Lark: Software Defined Networking Integration with HTCondor and More

Zhe Zhang
University of Nebraska-Lincoln

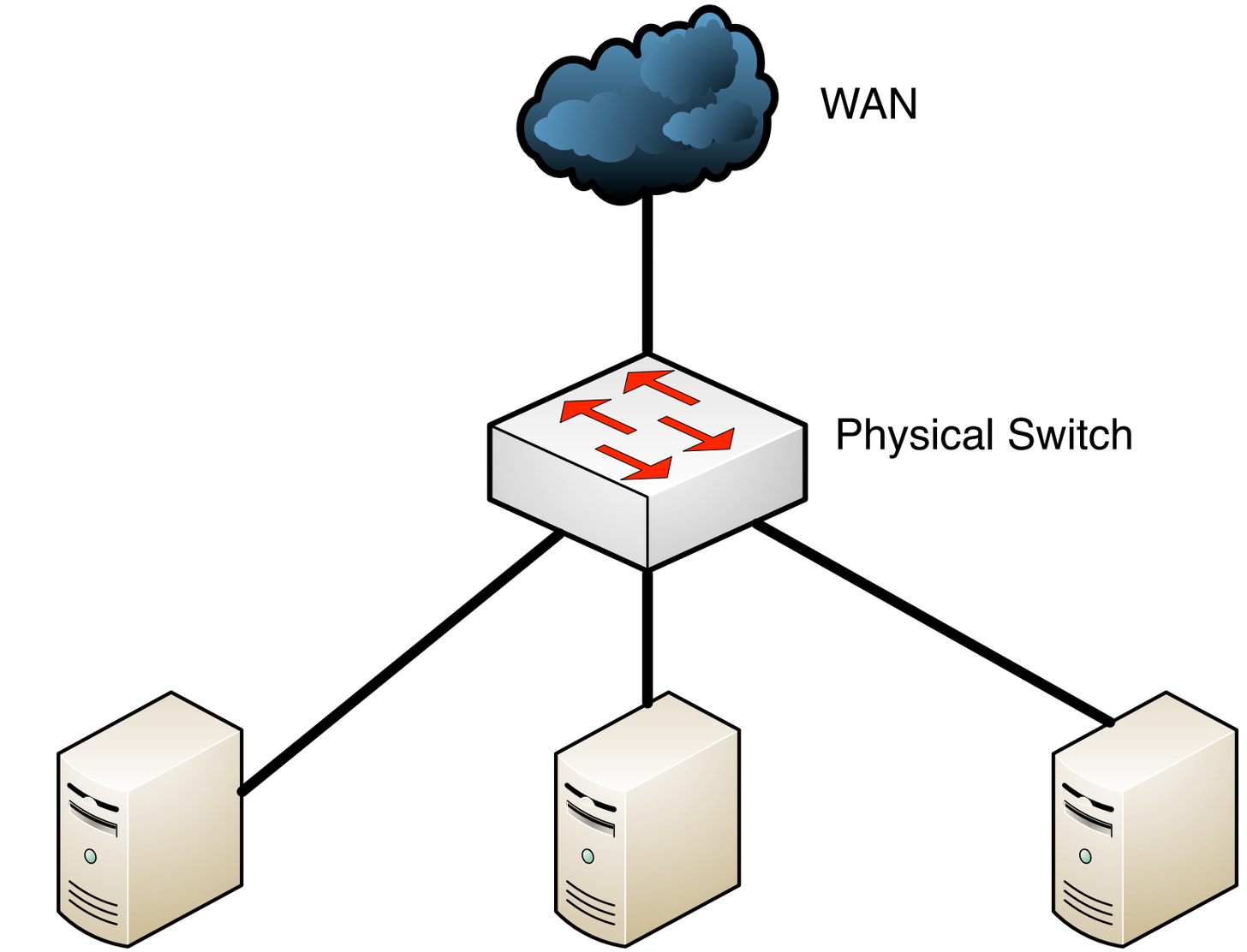


Figure 1: Traditional network topology for a cluster

What's the problem here for HTCondor?

Problem & Motivation

- At worker node level, HTCondor cannot manage networks similarly like CPU, memory and disk.

- At network layer, HTCondor cannot see differences among jobs come from the same host. Thus the granularity is per-host but not per-job.

- What we want is to improve this granularity to be per-job.

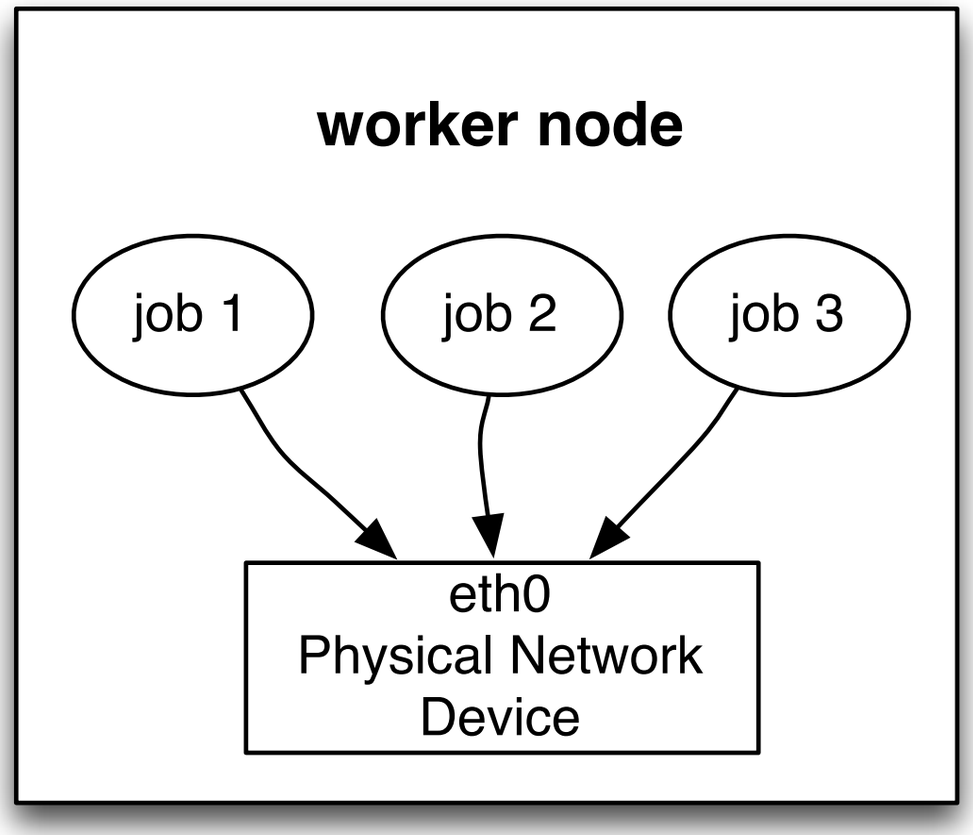


Figure 2: Zoom in view for a worker node

Previous Work

- The Lark project aims to tackle this problem.
- Related Presentations and talks from previous years' HTCondor weeks: Todd & Garhan@HTCondor Week13; Brian@HTCondor Week12.
- This talk will report our latest progress and development.

Partitionable Slot

- Allows HTCondor to use the resources of a slot in a dynamic way; these slots may be partitioned according to the jobs running as opposed at configuration time.
- Slots have a fixed set of resources which include the cores, memory and disk space.
    - They spawn a *dynamic slot* based on the resources the job needs.
- By partitioning the slot at runtime, we can become more flexible about resource usage.

Partitionable slot examples

- Partitionable slot resource

```
cpu = 10
memory = 10240
disk = 102400
```

- Job A allocates to this slot with resource requirement

```
cpu = 3
memory = 1024
disk = 10240
```

- After Job A occupies a dynamic slot, remaining resource:

```
cpu = 7
memory = 9216
disk = 92160
```

Can we do this for network bandwidth resource?

- Yes, we can! By using extensible machine resource mechanism.

- In HTCondor configuration file, add:

```
MACHINE_RESOURCE_INVENTORY_BANDWIDTH
= /path/to/executable
```

- Executable to check available bandwidth resource and attributes are inserted to machine ClassAd.

- Now machine slot broadcasts resource:

```
cpu = 10
memory = 10240
disk = 102400
bandwidth = 1000
```

- How does user request for bandwidth resource?

```
Universe = vanilla
Executable = /usr/bin/curl
Output = test.out
Error = test.err
Log = test.log
request_bandwidth = 10
Arguments = -4 -O http://hcc-lark03.unl.edu/1GB.zip
Queue
```

Not enough…

- We still need to make sure each job can only use the amount of bandwidth it requests.

- We utilize network namespaces + Open vSwitch: network namespaces give us network isolation and Open vSwitch provides host-level QoS and bandwidth limiting.

Open vSwitch

- A production quality, multilayer virtual switch (a piece of software runs in the kernel which performs like a hardware switch).
- Enable network automation through programmatic extensions.
- We utilize QoS functionality and OpenFlow support.

