Traffic visibility and control with sFlow

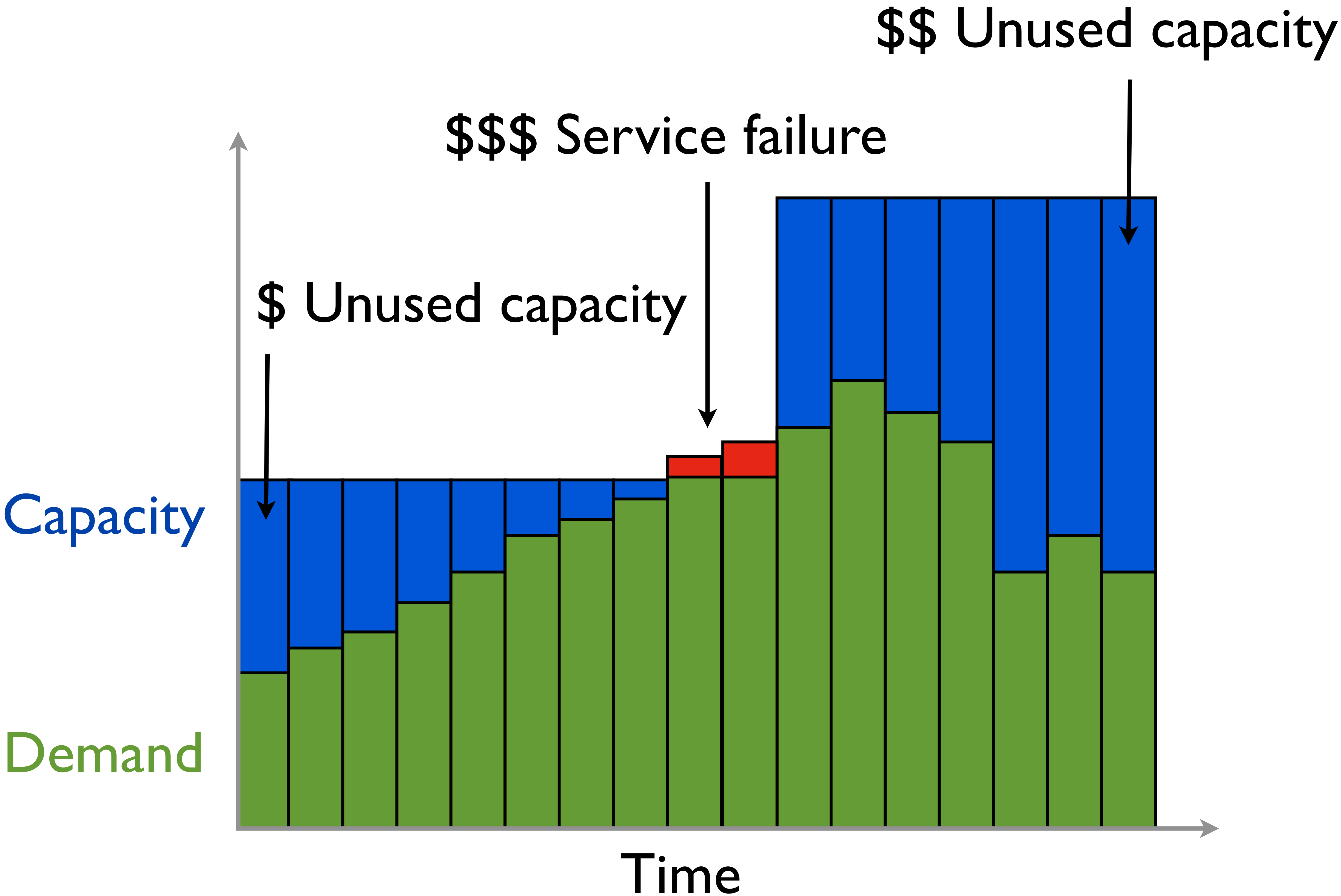
Open vSwitch 2014 Fall Conference

