

Bringing High Throughput Computing to the Network with Lark

Outline

- Since when was the network missing?
- Resource management in HTCondor.
- Network Namespaces in HTCondor.
- Networking-aware policies and policy-aware networks.
- Expanding the network ecosystem.
- Futures.

Part I:
**Since when was the
network missing?**

In the beginning...

- The network has never been explicitly managed by the HTC layer.
- Instead, it was implicitly assumed to always be there - like power. Like power, if it's not present, it was considered outside the upper layer's control.
- Within HTCondor, a lot of logic is devoted to retries in case the network is unreliable. However, retries - while valuable - are not equivalent to management.
- For quite some time, this has been a reasonable assumption.
- We ran batch systems on clusters and HTC-oriented clusters tended to have flat, boring networks.

But Wait!

- Some of these assumptions have been changing:
 - HTCondor is used to manage and schedule resources in overlay pools which may have a very heterogeneous underlying network.
 - Jobs are increasingly network connected - gone is the day where **all** jobs have a few KB of input, a few KB of output, and run for hours.
 - We see multi-GB input/output sandboxes, jobs that read in multi-GB, and jobs that need access to a firewalled server.
- These are the jobs we run! Think about the jobs we ignore - what if my “job” serves data to others in a pipeline?

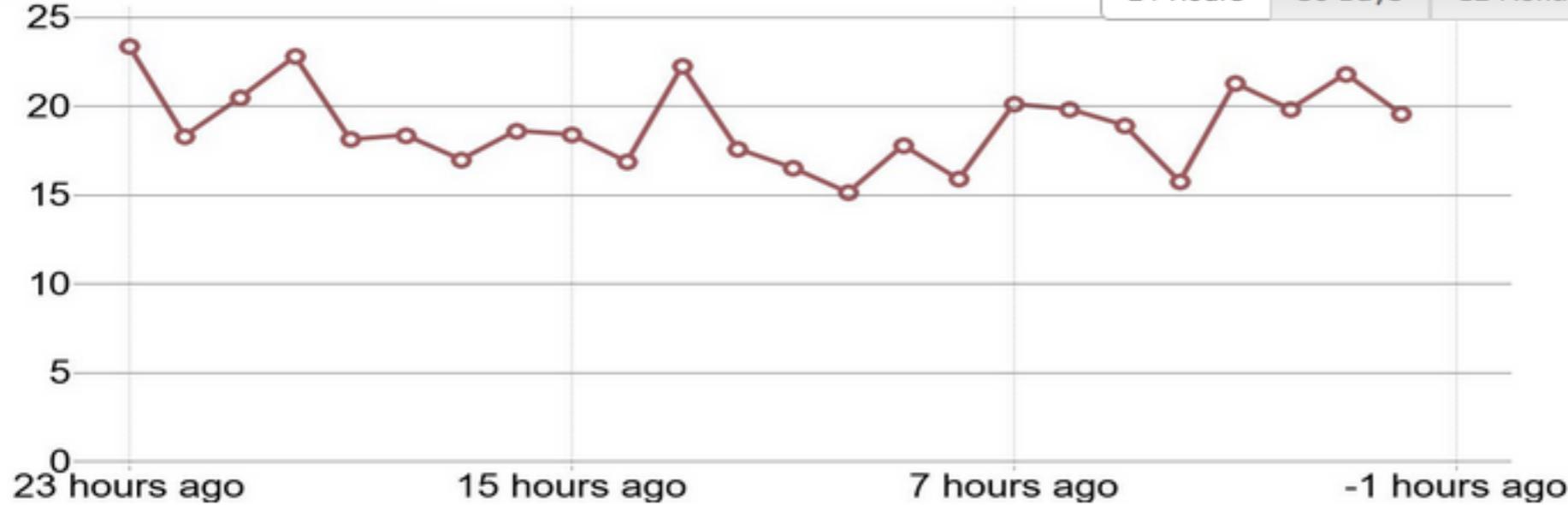
Open Science Grid

A national, distributed computing partnership for data-intensive research

Jobs CPU Hours Transfers TB Transferred Status Map

Thousands of Jobs/Hour

24 Hours 30 Days 12 Month



Each finished job on an OSG resource is reported to the central accounting system. The above graph shows the number of completed jobs per hour. A total of 454,000 jobs were completed.

OSG delivered across 117 sites

In the last 24 Hours

454,000	Jobs
1,744,000	CPU Hours
2,196,000	Transfers
1,041	TB Transferred

In the last 30 Days

13,975,000	Jobs
57,083,000	CPU Hours
56,541,000	Transfers
28,033	TB Transferred

In the last Year

167,720,000	Jobs
715,168,000	CPU Hours
708,367,000	Transfers
363,627	TB Transferred



Status at 1:15 PM
Data is loaded every 15 minutes



Open Science Grid

A national, distributed computing partnership for data-intensive research

Jobs CPU Hours Transfers TB Transferred **Status Map**



OSG delivered across 117 sites

In the last 24 Hours

450,000	Jobs
1,739,000	CPU Hours
2,196,000	Transfers
1,041	TB Transferred

In the last 30 Days

13,975,000	Jobs
57,083,000	CPU Hours
56,541,000	Transfers
28,033	TB Transferred

In the last Year

167,720,000	Jobs
715,168,000	CPU Hours
708,367,000	Transfers
363,627	TB Transferred

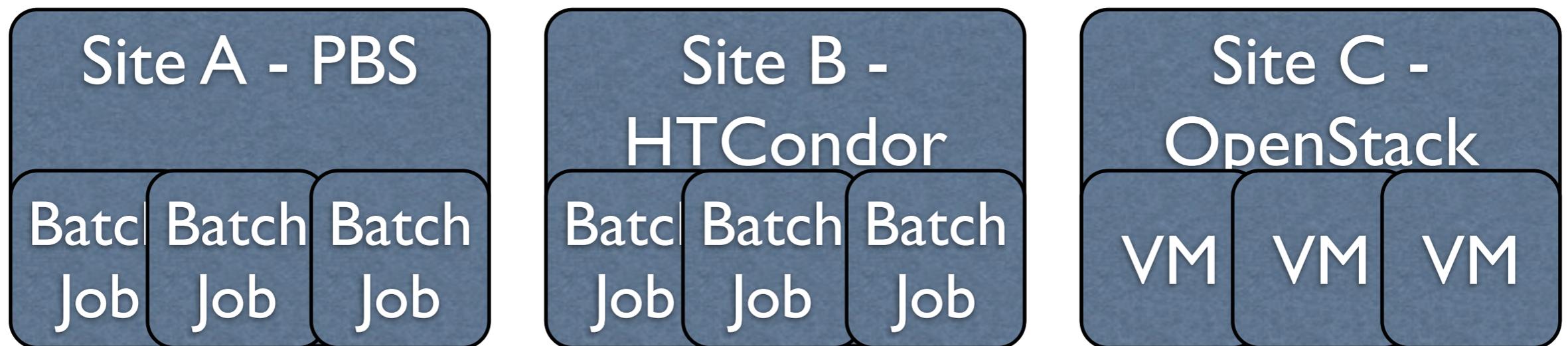


Status at **1:30 PM**
Data is loaded every 15 minutes



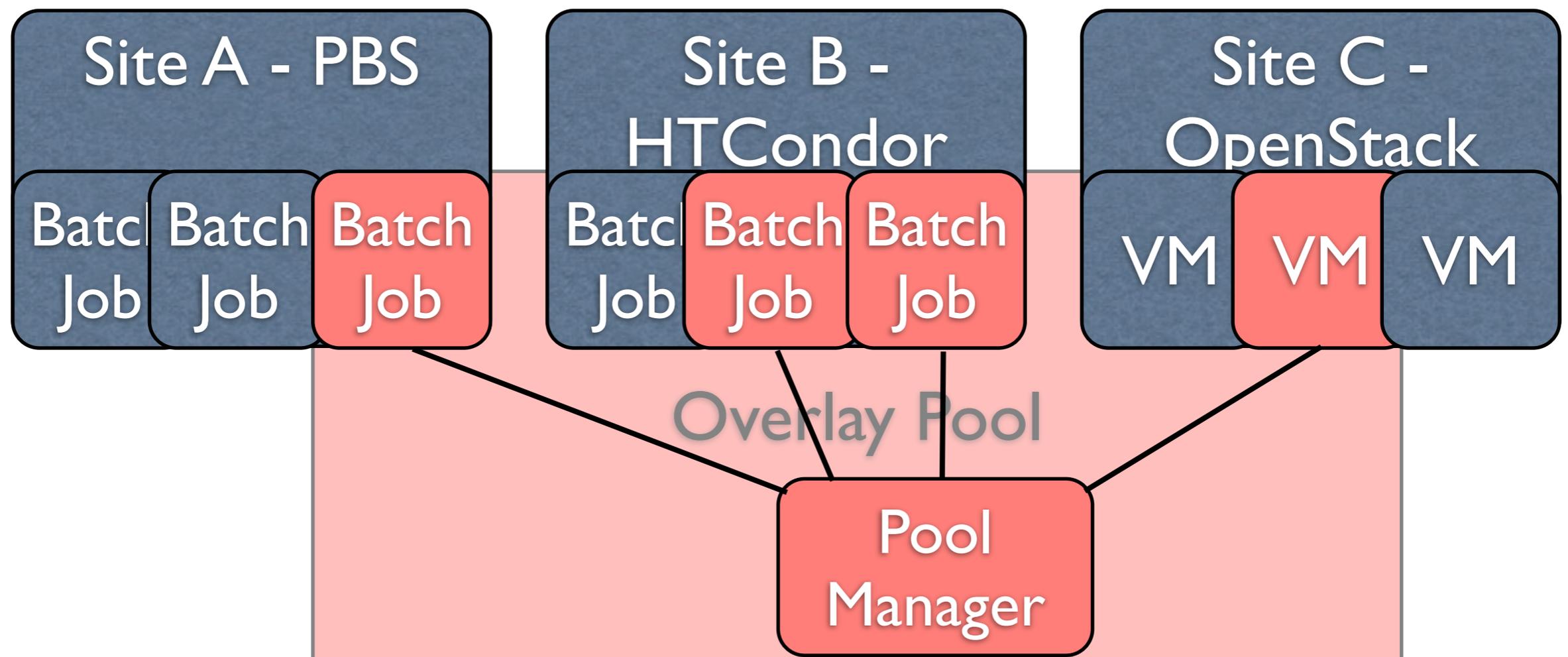
Overlay Pools

Suppose we have three sites which allocate resources - a batch job or a VM - to us.