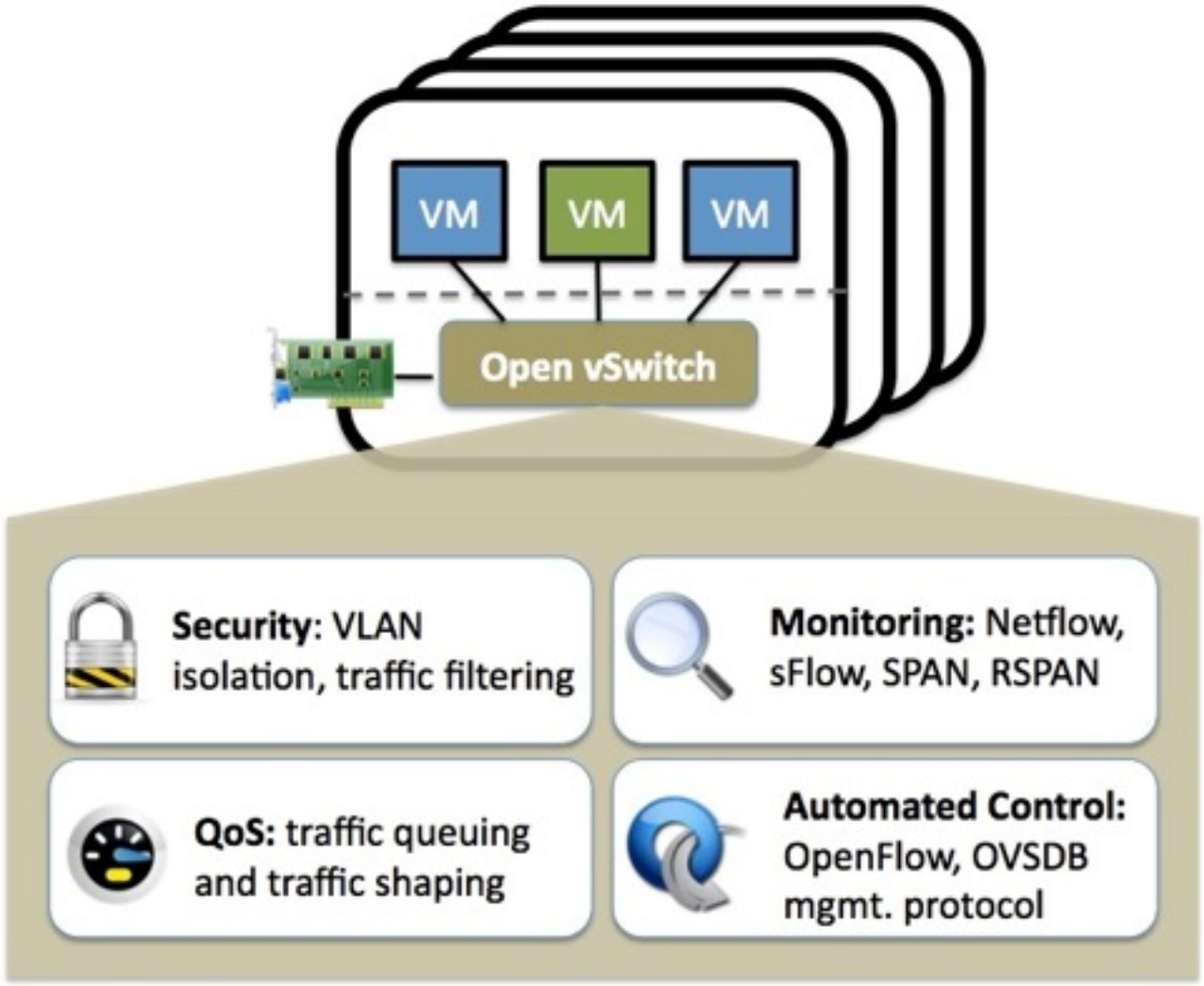
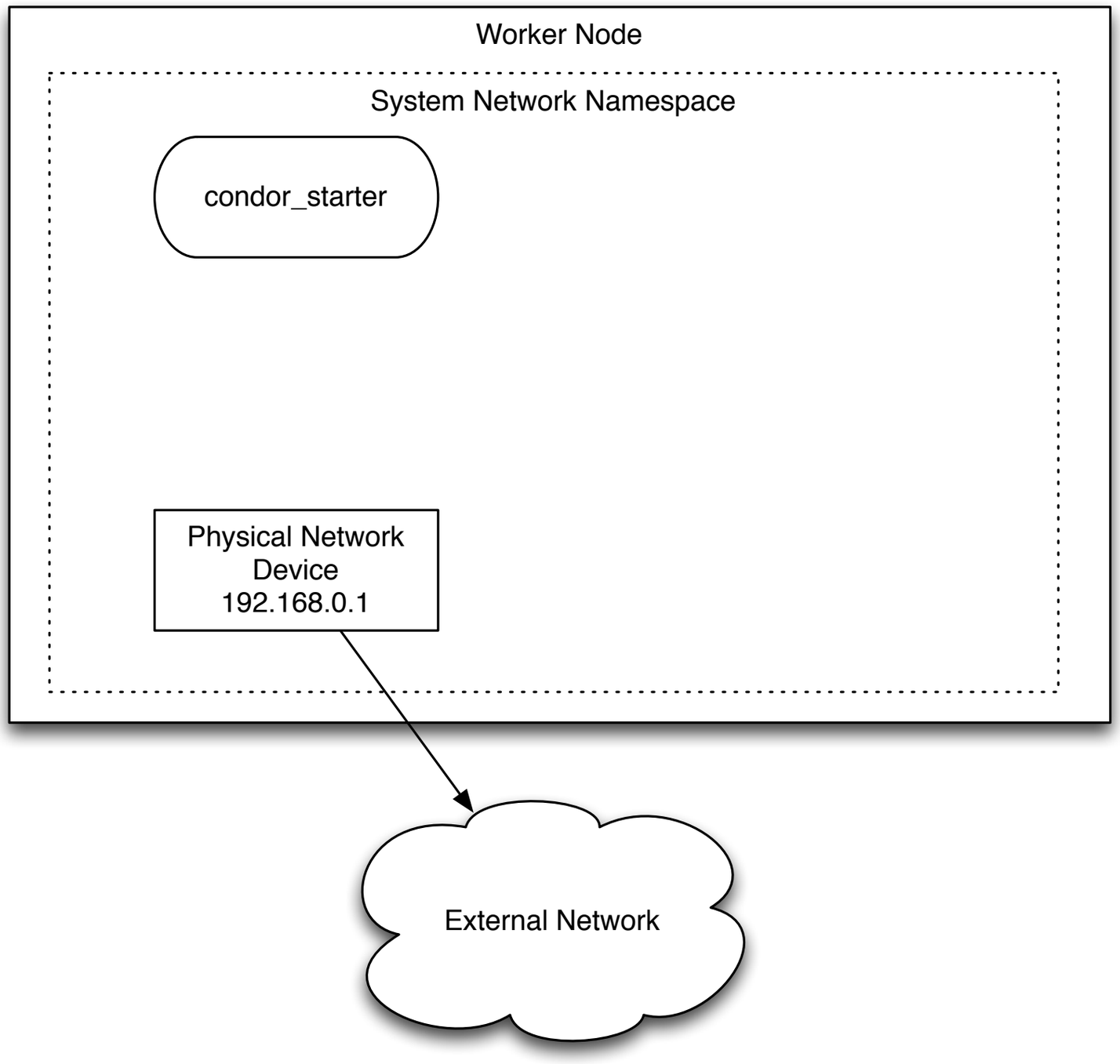