Peter Phaal
InMon Corp.
November 2014

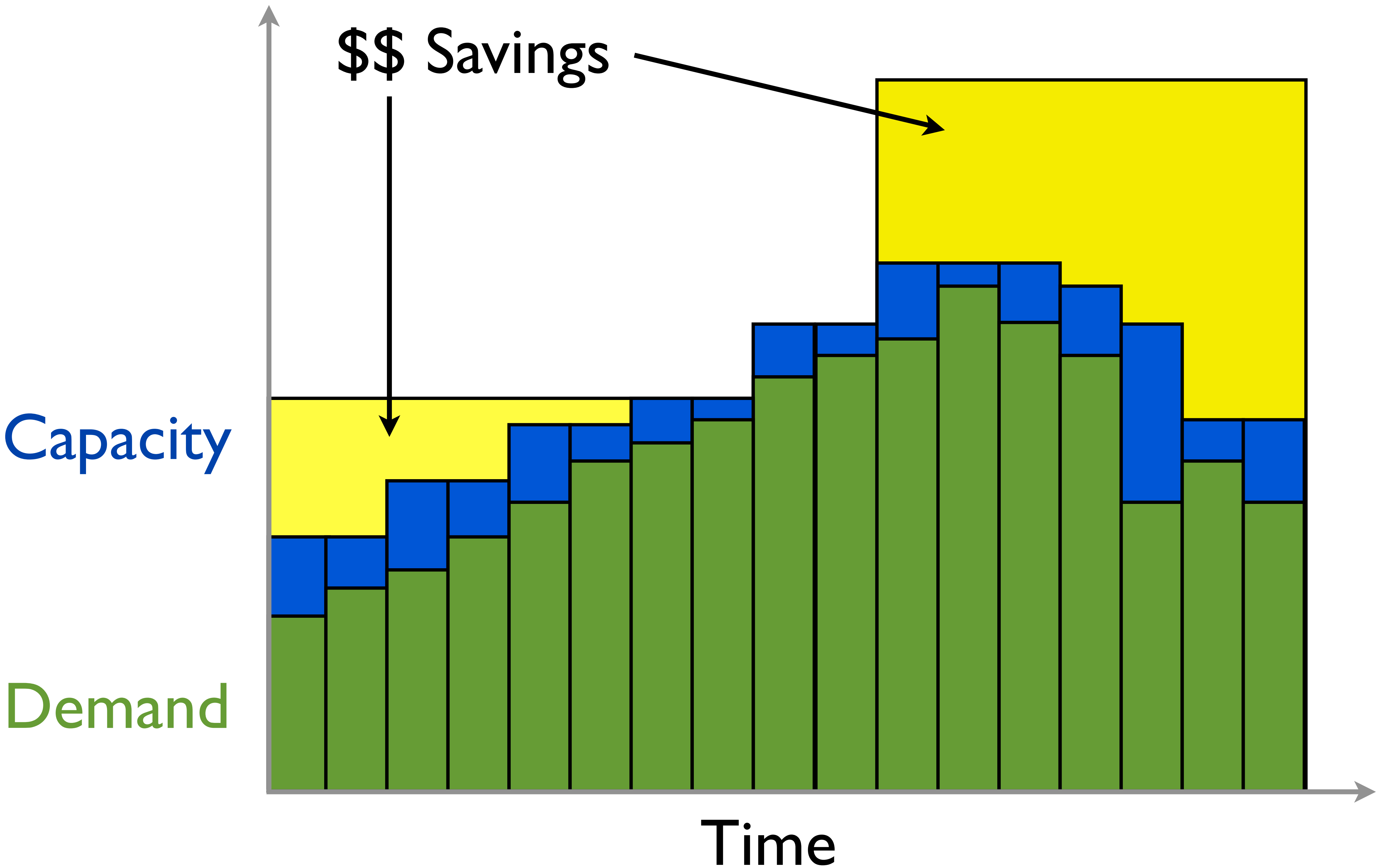
Why monitor performance?