- An overlay pool is formed when we treat the aggregate of our resources as a single batch pool.
- Batch jobs which create the overlay pool are referred to as *pilots* or *glideins*.

Overlay Pools



It's probably reasonable to assume a homogeneous network at each site - but no safe assumptions can be made between sites!

So What?

- These deficiencies pop up in various places in our HTCondor usage:
 - Long transfer wait times.
 - Priority inversion due to transfers.
 - Scheduler continues to accept matches when underwater.
 - Or even trying to start jobs requiring 1TB of input at a site with a 1Mbps network connection.
 - We don't even consider crossing / tunneling over network boundaries!

And What Now?

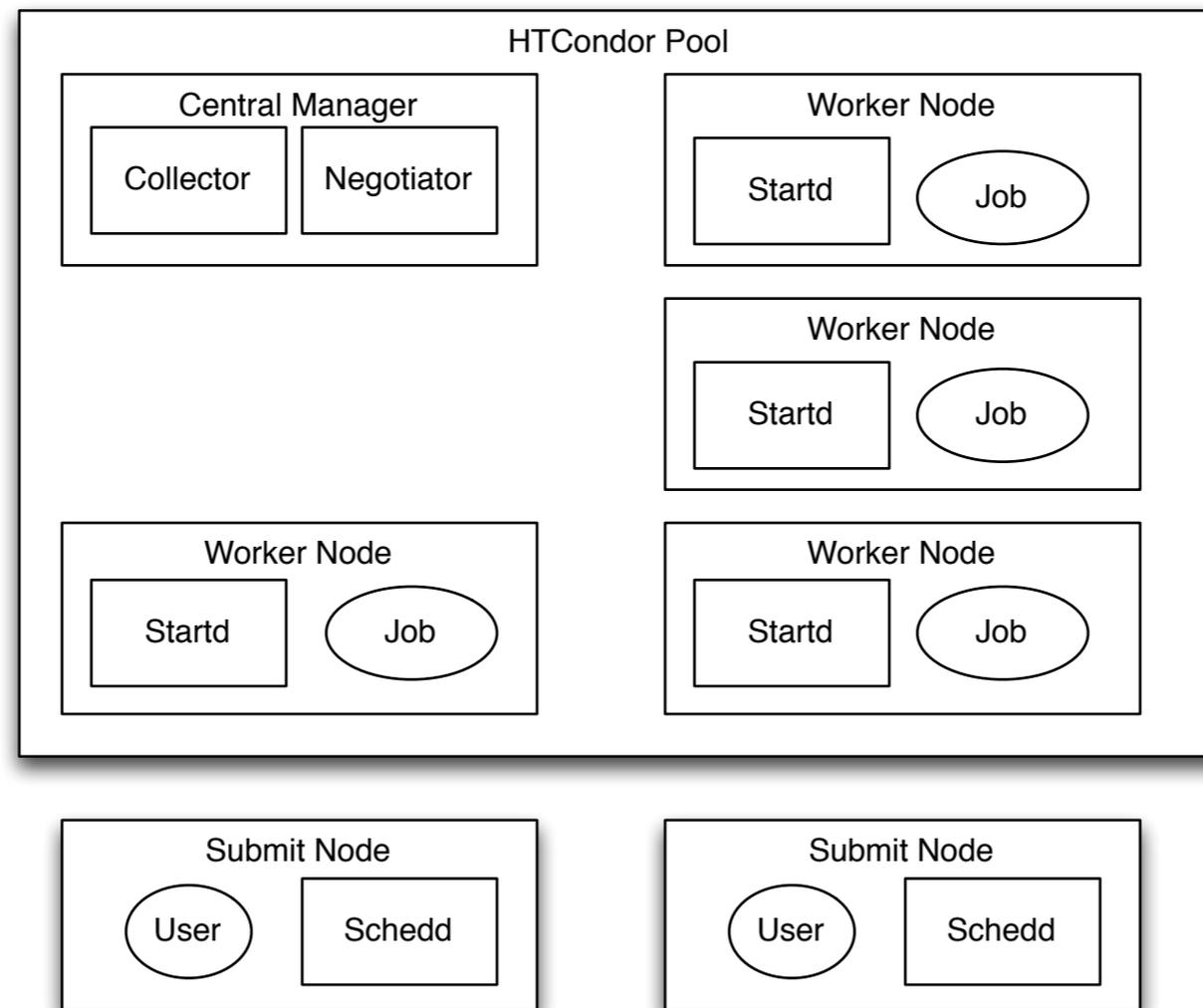
- Lark!
- **Reactively** - Provide network performance monitoring data into the HTCondor ecosystem, taking it into account for scheduling purposes.
- **Proactively** - Collaborate with & manipulate the network to satisfy job policies.

Lark

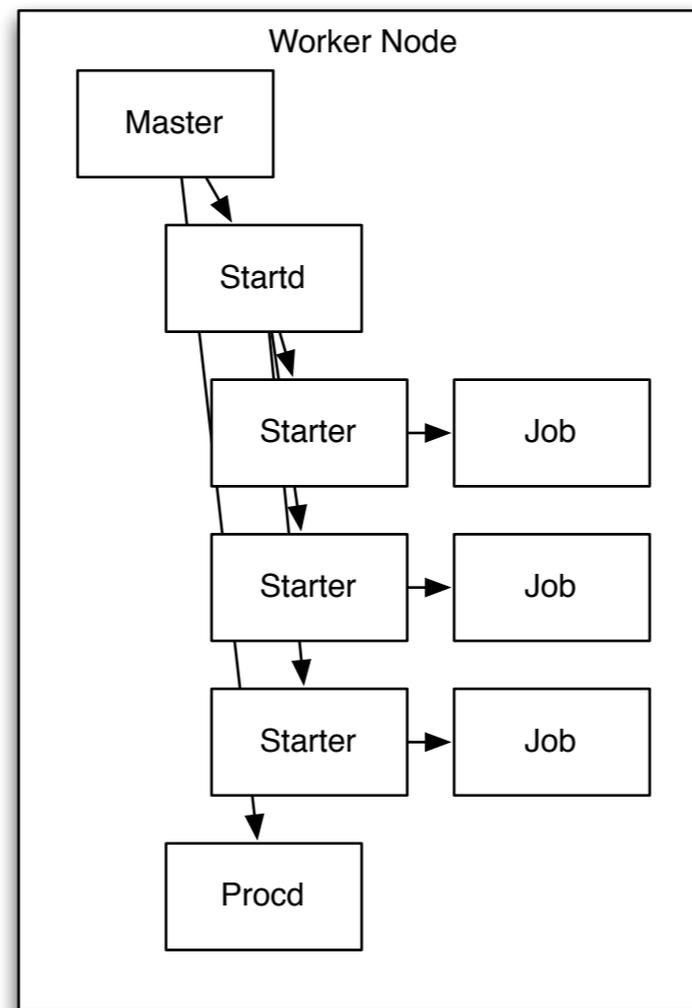
- NSF-funded (NSF #I245864), 2-year project through the CC-NIE program. We want to take the first steps at bridging DHTC and the network layer.
- Main focus is integrating HTCondor with advanced network technologies.
- Includes hardening/deploying IPv6 support, integrating with perfSONAR, DYNES, and OpenFlow. Will touch on all these today.

Part II: Resource Management in HTCondor

Anatomy of a HTCondor Pool



HTCondor Worker Nodes



Managing Resources - The Art of Making Boxes

- The perfect box:
 - Allows nesting - sub-delegate resources.
 - Does not require superuser privileges.
 - Can't be escaped.
 - Portable across OSs
 - Allows full management - creation/
destruction, monitoring, limiting.