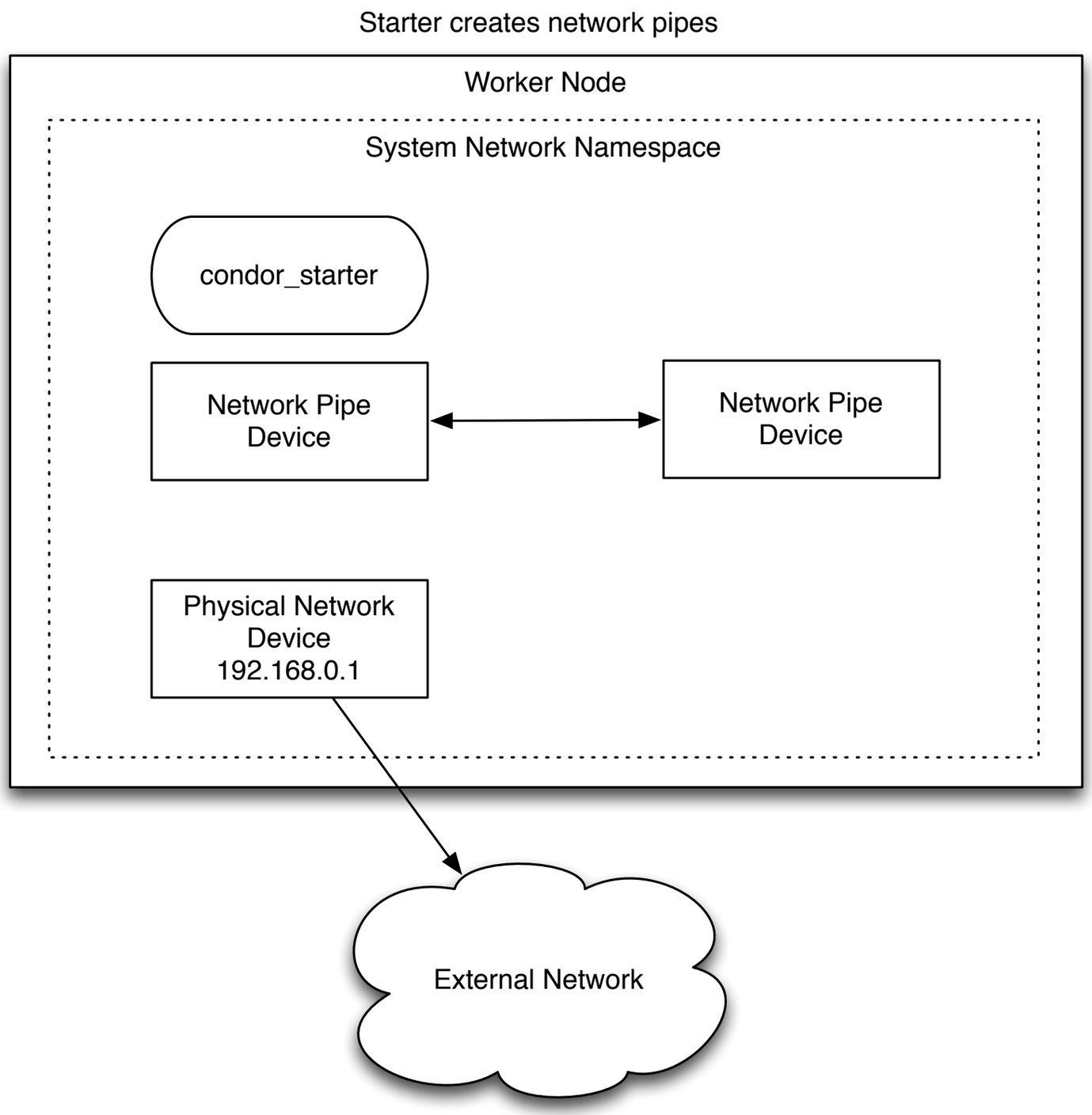
Figure 3: Open vSwitch

Network namespace with Open vSwitch Illustration - Step 1

Initial Configuration

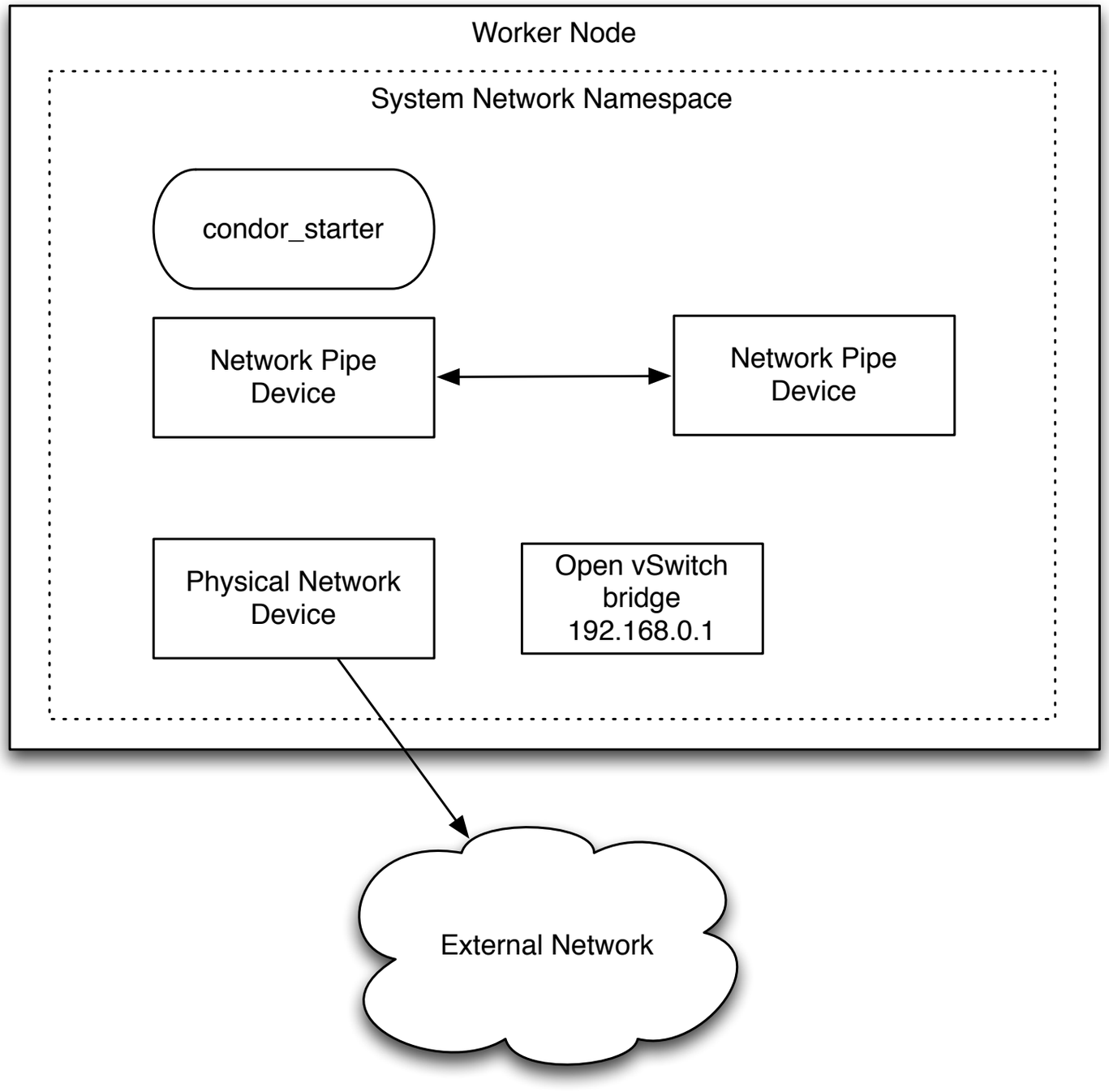


Network namespace with Open vSwitch Illustration - Step 2



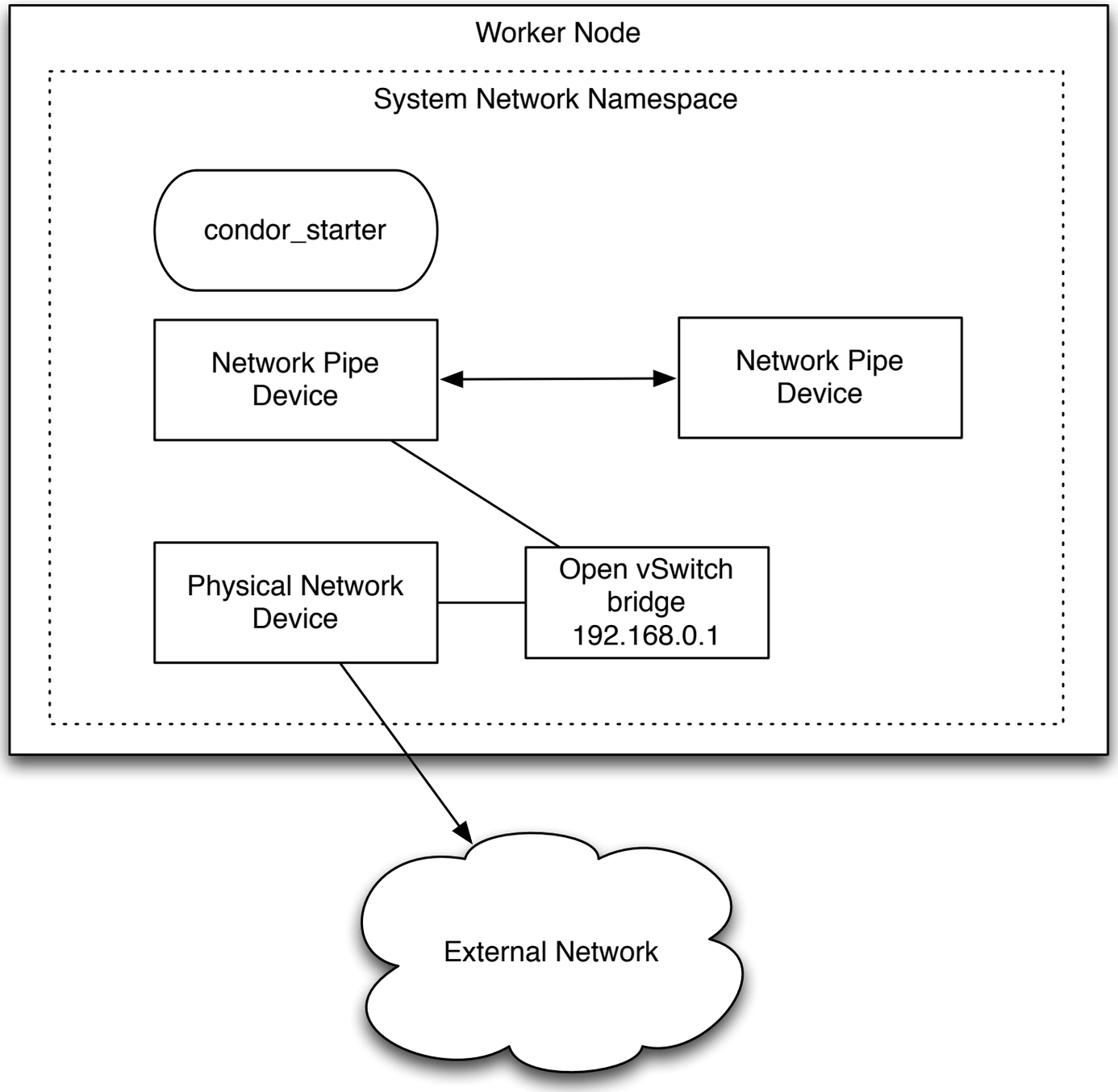
Network namespace with Open vSwitch Illustration - Step 3

Create Open vSwitch bridge
Assign IP address from physical device to bridge



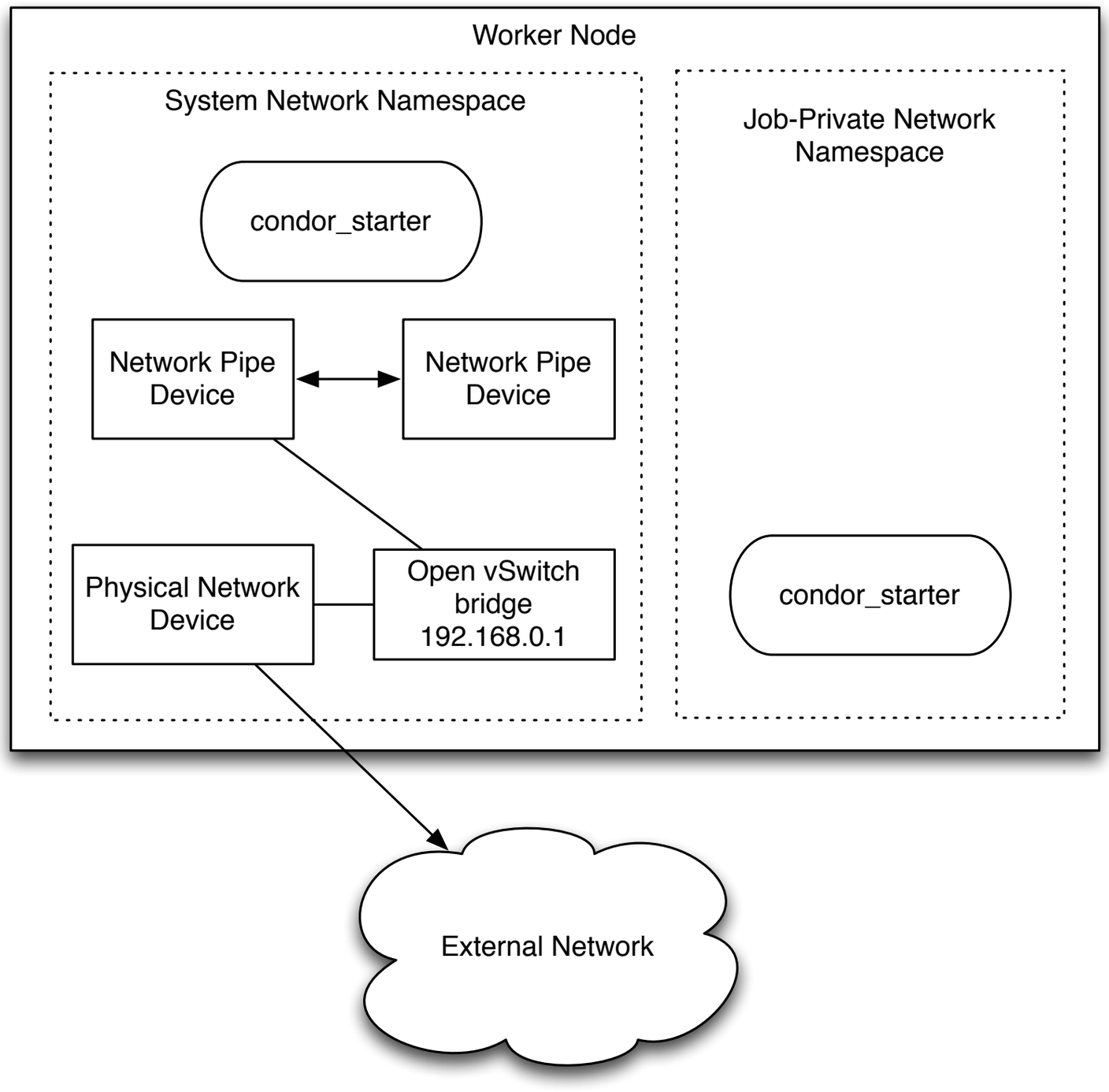
Network namespace with Open vSwitch Illustration - Step 4

Add physical interface and one end of virtual
Ethernet device pair to bridge



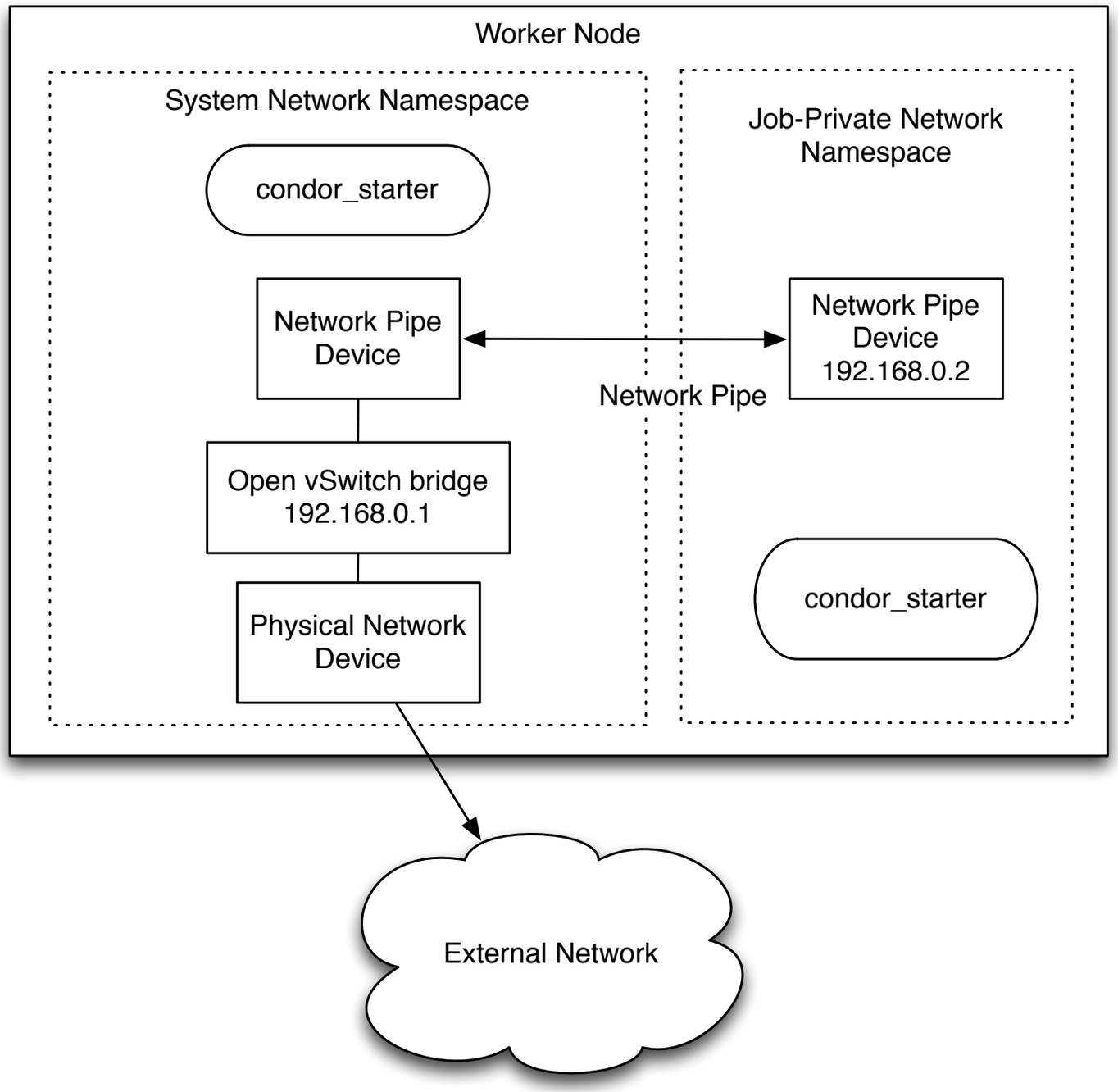
Network namespace with Open vSwitch Illustration - Step 5

