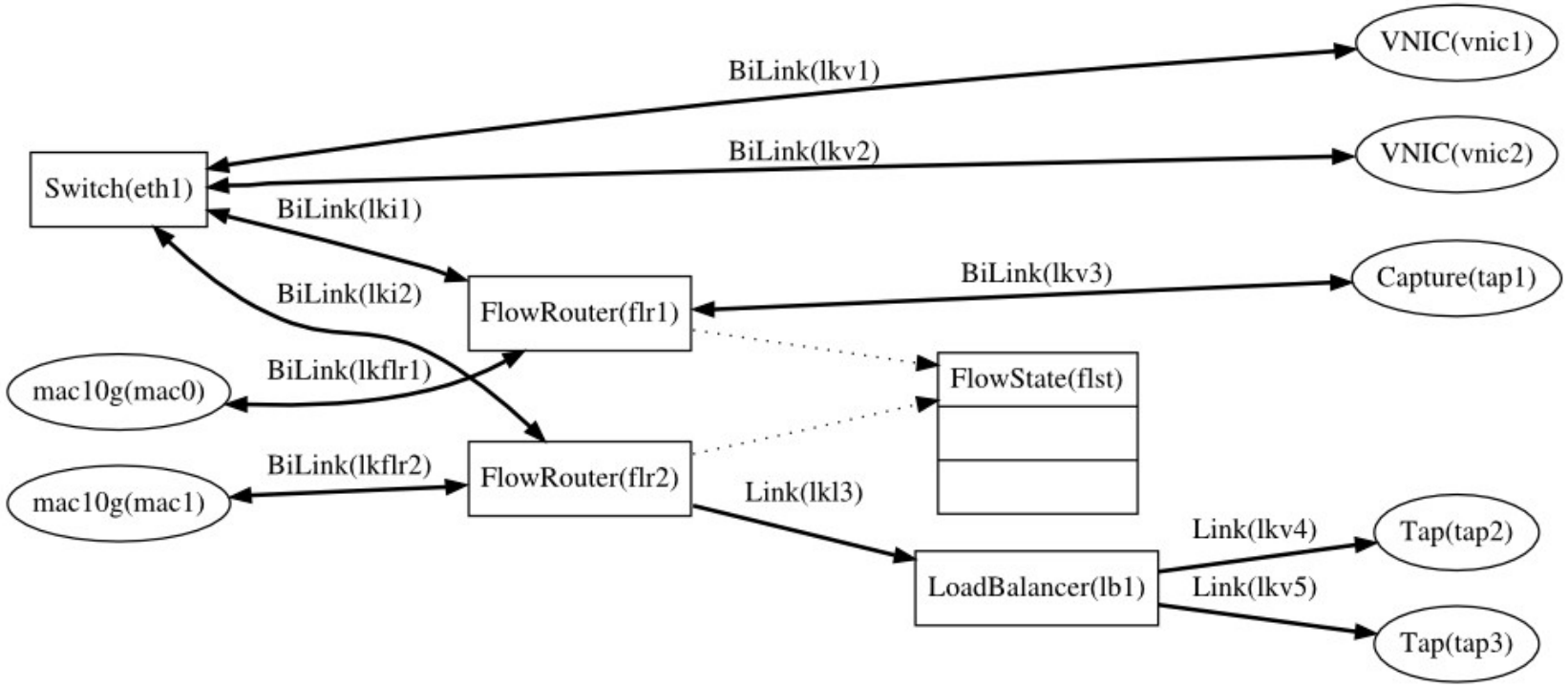
```
d = NFE3200() # This is for a NFE3200
# First the functional blocks
eth1 = Switch("eth1", 4)
flst = FlowState("flst")
flr1 = FlowRouter("flr1", 2, flst)
flr2 = FlowRouter("flr2", 2, flst)
lb = LoadBalancer("lb1", 2)

# the next two line are ony needed to
make the backends simpler
d.add_table(flst)
d.add_nodes((eth1, flr1, flr2, lb))

# Instantiate some NICS and VNICS
d.pep[0] = MAC10g("mac0")
d.pep[1] = MAC10g("mac1")

d.vep[0] = VNIC("vnic1")
d.vep[1] = VNIC("vnic2")

d.vep[2] = CaptureVNIC("tap1")
d.vep[3] = TapVNIC("tap2")
d.vep[4] = TapVNIC("tap3")
```

```
# Wire everything up
lkpp1 = BiLink("lkflr1", flr1, 0, d.pep[0], 0) # Phys to FC
lkpp2 = BiLink("lkflr2", flr2, 0, d.pep[1], 0)

lk1 = BiLink("lki1", eth1, 0, flr1, 1) # FC port 1 to switch
lk2 = BiLink("lki2", eth1, 1, flr2, 1)

lkv1 = BiLink("lkv1", eth1, 2, d.vep[0], 0) # Switch ports 2 & 3 to
VNICS
lkv2 = BiLink("lkv2", eth1, 3, d.vep[1], 0)

lkv3 = BiLink("lkv3", flr1, 2, d.vep[2], 0) # Tap from flr1 to a
Capture VNIC

lkl3 = Link("lkl3", flr2, 2, lb, 0)  # Tap from flr2 to load balancer

lkv4 = Link("lkv4", lb, 1, d.vep[3], 0)   # Load balance across
two Tap VNICs
lkv5 = Link("lkv5", lb, 2, d.vep[4], 0)

# the next two line are ony needed to make the backends
simpler
d.add_links((lkpp1, lkpp2, lk1, lk2, lkv1, lkv2, lkv3, lkl3, lkv4,
lkv5))
```

NETRONOME

Topology (auto generated)

- Generated from topology description (using graphviz)



NETRONOME