[http://research.cs.wisc.edu/htcondor/HTCondorWeek2013/
presentations/ThainG_BoxingUsers.pdf](http://research.cs.wisc.edu/htcondor/HTCondorWeek2013/presentations/ThainG_BoxingUsers.pdf)

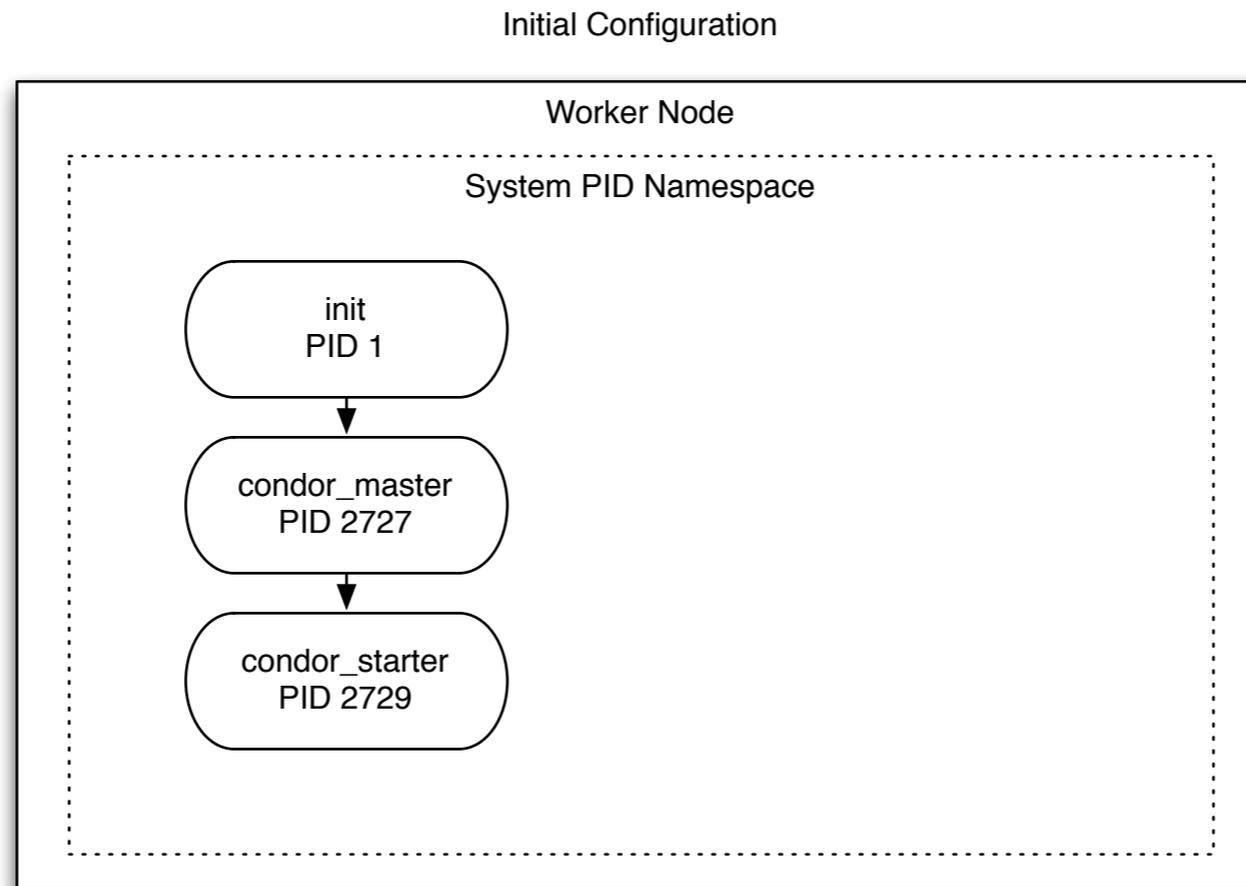
Resource Management

- Resource management in HTCondor happens in several layers / daemons. All of them need to be made network-aware.
- **Collector** - database of all known resources and resource requests.
- **Negotiator** - matches available resources and resource requests based on policy.
- **Schedd** - Manages and schedules a queue of jobs; posts the resource requests to the collector.
- **Startd** - Manages the available resources on a worker node.
- **Starter** - Configures resources and launches the jobs.
- **Procd** - Monitors and manages processes on the worker node.

Putting Users in a Box

- The HTCondor project has been increasingly focused on resource management on the worker node:
 - **Accounting:** CPU, memory usage, block IO. (Mostly via cgroups)
 - **Isolation:** PID namespaces, chroots, per-job /tmp directories. (Mostly via namespaces)
 - **Resource management:** CPU fairshare & affinity, memory limits, guaranteed process killing. (Mostly via namespaces, cgroups, and obscure syscalls)
- What's missing? The network!

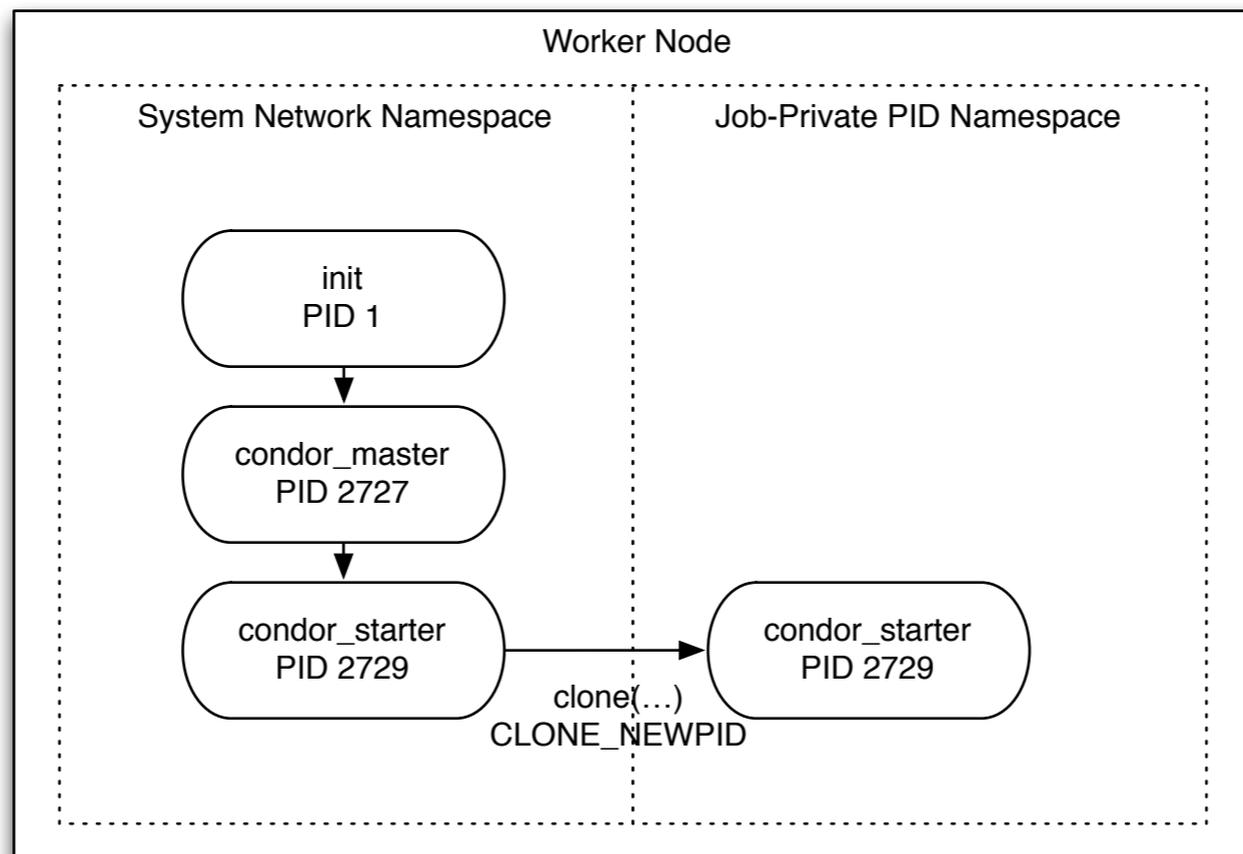
Example - PID Namespaces



The PID namespace is a system resource configured by the starter for the job.

Example - PID Namespaces

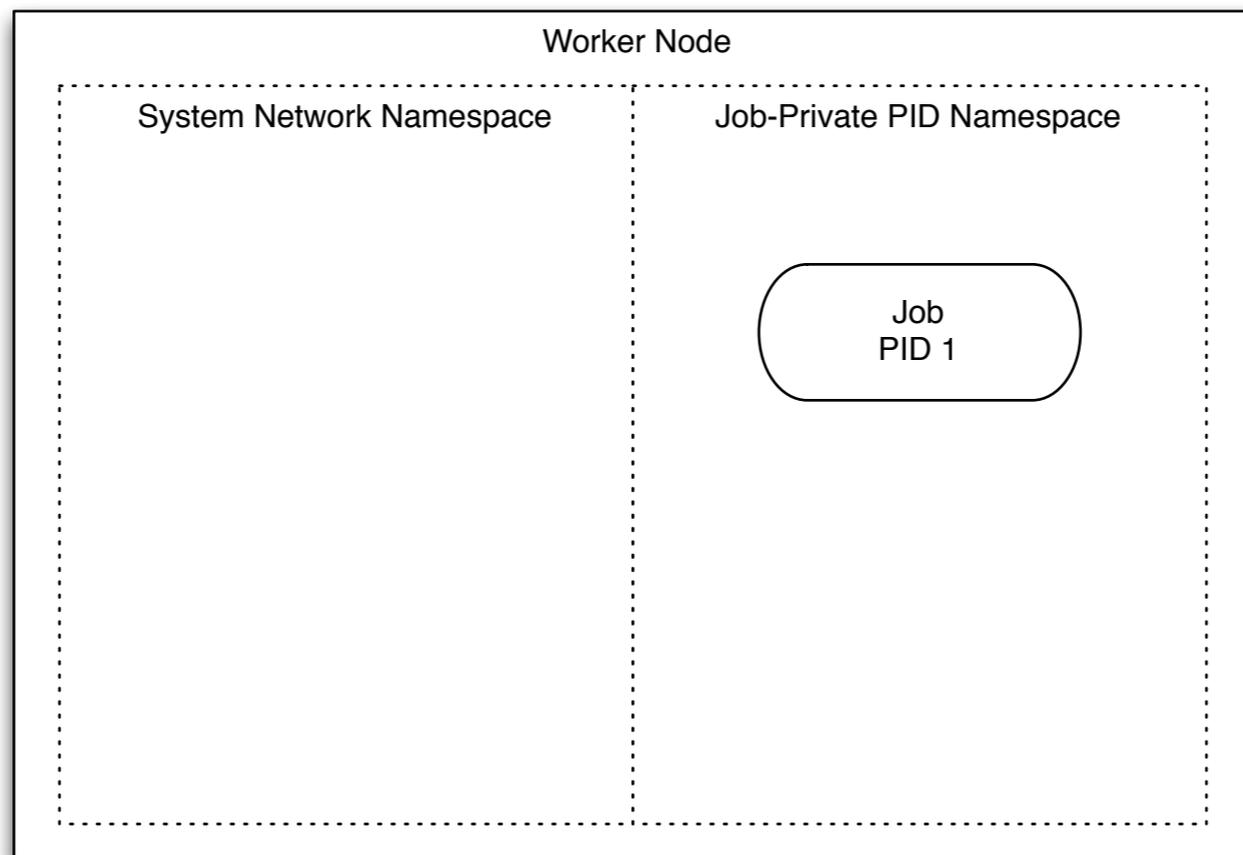
Starter forks a process using the CLONE_NEWPID,
creating a new namespace