Starter forks new process with new network namespace



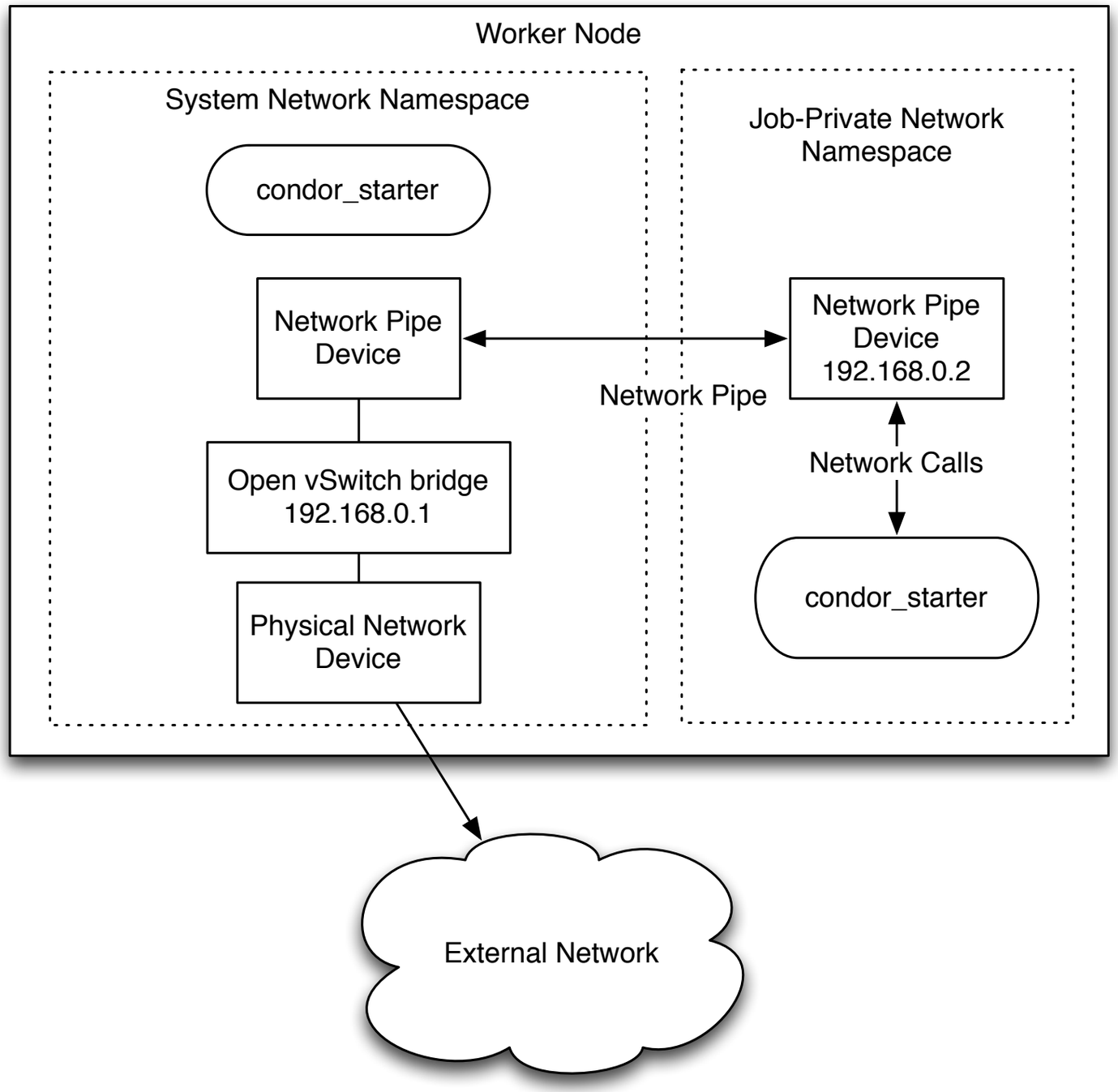
Network namespace with Open vSwitch Illustration - Step 6

Parent starter passes one end of network pipe to network namespace,
Child starter configures IP address via DHCP or static allocation



Network namespace with Open vSwitch Illustration - Step 7

Final Configuration



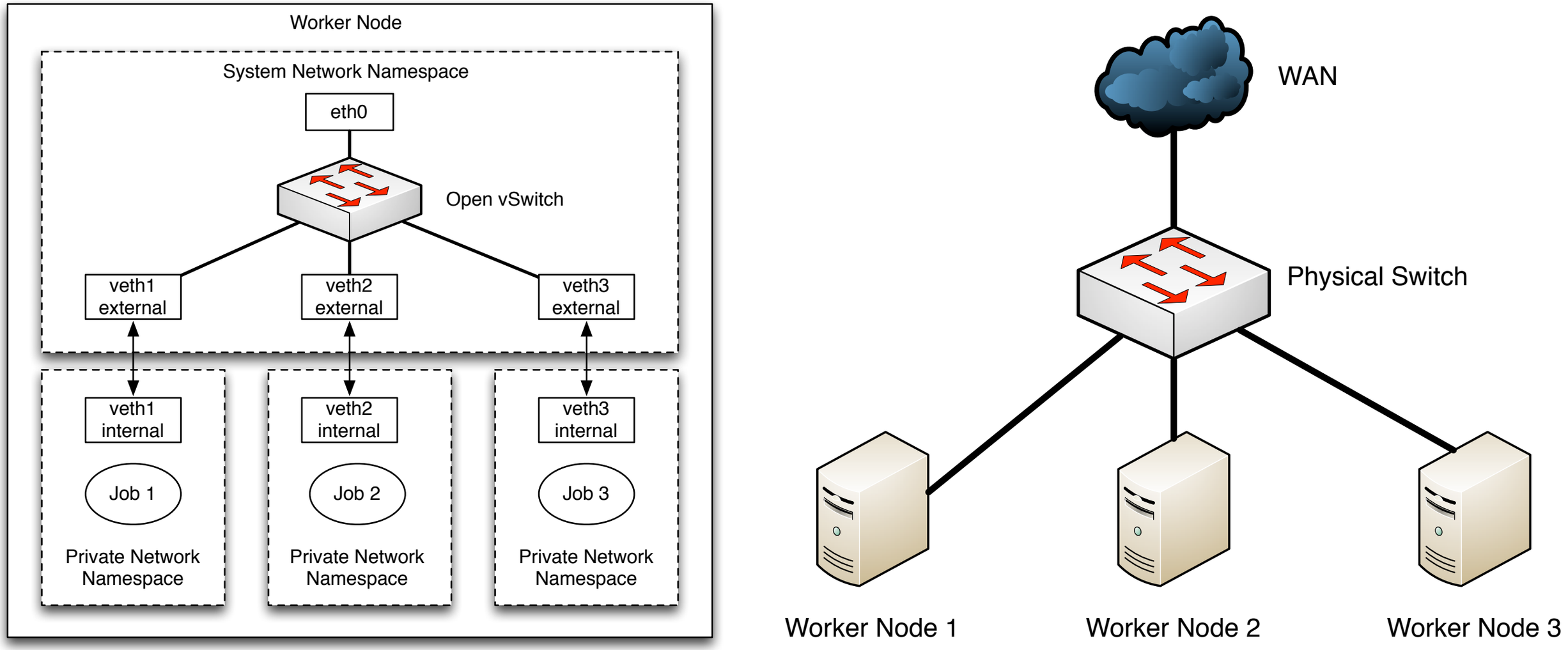


Figure 4: Worker node network configurations with Open vSwitch integration v.s. LAN configuration

Host level bandwidth management

- Now each job owns an unique network device, we can apply Open vSwitch QoS on the port that connects to the external end of the network pipe.

- Combine these two aspects: extensible machine resource + Open vSwitch integration, we can have a basic host level bandwidth management feature for HTCondor jobs.

- Let’s consider the previous job again.

Test HTCondor job

- Download a file of 1GB from hcc-lark03.unl.edu, job requests bandwidth 10Mbps.