View from outside

Example - PID Namespaces

condor_starter exec's the job process



View from inside

The job believes itself to be PID 1 - the namespace provides a different view of the system

Part III:

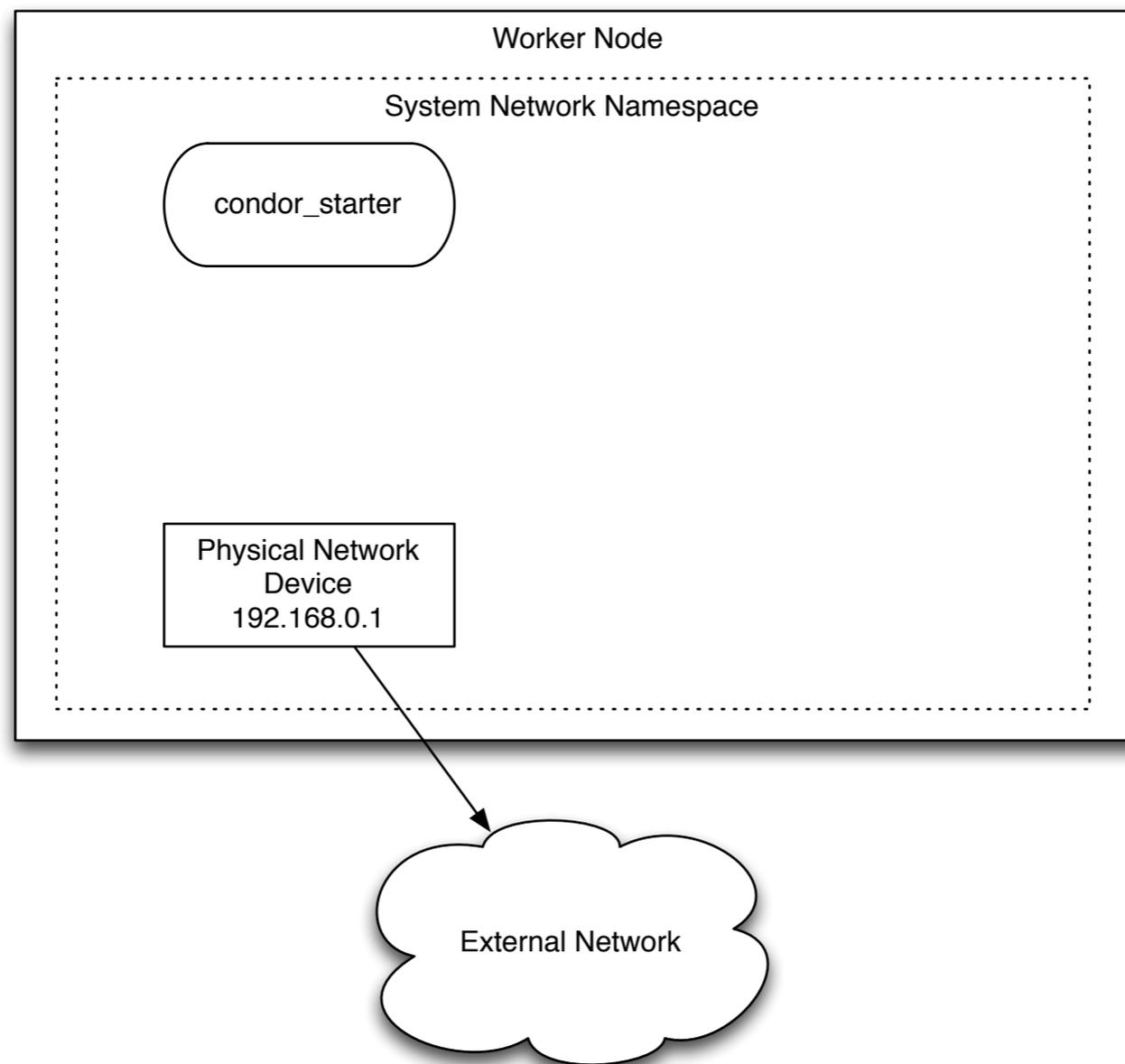
Network Namespaces

A network box

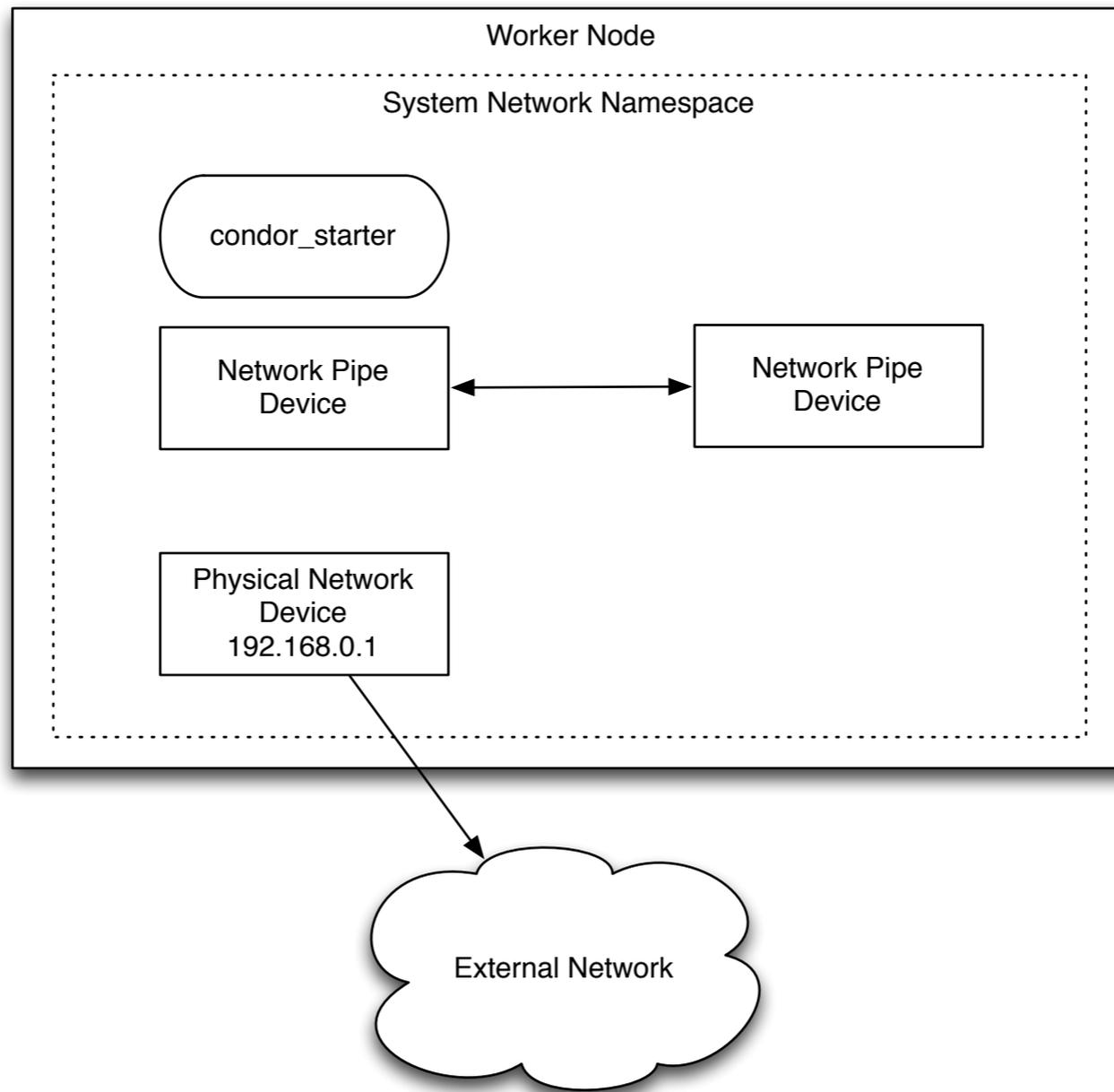
- We want to provide a network interface per job and force the job to work exclusively with this interface.
- This will allow the job to be directly addressable by the network.
- Manage the interface and you manage the job's network.
- In Linux, we can isolate a set of processes to a specific set of network interfaces using a *network namespace*. Along with a pair of *virtual ethernet pipe* devices, we can integrate the job to the network.