  `request_bandwidth = 10`

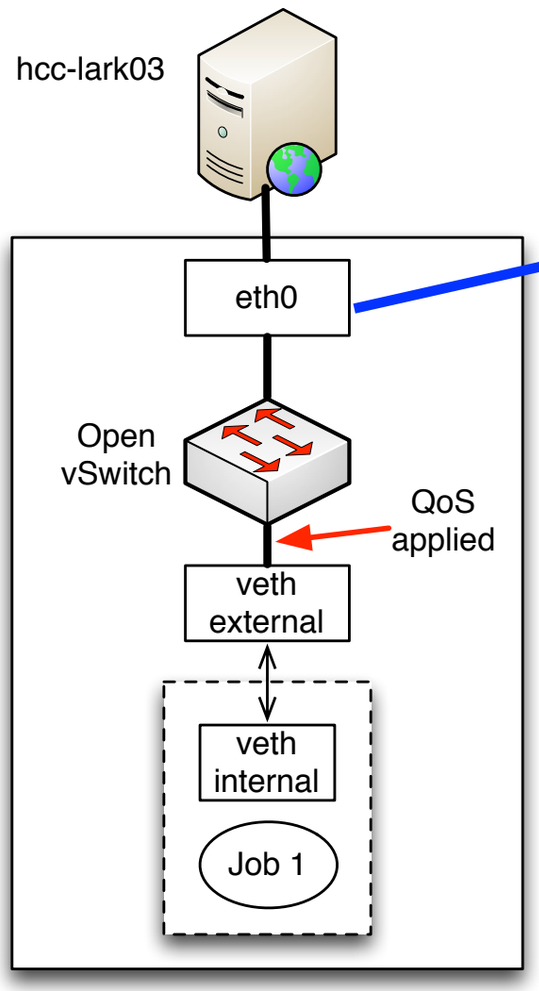


Figure 4: Test job network topology

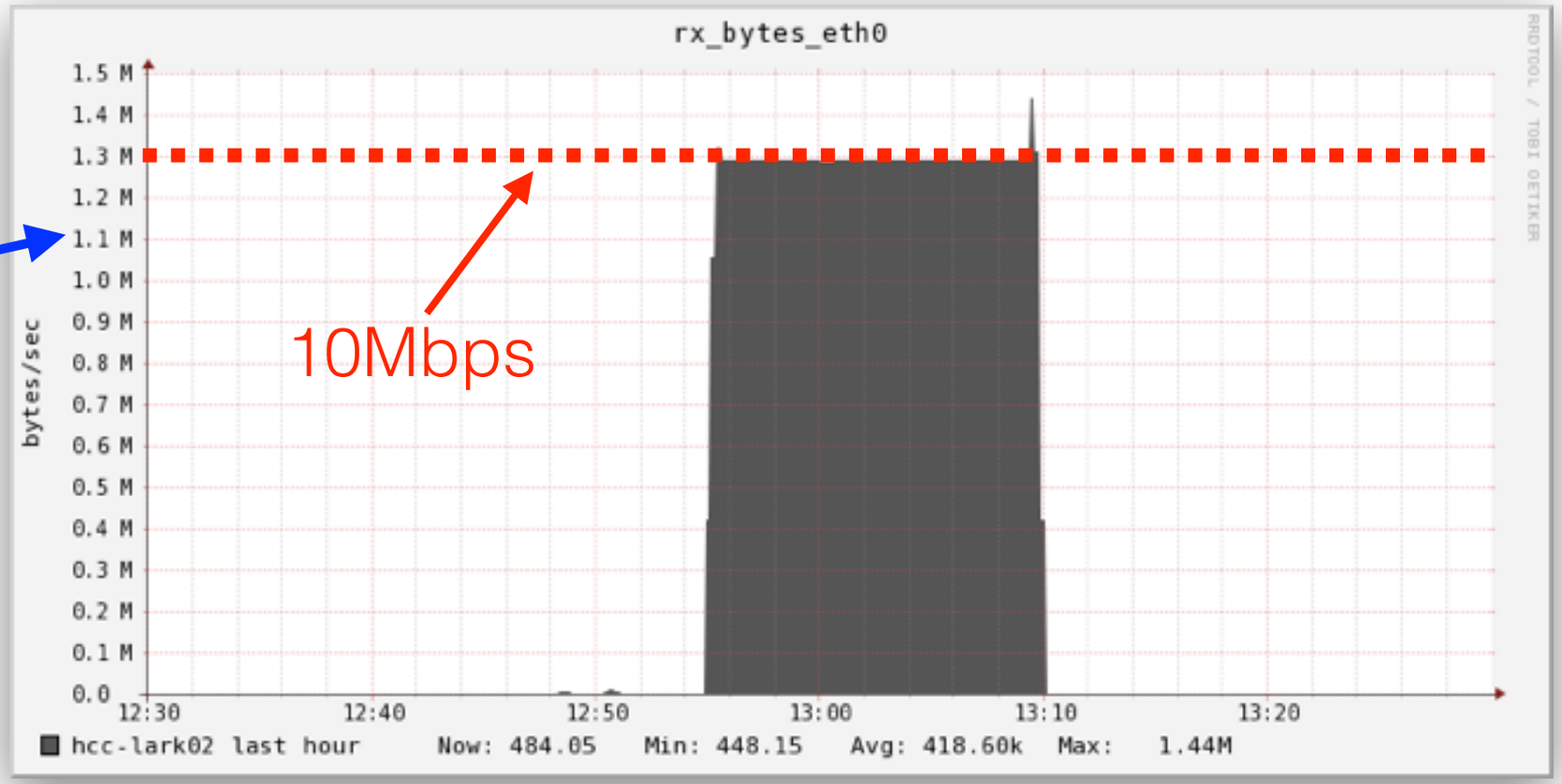


Figure 5: Traffic rate for test HTCondor job

See backup slides for more!

Now What?

- It is nice to have host-level bandwidth management functionality, but it is only useful at host level.

- Most sites *really* want to do WAN bandwidth management for HTCondor, which is more expensive and hence a more common bottleneck.

- To enable this capability, we need the network hardware to be able to associate application information (HTCondor) with network traffic.

- We seek the help from Software Defined Networking (SDN).

What is SDN?

"Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow™ protocol is a foundational element for building SDN solutions."

— Definition from Open Network Foundation (ONF)

OPEN NETWORKING FOUNDATION

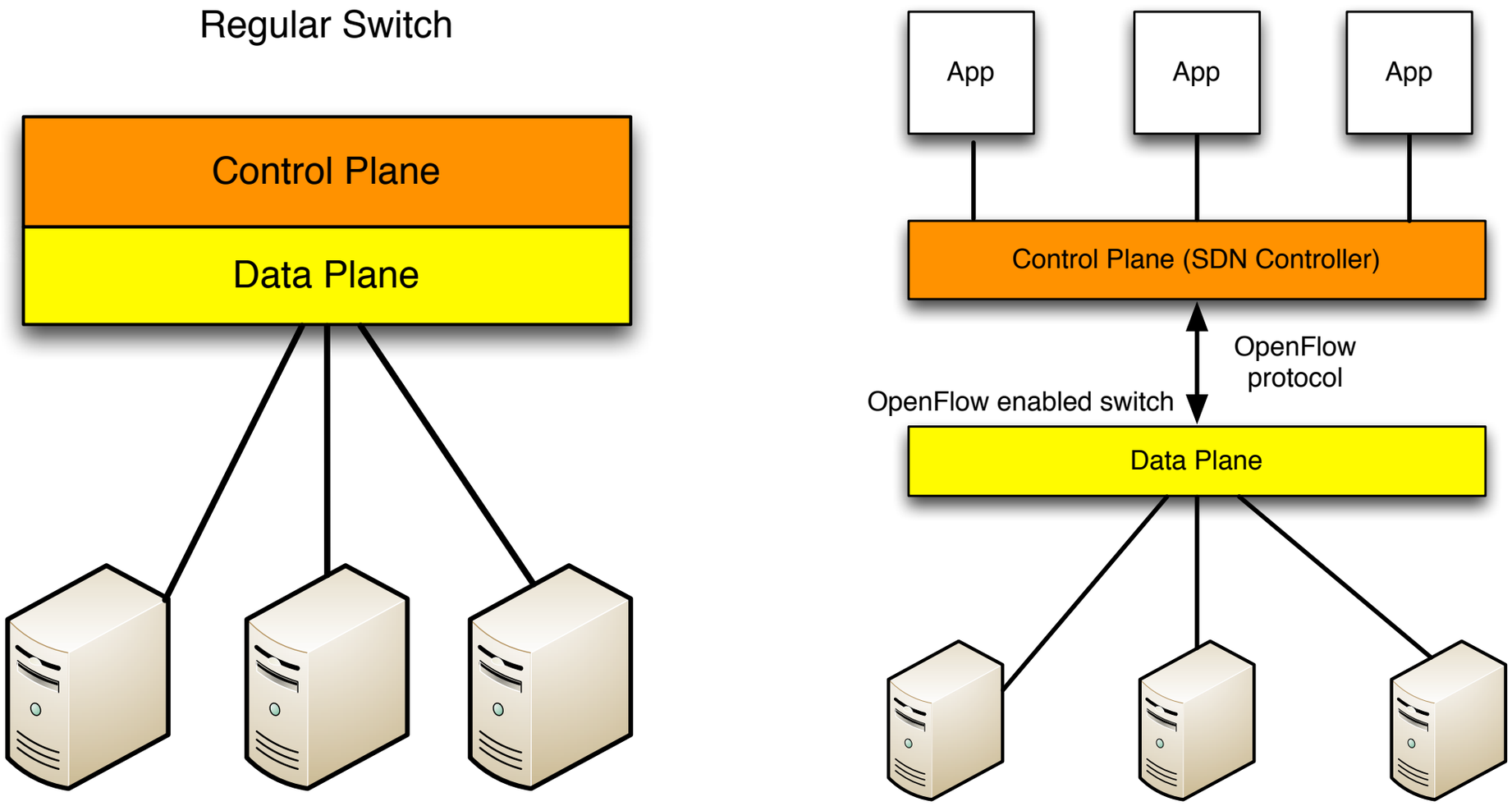


Figure 8: Regular switch v.s. OpenFlow enabled switch

- Control Plane becomes a software runs on commodity hardware as a controller.
- There are *flow table* on switch, which stores the OpenFlow rules used for packet forwarding.
- When packet hits switch, first iterate flow table for match; otherwise it is sent to controller.
- Controller determines what to do with the packet according to the controller program and then installs rule for this packet match to the flow table on switch.

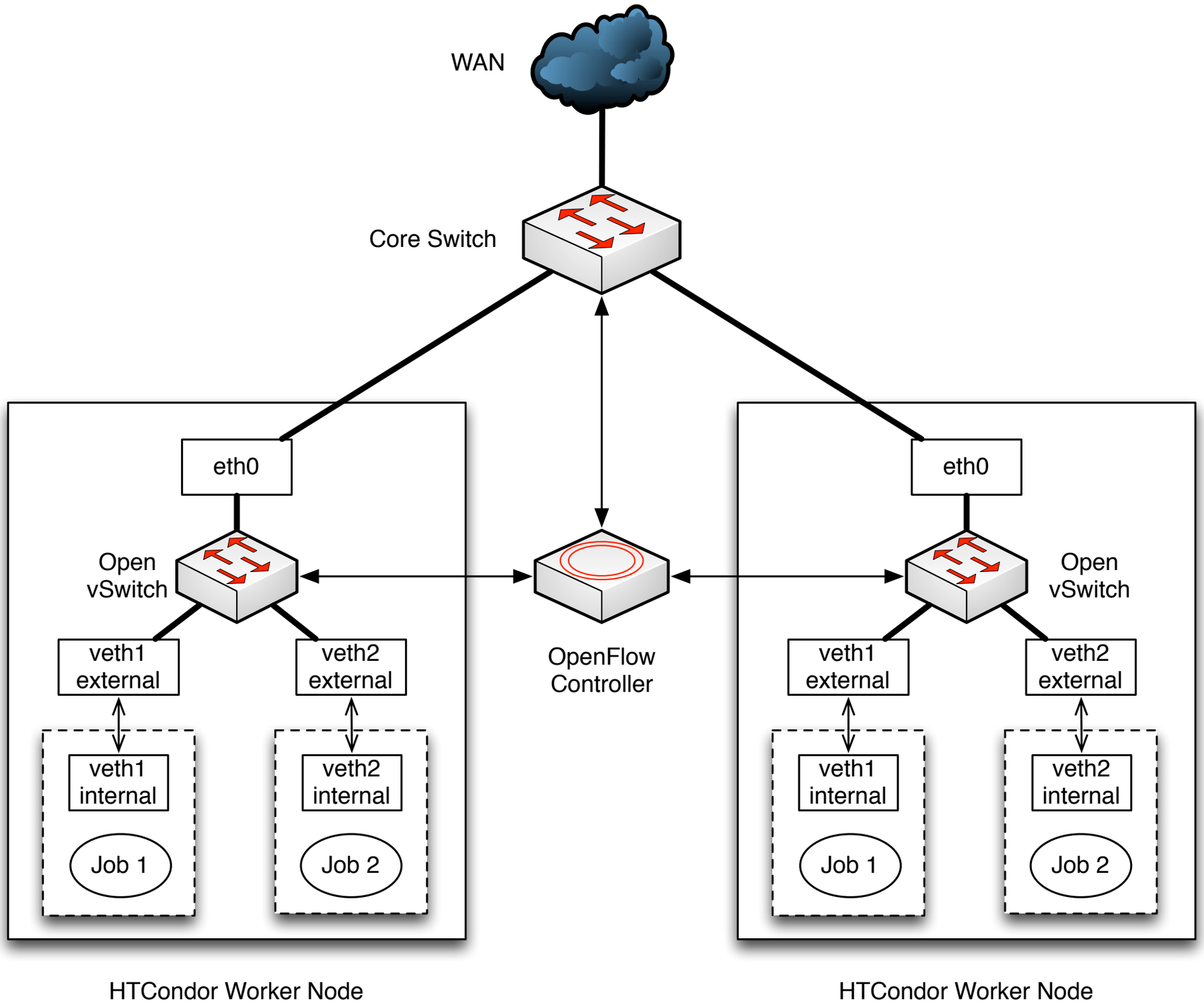


Figure 9: SDN-enabled cluster network topology

HTCondor + SDN

- A `condor_starter` plugin sends job and machine ClassAd (which includes the job IP address) to an OpenFlow controller before the job executes.
  - This uniquely **identifies** the traffic coming from a job.
- The controller **classifies** observed network traffic using this information; for example, it may classify the traffic by the job's `Owner` attribute.
- The controller **applies policy** to the packet by installing an OpenFlow rule in the correspond switch.

Hardware Switch

- On the network, we usually don't care about individual users - but what groups or projects they work on.

- Hence, we create a QoS queue with different bandwidth allocations (prioritization) for different projects on the WAN port.

- The controller classifies the network traffic by project associated with that job and writes a new hardware rule.

- The switch enqueues packets into the correct QoS queue.

Core Switch Level - Example

- Two HTCondor jobs, each of which uploads a file of 200MB to FTP servers.
- Belong to different projects, e.g. CMS vs NonCMS, with different WAN bandwidth allocated. (8Mbps vs 4Mbps)
- OpenFlow controller associates project with HTCondor traffic and do project-based QoS at core switch.