Network Namespace - A flipbook

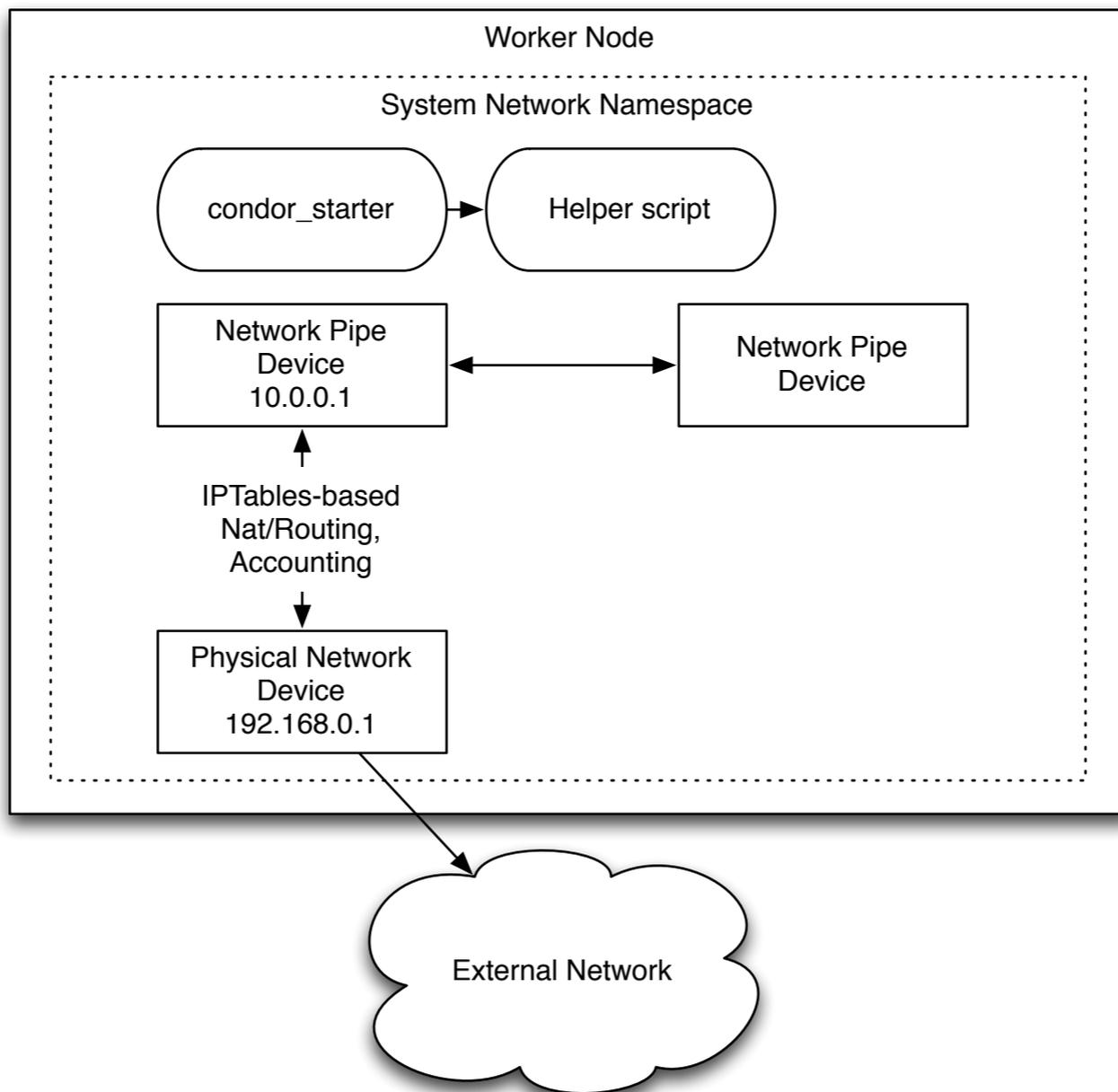
Initial Configuration



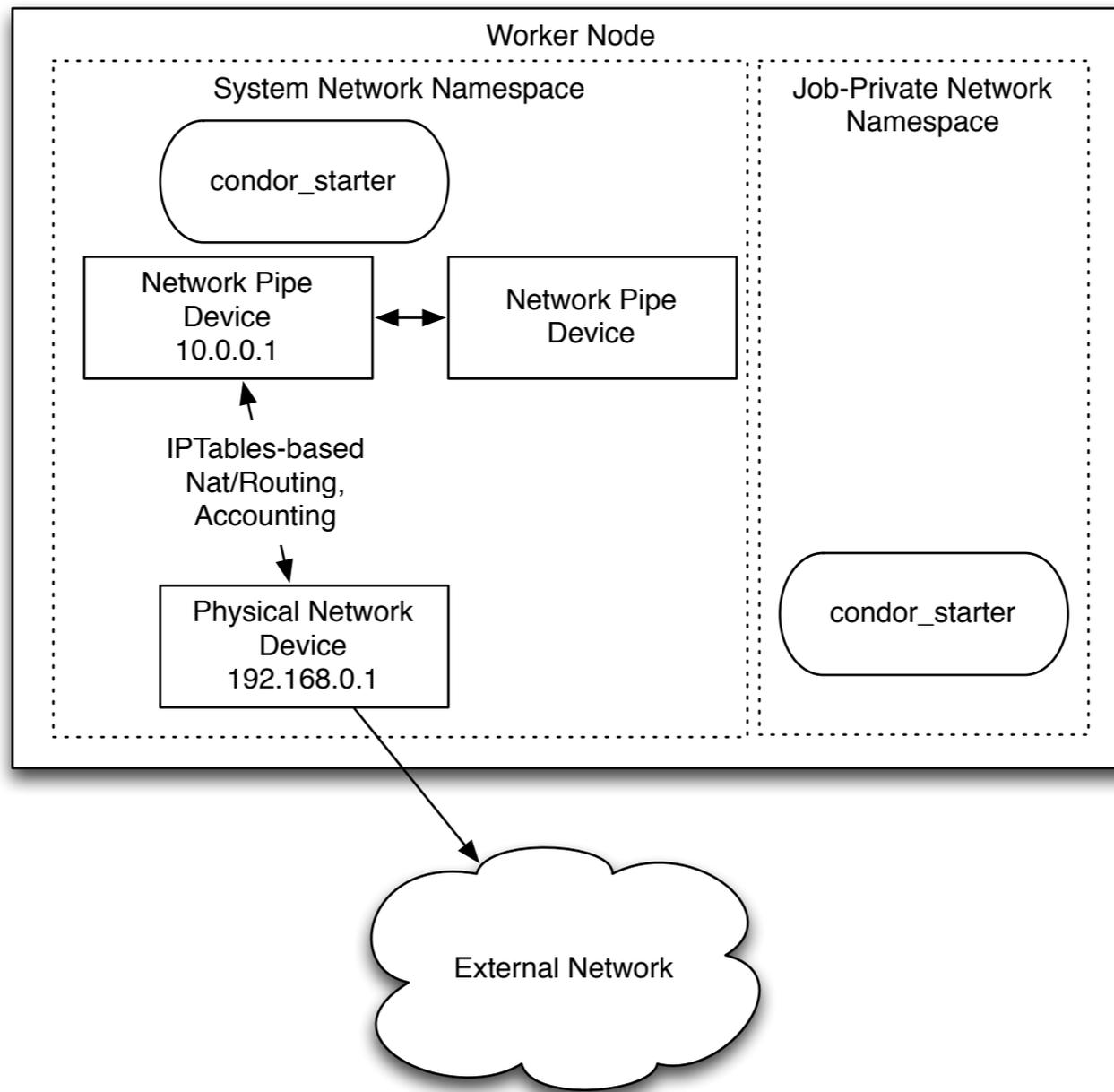
Starter creates network pipes



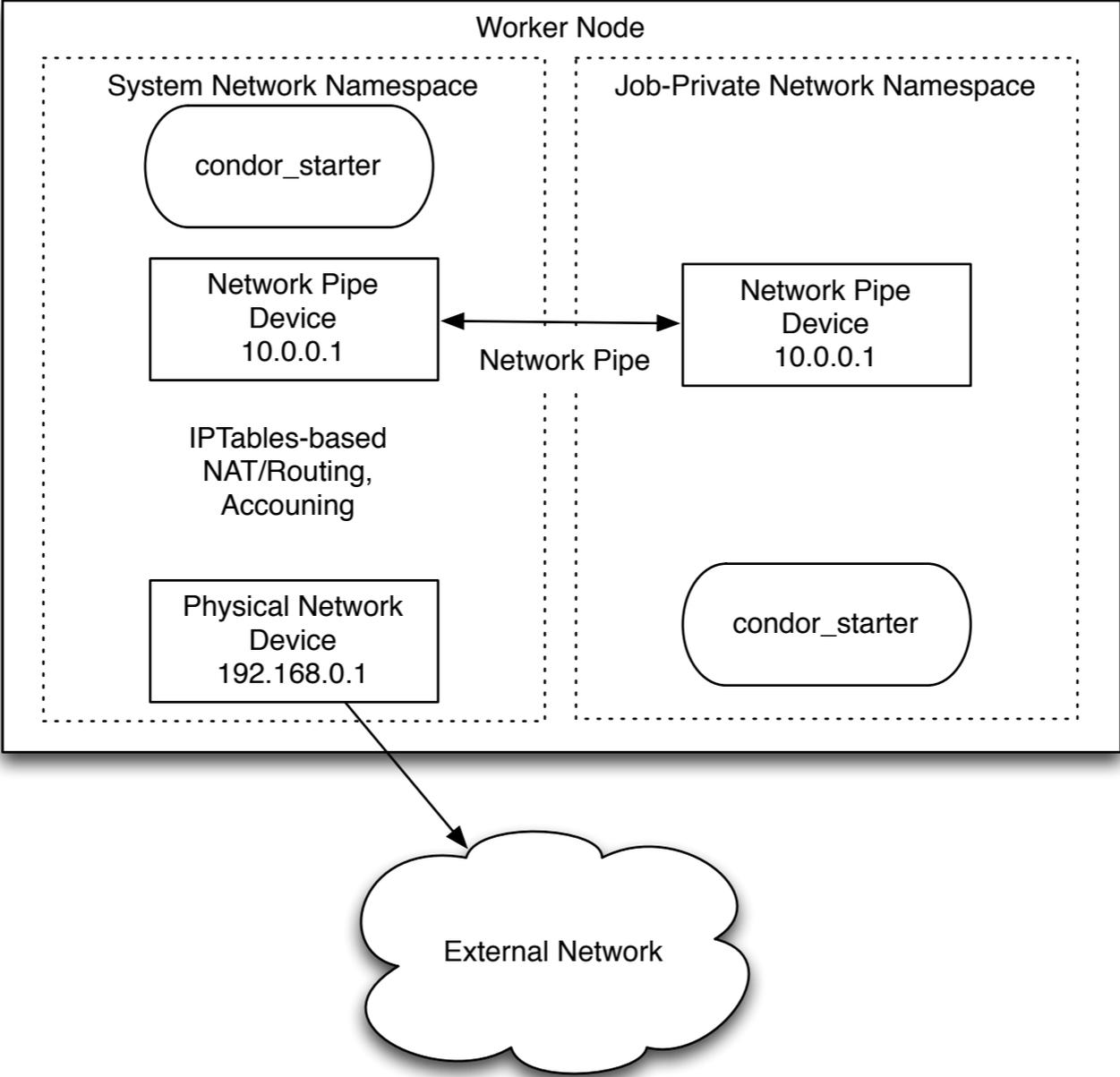
Starter creates a helper process
Helper configures IPTables and assigns addresses