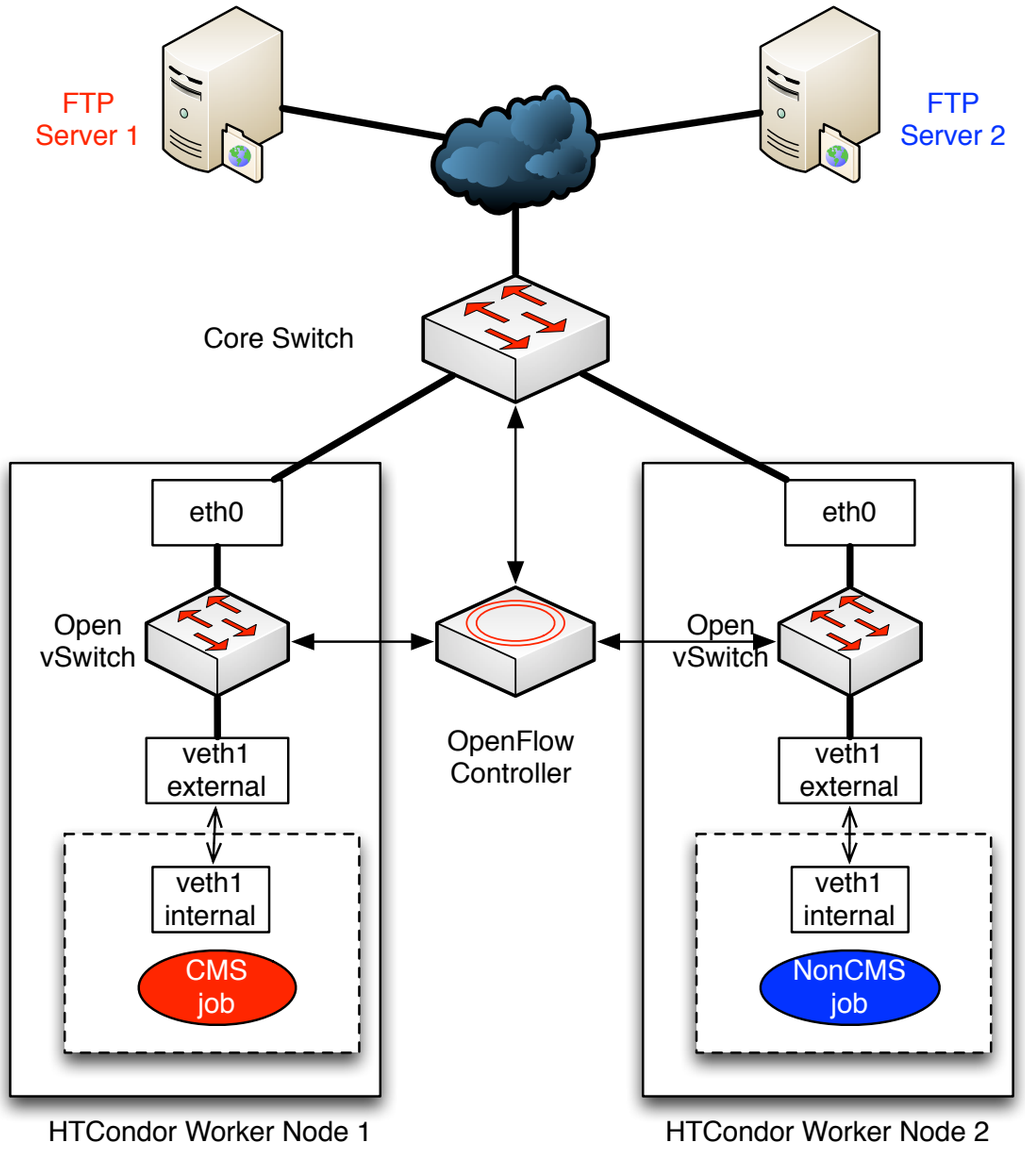


Figure 10: Network setup for HTCondor jobs

Experiment result

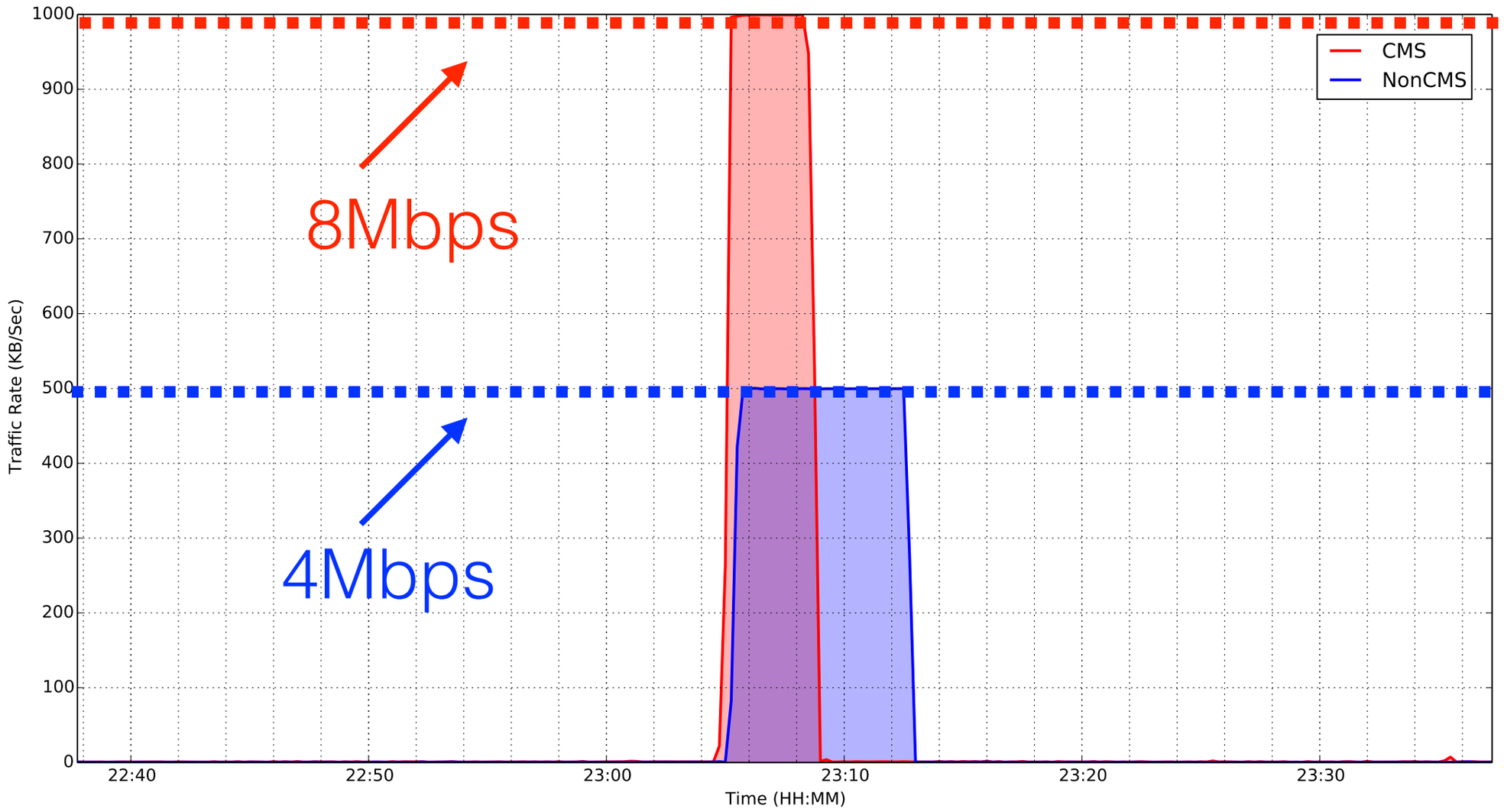


Figure 11: WAN bandwidth management for HTCondor jobs

OpenFlow-enabled GridFTP

- There are other users of WAN bandwidth besides HTCondor jobs - GridFTP is often a culprit!

- We have implemented a GridFTP callout to provide the controller with an association of application-level file transfer info (such as *transfer type*, *filename*, *username*) with a TCP flow.

- The controller can then prioritize GridFTP traffic in a manner similar to HTCondor traffic.

GridFTP example

- Take three GridFTP clients, each downloading a file of 200MB from the GridFTP server.

- Client1 -> /test1/200MB
  Client2 -> /test2/200MB
  Client3 -> /test3/200MB

- We prioritize different access based on directory:
  /test1/* -> 4Mbps
  /test2/* -> 2Mbps
  /test3/* -> 1Mbps

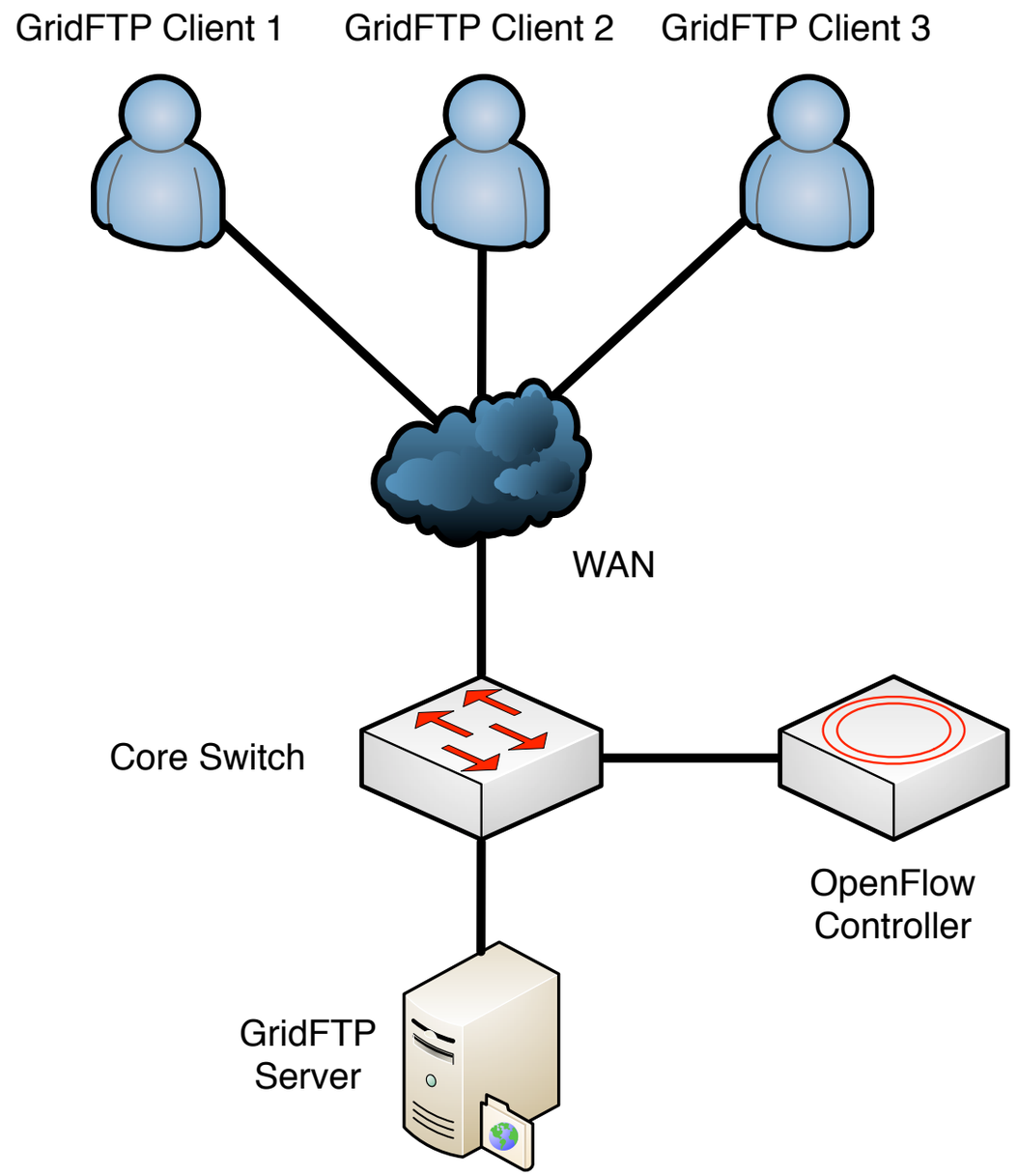


Figure 12: GridFTP experiment setup

Experiment Result

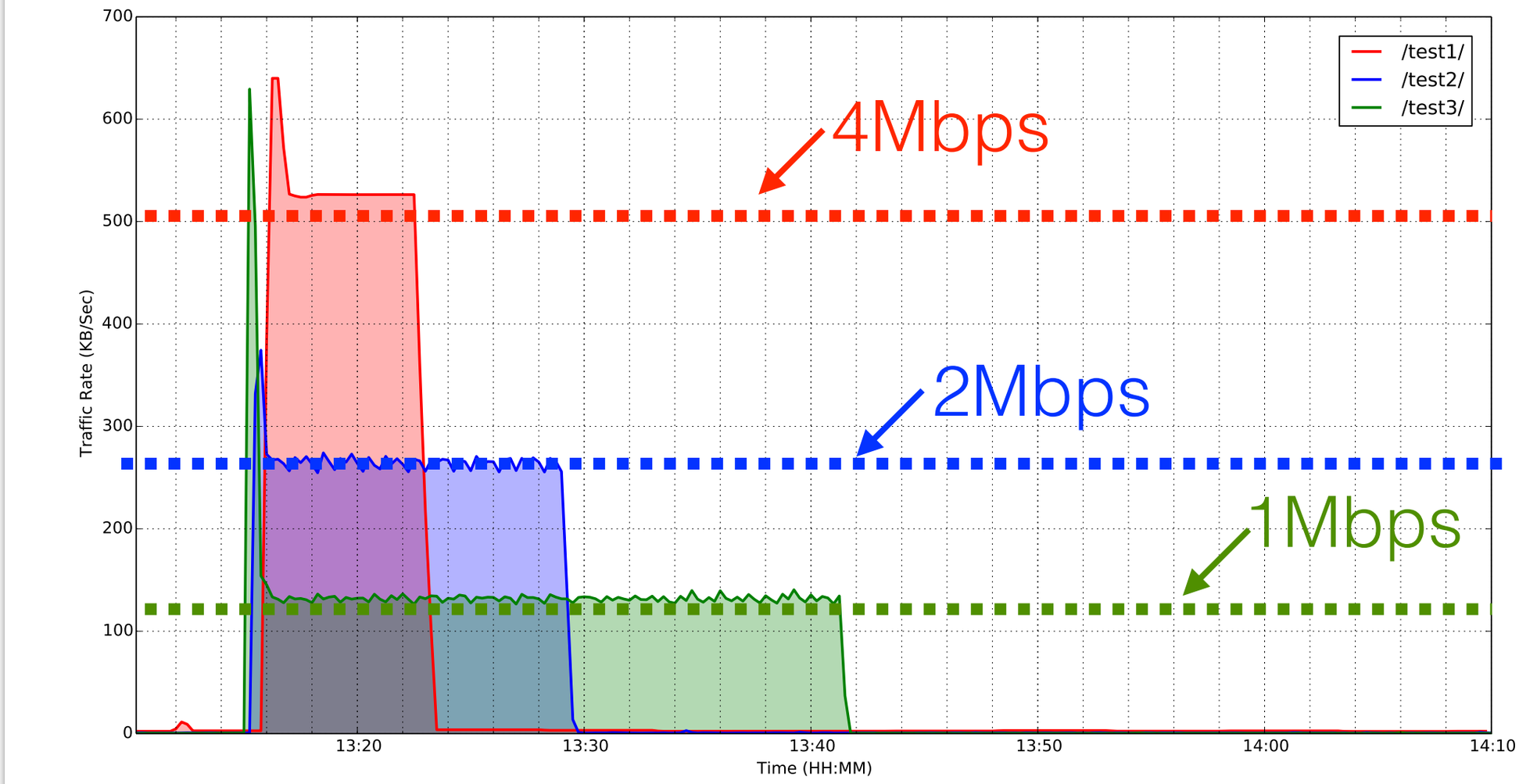


Figure 13: GridFTP file transfers when downloading files in different directories.

Combining the examples

- Each project likely has network traffic from multiple applications (HTCondor + GridFTP).
- We want to direct network traffic for all the supported applications in one project to its associated QoS queue.
  - Traffic competes according to normal TCP rules intra-project but bandwidth is protected inter-project.

- We combine previous HTCondor and GridFTP example together.
- HTCondor job: upload 200MB file to FTP server in WAN. (CMS and NonCMS)
- GridFTP clients download file of 200MB from GridFTP server in cluster. (also CMS and NonCMS users)

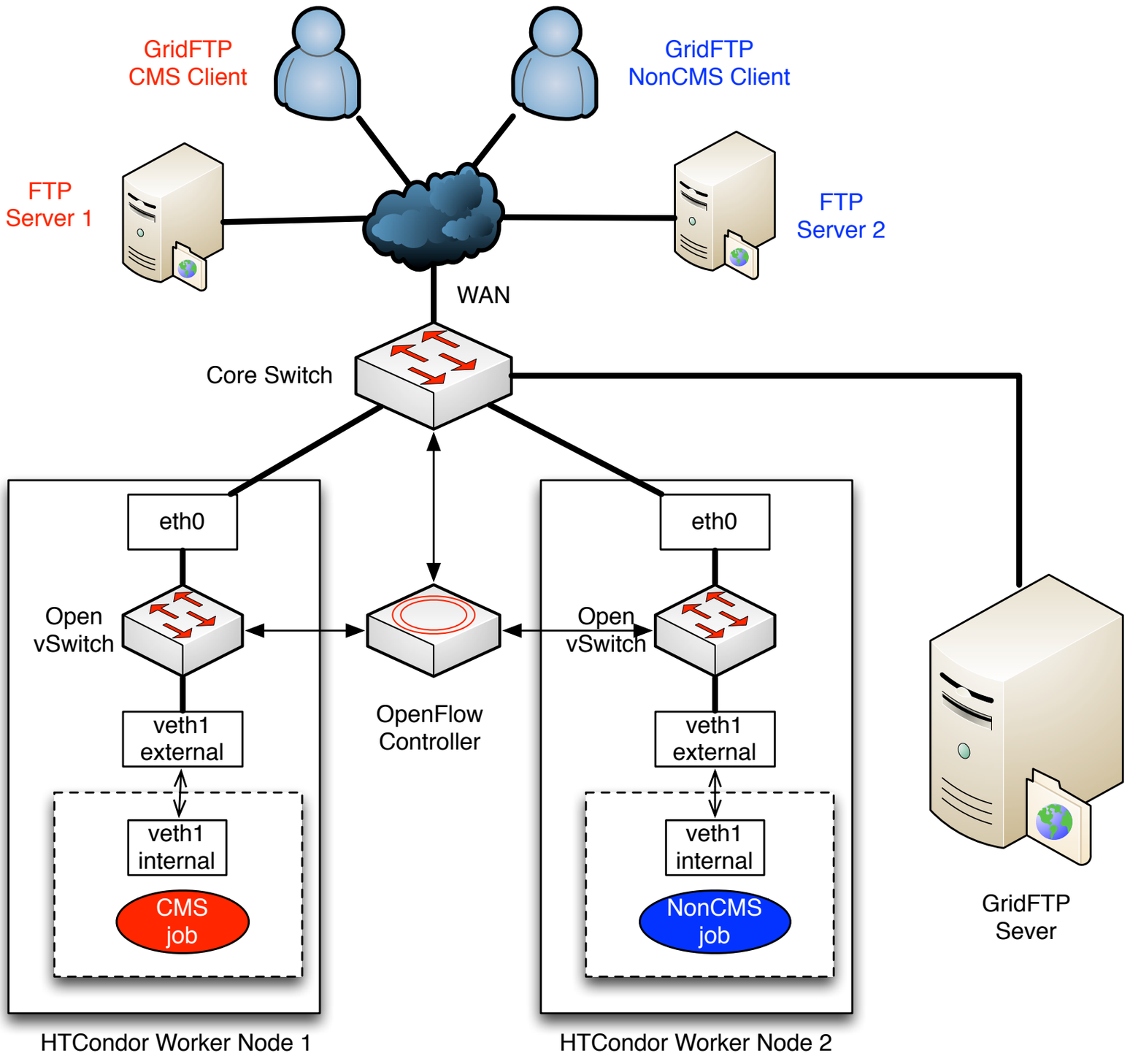
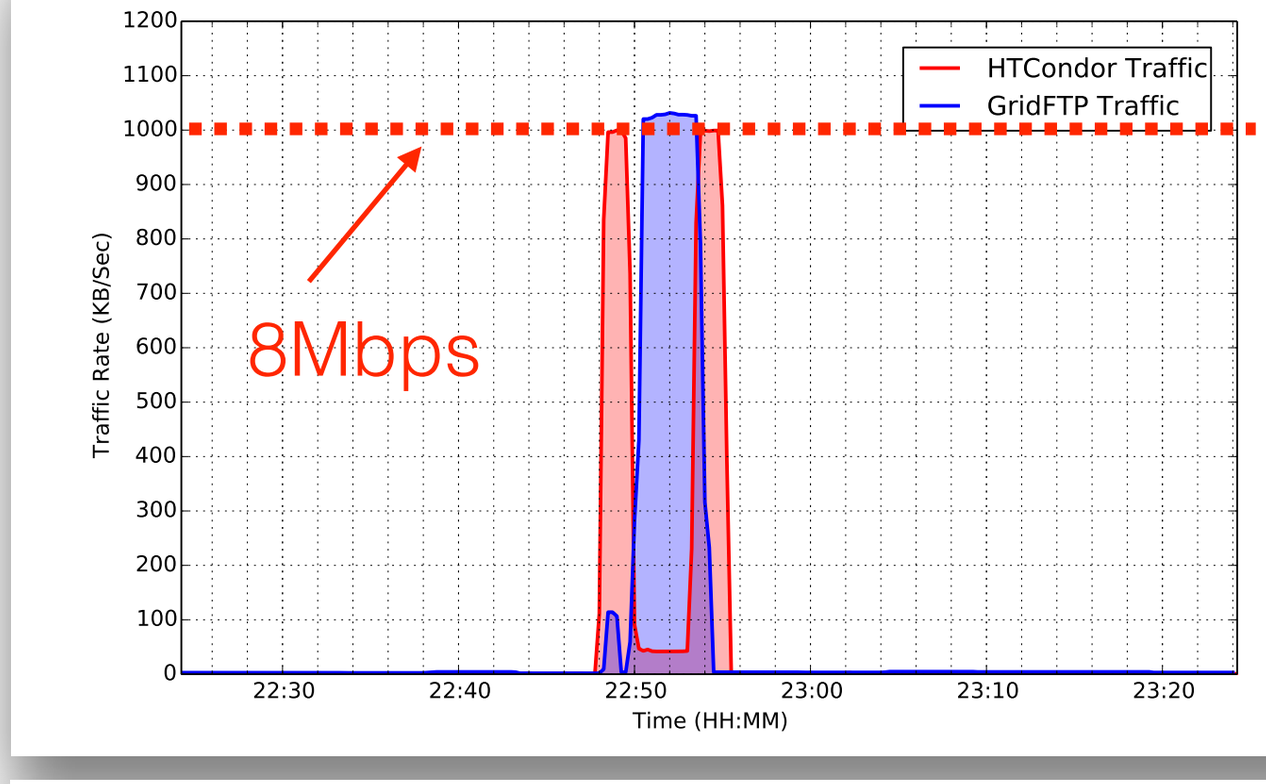


Figure 14: HTCondor jobs + GridFTP file transfer

- CMS project has a queue with bandwidth 8Mbps.
- NonCMS project has a queue with bandwidth 4Mbps.
- Within each project, GridFTP traffic grabs most of the bandwidth when both of the two applications are running.
- Across projects, their assigned total bandwidth are reserved.