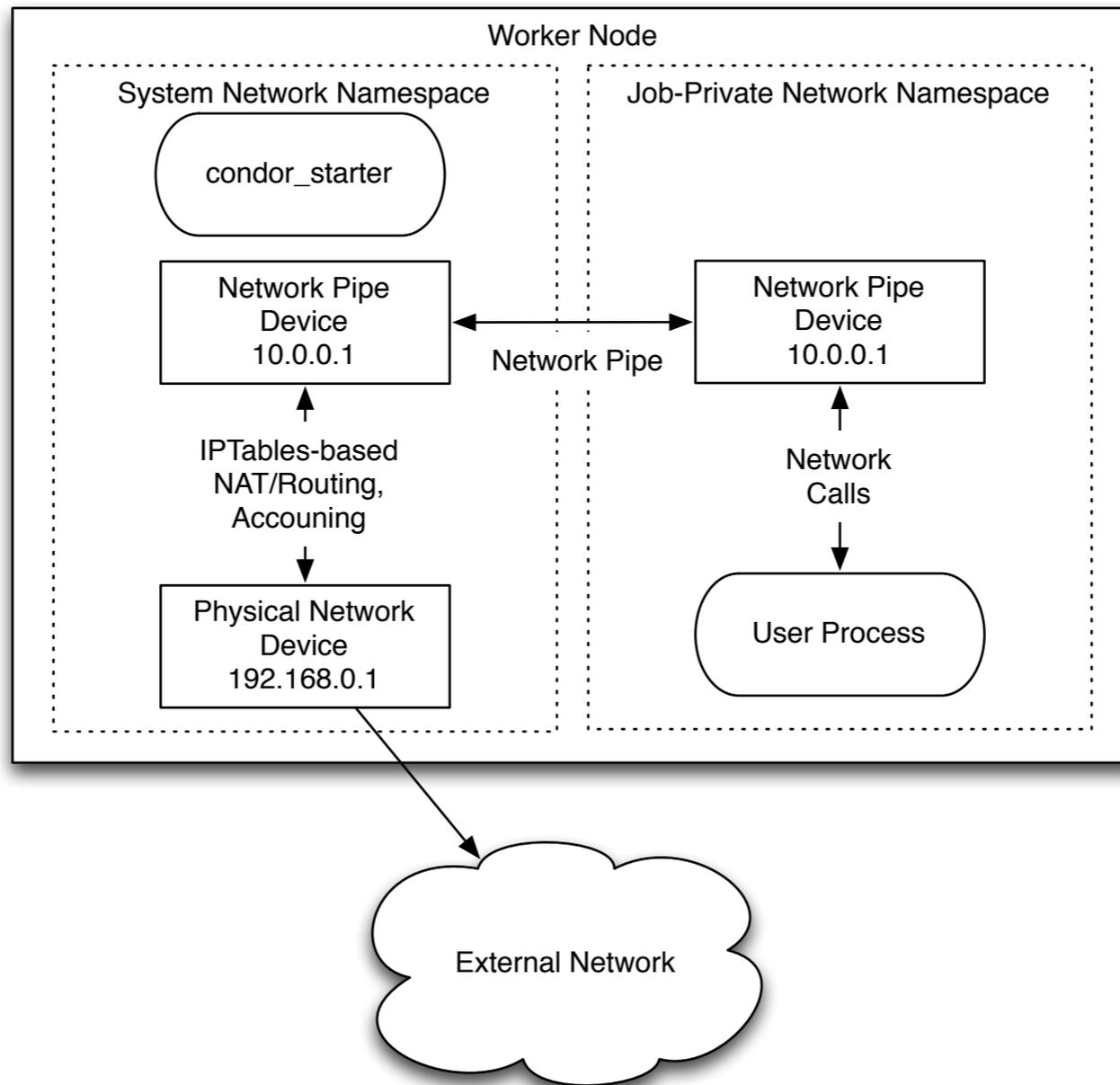
Starter forks new process with new network namespace



Parent starter passes one end of network pipe to network namespace,
Child starter configures routing and IP address



Final Configuration



Capabilities

- The last few slides illustrated configuring a NAT.
- Bridging the job behaves similarly, except we now need to figure an appropriate network configuration:
 - **Statically** - admin specifies an available range of IP addresses HTCondor can use, per node.
 - **DHCP** - Discovery/Offer multicast prior to device creation; Request/Ack done post-fork.
 - **IPv6** - SLAAC. (Oh please oh please oh please)

In the end

- At the end of the process, the job has its own functional network device, hooked up to the network in some way.
- Depending on whether in NAT or bridge mode, the job is addressable by the host networking subsystem or external network devices.
- (And this is where the fun starts)

Part IV:
Network-aware
policies
and policy-aware networks

Now what?

- Once we have a handle to the network, what do we want to do with it?
- **Accounting** - one of our first drivers. Given a job, how much money should I charge for network usage?
- **Network layer management** - Force job to keep within its allocated resources.
- **Policy** - What actions is the job allowed to perform?

Job Network Accounting

- How many bytes did my job read from the network?
- Fairly straightforward to ask from a VM, maddeningly difficult for batch systems.
- We take the per-job network device and run all the packets through an IPTables chain.
- Lark sets up two no-op rules that match packets in and out, respectively. These get reported to the job accounting for HTCondor.
- There are hooks for the sysadmin to add additional rules that are read by the accounting. For example, we can separately record on-campus traffic versus off-campus traffic.

Policies

- There are some simple policies we can enforce in NAT-mode:
 - **No network:** the job has no network connectivity.
 - **Host-level bandwidth limits:** Hand out only certain amounts of host bandwidth to each job.

Bandwidth Management

- By adding additional iptables rules, we can do rudimentary host-level bandwidth management.
- We can force the jobs to share the node bandwidth equally - or even do per-user shares.
- However, in NAT-mode, we are limited to the local host's network.

Network policies

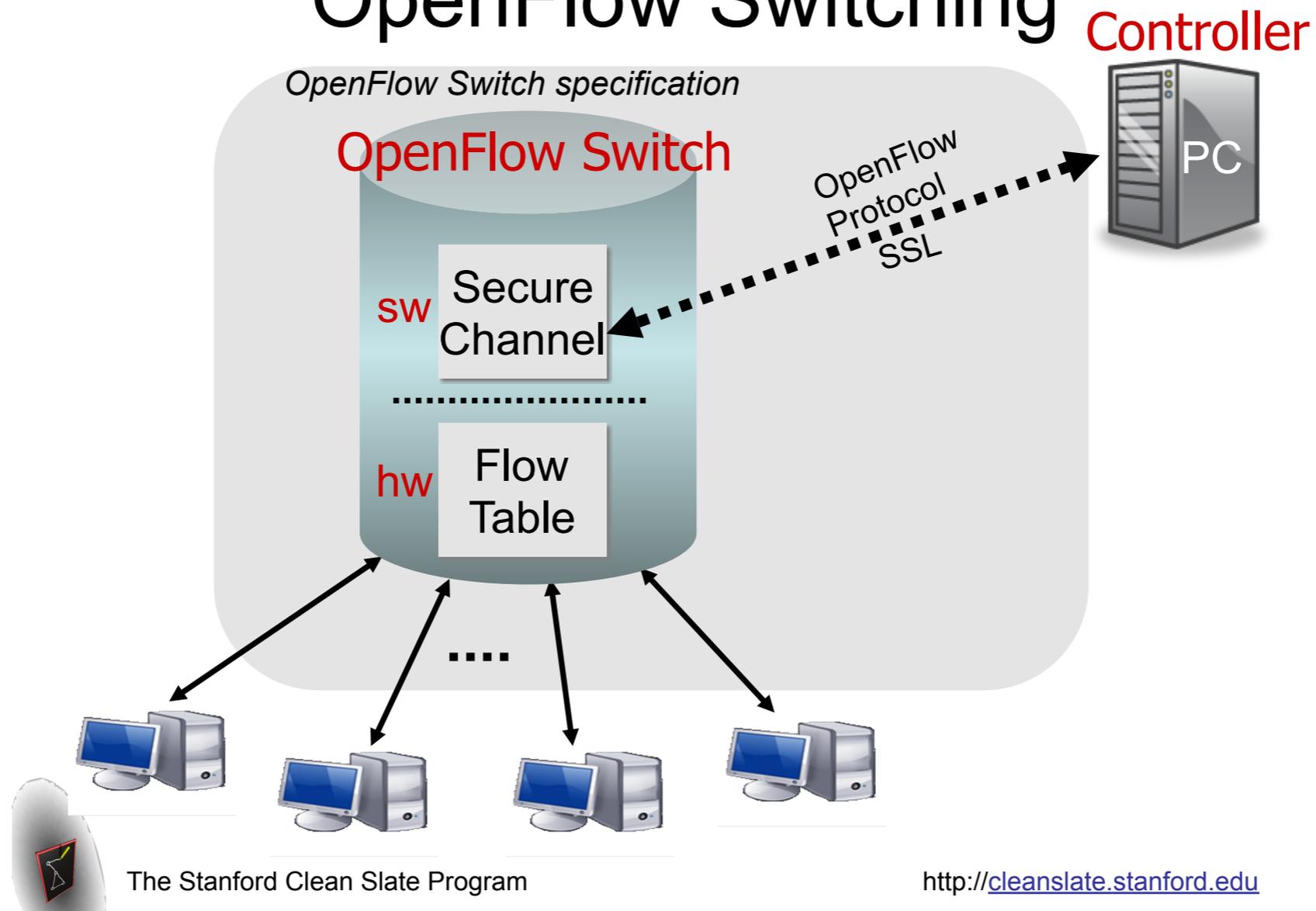
- What happens if we want to implement a policy which requires interacting with larger network?
- For example, what if a specific compute job needs access to a file server behind a firewall?
- How about limiting access to a license server to a certain set of users?
- We'll need some help from the network world...

OpenFlow and SDN (in a slide or two)

- Routers tend to be divided into the *data plane* and *control plane*.
 - The data plane is simple - a table of rules on how to handle packets, based on their headers.
 - The control plane is complex - various distributed algorithms for setting up the tables in the data plane.
 - For example, the control plane is in charge of doing shortest path calculations for packets across the network.
 - Very hard to do correctly, very expensive, changes very slowly.
- *Software defined networking* - replace the control plane silicon with a callout to a piece of software. The *OpenFlow* protocol is a standard for having the data plane managed by an external *controller* (typically a central server).
 - By replacing hardware with software and switching from decentralized with centralized, we can .
 - OpenFlow lives between the switch and the controller and is a open standard. It's a *southbound* protocol.
 - From the controller to the outside world (management, other software agents) is the *northbound* protocol. This is not standardized.

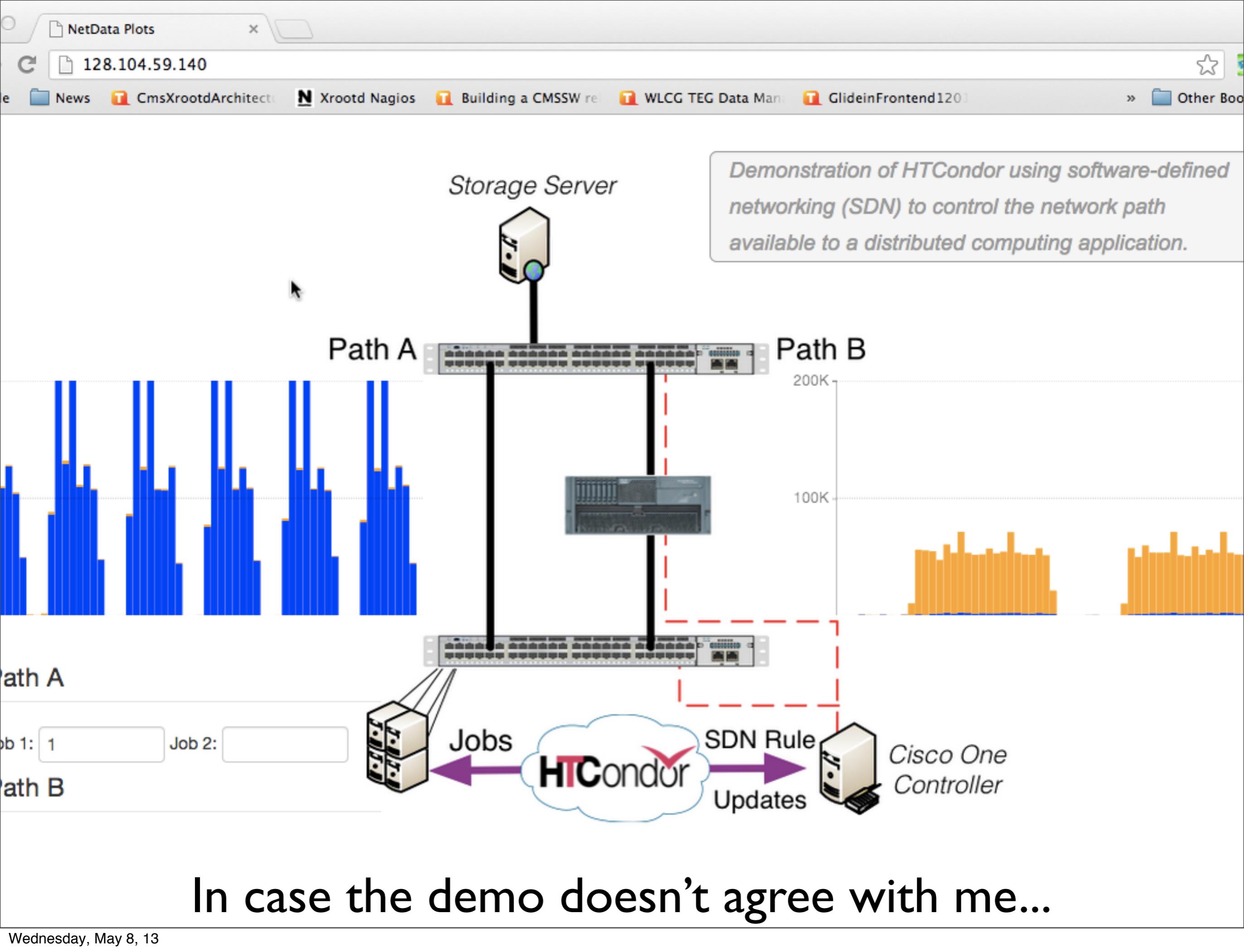
OpenFlow-enabled Network

OpenFlow Switching

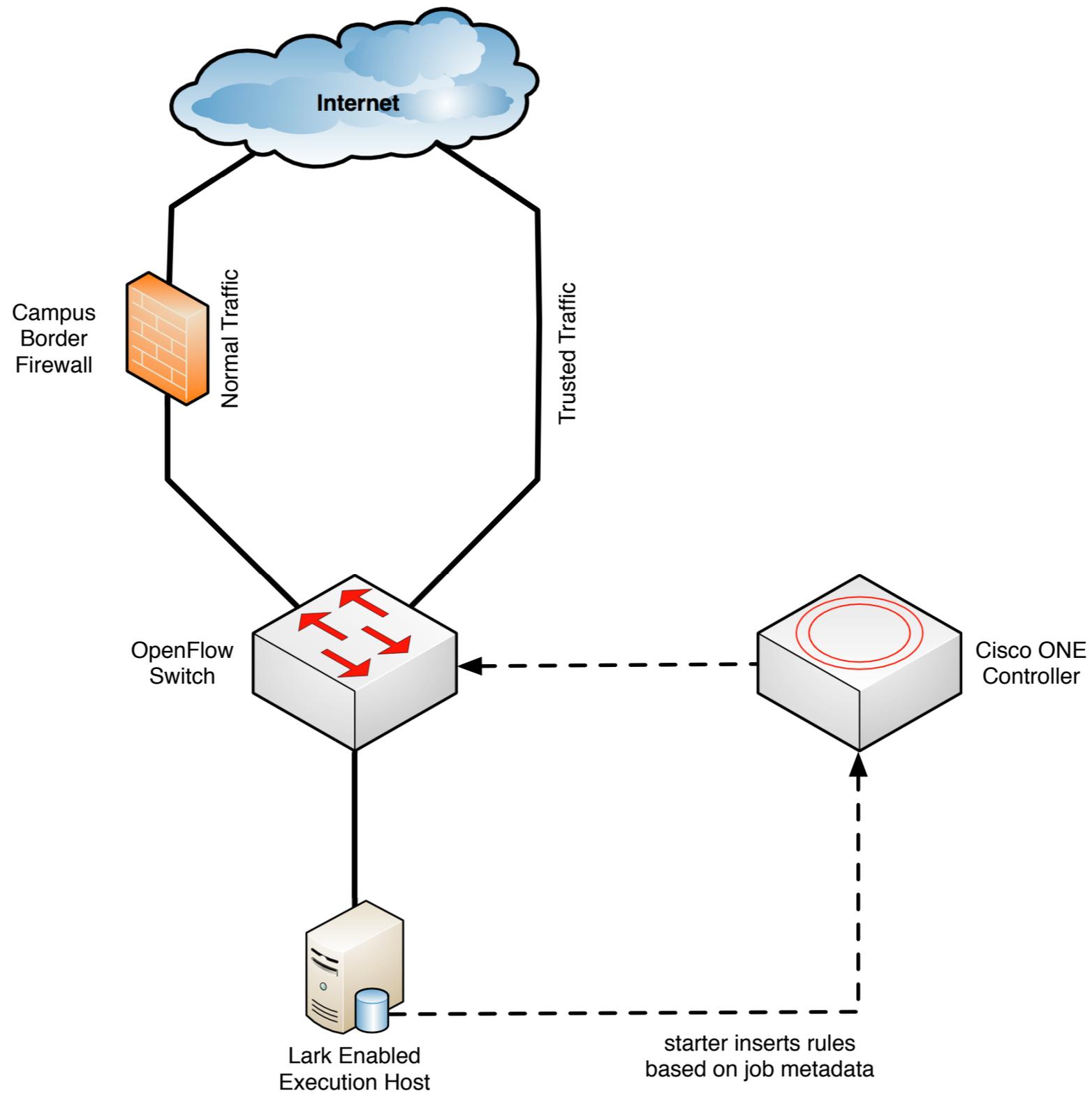


<http://www.openflow.org/documents/OpenFlow.ppt>

**Watch as Brian defies
Murphy's Law and tries
a live demo...**



In case the demo doesn't agree with me...