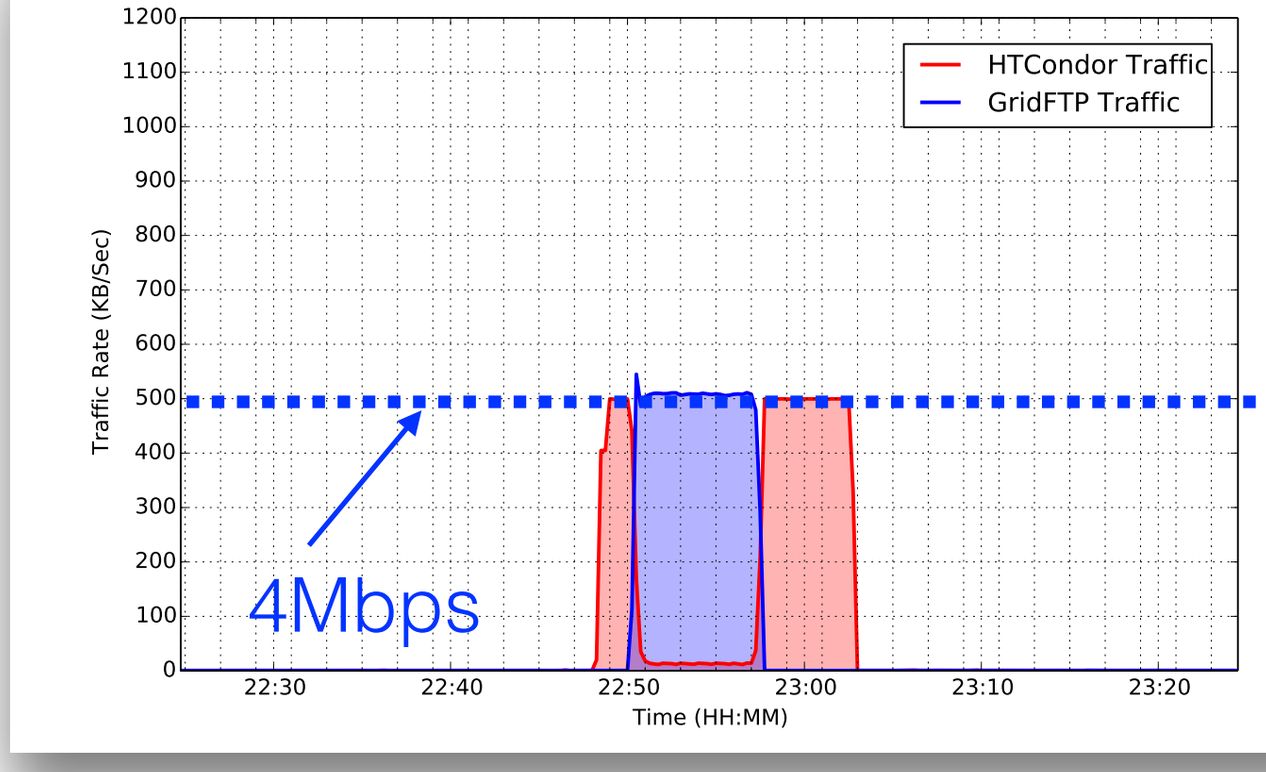


Figure 15: WAN bandwidth management for multiple applications

Conclusion

- HTCondor has no mechanism to do network resource management. With network namespace, extensible machine resource and Open vSwitch, we can manage network at host level.

- Further by integrating SDN with this framework, we can manage WAN bandwidth resource for HTCondor and other applications.

Future Work

- Support more cluster computing softwares to investigate network traffic characteristics within project and across project.

- Get current work deployed in production cluster and do real-world measurements.

Acknowledgement

- Nebraska: Brian Bockelman, Garhan Attebury
- Wisconsin: Alan DeSmet, Dale W. Carder, Todd Tannenbaum
- NSF Funding ACI-1245864

Thanks!

Questions?

Backup Slides

A more complicated example for host-level bandwidth management

- Enforce total bandwidth resource at a worker node to be 10Mbps.

- Two users submit several `curl` jobs. Each of them downloads a 200MB file with bandwidth requirement.

- User A requests for 1Mbps; User B requests for 9Mbps.

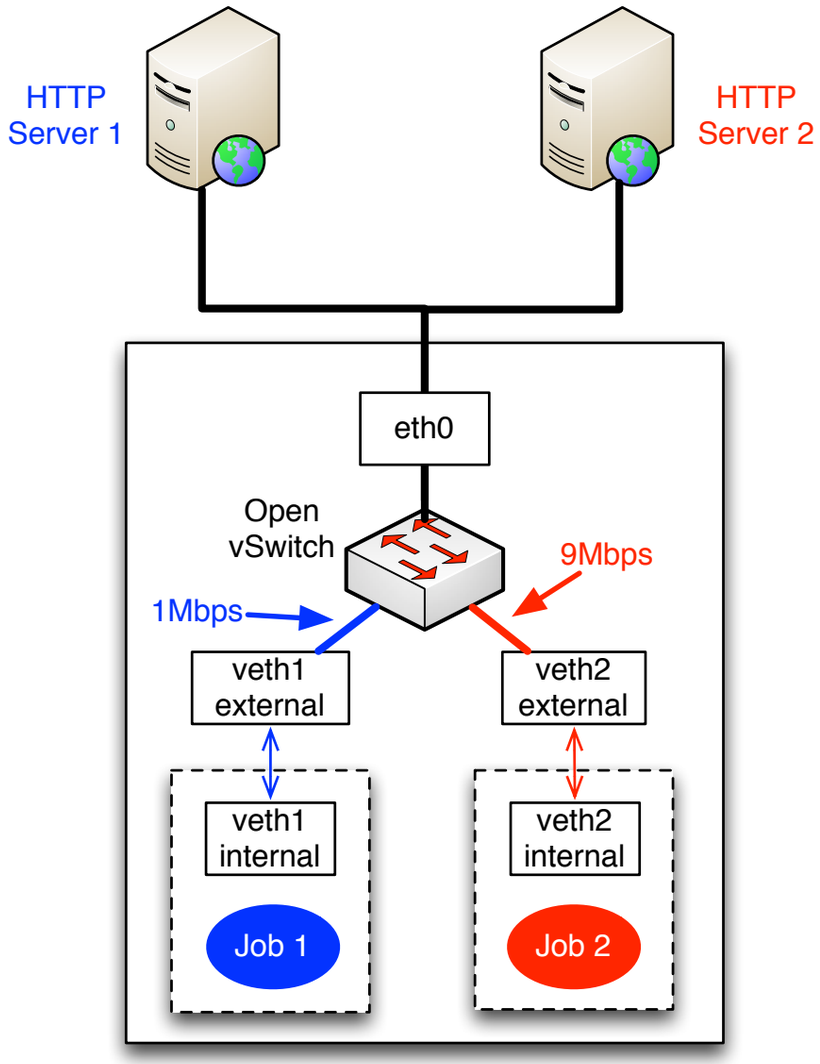


Figure : Worker node network topology for multiple jobs

- Due to host-level bandwidth management, two jobs can run at the same time (to saturate the total bandwidth resource), and jobs get their requested bandwidth.

- Without bandwidth resource requirement, when two jobs from different users are running, each of the job is expected to get 5Mbps bandwidth on average (equal competition).

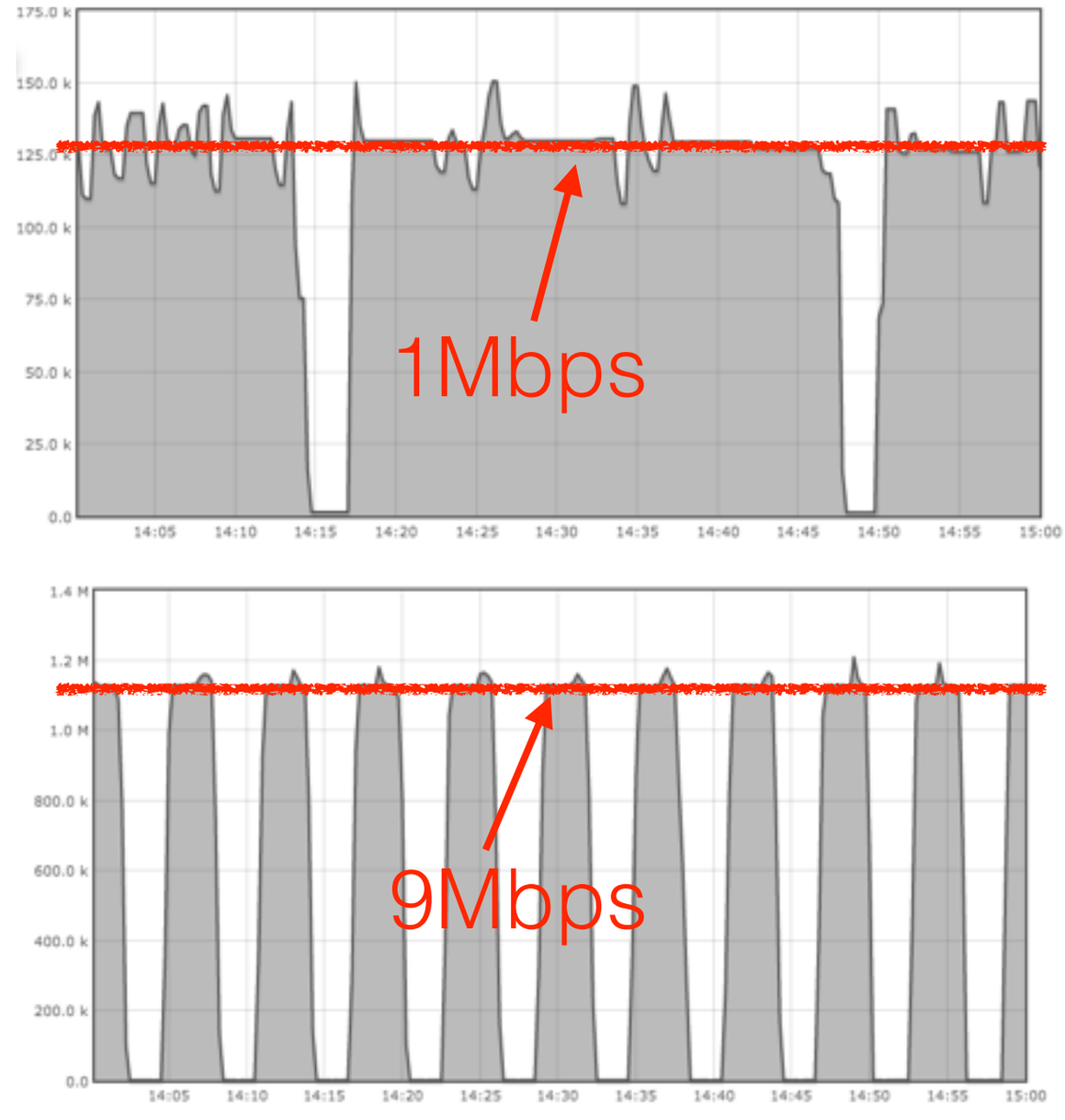


Figure : HTCondor job traffic for two users

Attributes	Meaning
in_port	Switch port number packet arrives
dl_src	Ethernet source address
dl_dst	Ethernet destination address
dl_vlan	VLAN ID
dl_vlan_pcp	VLAN priority
dl_type	Ethertype/length (e.g. 0x0800=IPv4)
nw_tos	IP TOS/DS bits
nw_proto	IP protocol (e.g., 6=TCP)
nw_src	IP source address
nw_dst	IP destination address
tp_src	TCP/UDP source port
tp_dst	TCP/UDP destination port

Table 1: Packet match field

OpenFlow Actions	Class
Output	ofp_action_output
Enqueue	ofp_action_enqueue
Set VLAN ID	ofp_action_vlan_vid
Set VLAN priority	ofp_action_vlan_pcp
Set Ethernet src or dst address	ofp_action_dl_addr
Set IP src or dst address	ofp_action_nw_addr
Set IP Type of Service	ofp_action_nw_tos
Set TCP/UDP src or dst port	ofp_action_tp_port

Table 2: Available OpenFlow actions (1.0)

Example Network Policies for HTCondor

- Block network traffic from specific HTCondor users.
- Isolate network traffic among HTCondor users.
- Block specific HTCondor users to communicate with outside network.
- And, of course, manage WAN traffic!