Presented at Cisco Live 2013 - idea of incorporating the network with the scheduler really resonated there.

Other policies under consideration

- **Prioritization and bandwidth limiting** - having the site border understand the different classes of traffic and prioritize / drop accordingly.
 - I.e., “prioritize CMS production over CMS Xrootd over OSG usage”
- **Network Slicing** - Isolate different jobs on the same network from each other.
- **Network flocking** - joining a job temporarily to a remote network.

Across the campus, Across the world

- On the grid, we are seeing jobs increasingly utilizing the network.
 - This utilization is largely unmanaged; available network connectivity greatly affects the application performance.
 - If we cannot get a guaranteed amount of bandwidth, it may not be reasonable to run the jobs remotely.
- Once we plug the job into the local network, the local network can then make routing decisions about the wider area.
 - I2 has a project, DYNES, that provides wide-area circuit reservations. Idea is to use DYNES to guarantee certain amounts of bandwidth between UW and UNL.
 - “We have the technology,” but we haven’t quite been able to get everything to work. It’ll be a busy summer...

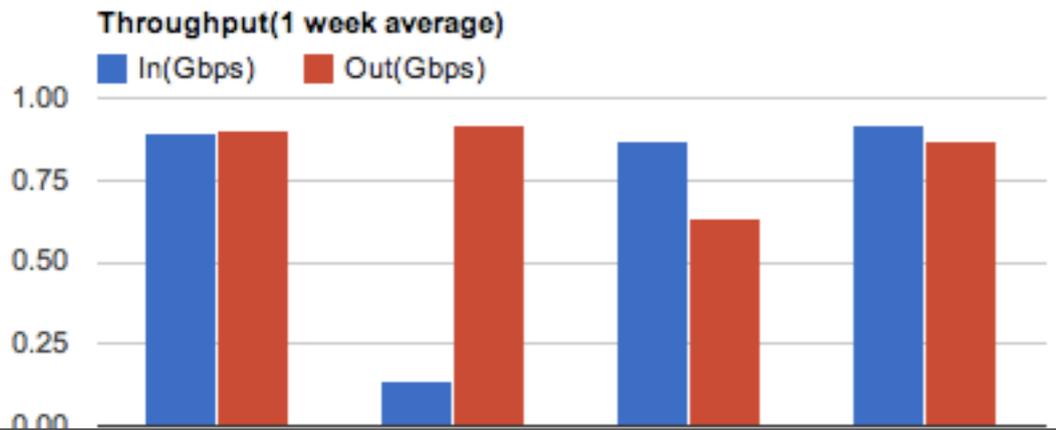
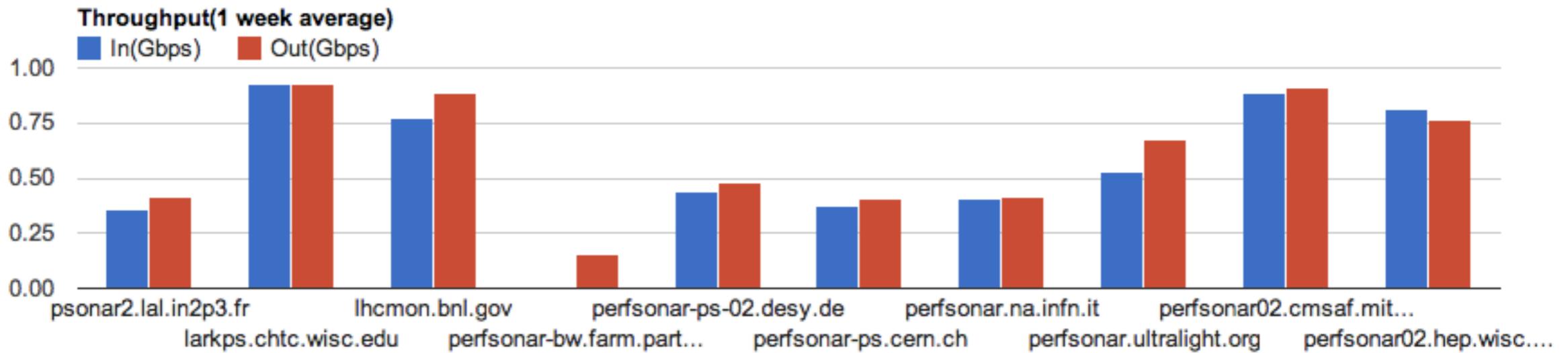
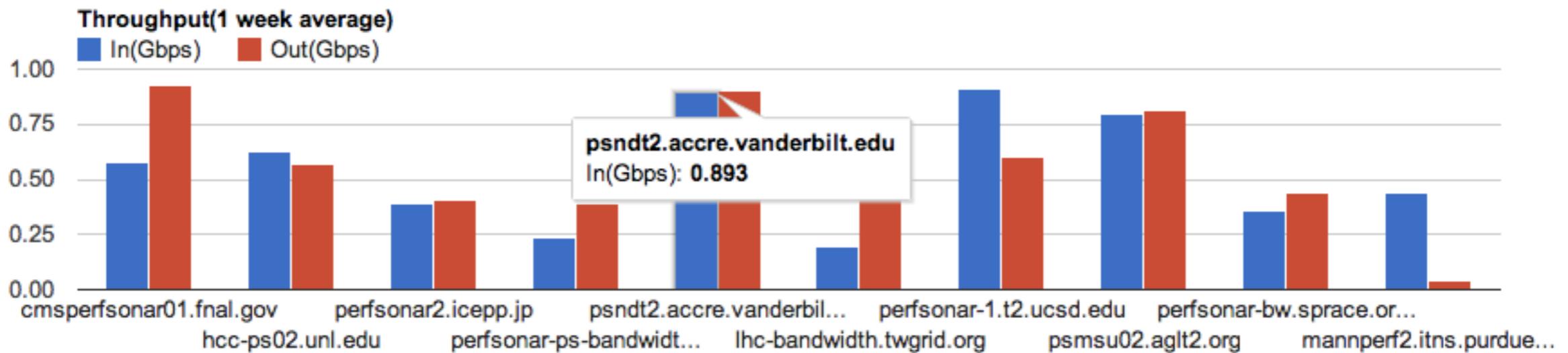
Part V: Expanding the Ecosystem

Thinking outside the worker node

- The worker node is only one place where resources are managed and relevant in HTCondor.
- We'd like to give the HTCondor scheduler information about the network performance.

What's up, perfSONAR?

- PerfSONAR is an appliance designed to measure and record network performance. Example hosts:
 - <http://hcc-ps01.unl.edu/> - packet loss measurements.
 - <http://hcc-ps02.unl.edu/> - throughput measurements.
- Idea is to place perfSONAR measurement hosts in various places of the network.



Gathering Network Data

- perfSONAR has a global set of lookup services for discovering these measurement hosts.
- You can then query each service individually for its performance data.
- Lark does the work of periodically “spidering” the perfSONAR network, then pushes this to the HTCondor collector.
- This provides a way to quickly - and centrally - analyze the point-to-point connectivity of the grid we’re running on.

Using Network Data

- Now that I can tell you the available bandwidth between the submitter (in Nebraska) and the worker node (in FNAL), we need to do something with it.
- Our first approach - summer 2013 - will be to implement circuit breakers.
- If the scheduler can determine there's no available bandwidth to a site, do not match jobs which require more than X GB of input data.

Part VI: Futures

Job Lifetime

- Right now, all our networking policies apply to the job while it's running.
- There's two other pieces of the job lifetime relevant to networking:
 - File stagein,
 - File stageout.
- We're looking at allowing jobs to specify different networking configurations for each part of the lifetime.

OpenFlow

- Right now, we integrate with OpenFlow controllers in a simplistic manner - we setup a new static rule at the job beginning and delete it at the end.
- The network doesn't really have any knowledge about our jobs. Each worker needs to know, for example, the routes needed for the job.
- Next step is to add a HTCondor module to an OpenFlow controller. We'd like to tell the switch what job we're starting by sending it the job description, then having it decide what rules to add.
- First step in having the switch providing intelligence back to the scheduler.

OpenVSwitch

- In Linux, bridges are relatively tricky.
 - The implementation and API are designed to be configured statically.
 - We've experienced a few headaches in bringing devices onto the network for a few seconds, then having them disappear forever.
 - Configuration interface is ioctl's.
- OpenVSwitch is designed from the ground-up for interacting with dynamic virtualized devices and OpenFlow.
 - We could have the OpenFlow controller even manage the jobs on the host level.
- We may swap out Linux bridging with OpenVSwitch in the future.

DYNES / Circuits

- We've had quite a few technical difficulties in automating the circuit setup.
- We aren't giving up on the idea of circuits.
- However, we are looking harder at tunneling / overlaying our own network on the existing one.
 - More traditional network approach - "smart ends, dumb middle" and does not violate layering.
 - No longer requires
- Will allow us to continue to increase the reach of our network integration by integrating the job into the overlay network.
 - We can circle back later and do bandwidth reservations.

perfSONAR

- There are several unsolved problems in effectively using the perfSONAR data:
 - It's difficult to discern data quality.
 - Mapping network locations to grid sites / worker nodes is currently done by hand.
 - We have several levels of scheduling in our grid system - should HTCondor or glideinWMS take networking into account?

Concluding...

- A resource manager can no longer ignore network availability and topology, either on the campus or running globally.
- With Lark's worker node integration, HTCondor can manipulate the host and local network at the per-job level.
- This allows us to expose application-level details to the network that were never previously available.
- Lark is working to aggregate network performance data into the HTCondor ecosystem and exploring ways to use it for scheduling.
- These are all the first steps into bringing our distributed high throughput computing ecosystem into the network.

<https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki?>

[p=LarkProject](#)