"If you can't measure it, you can't improve it"
Lord Kelvin

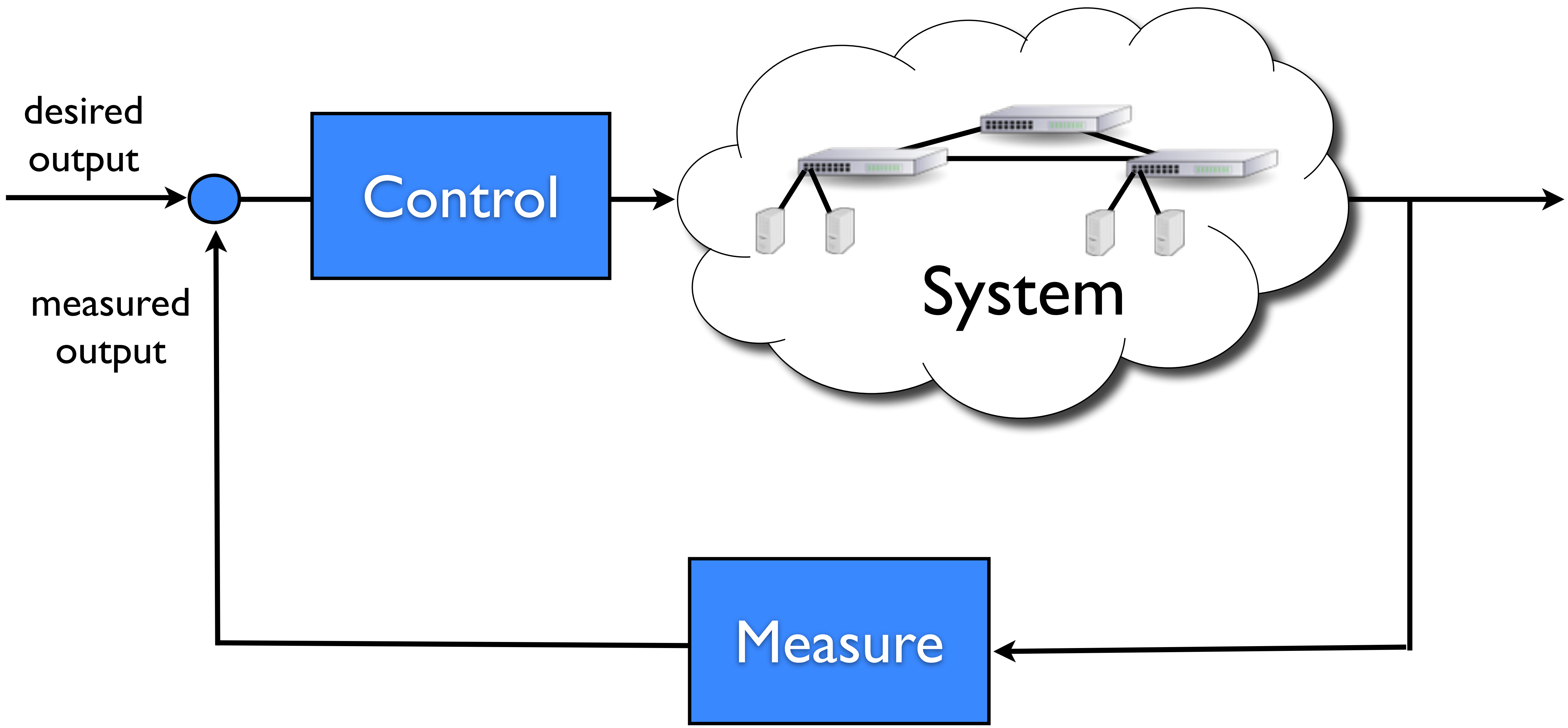
Static provisioning



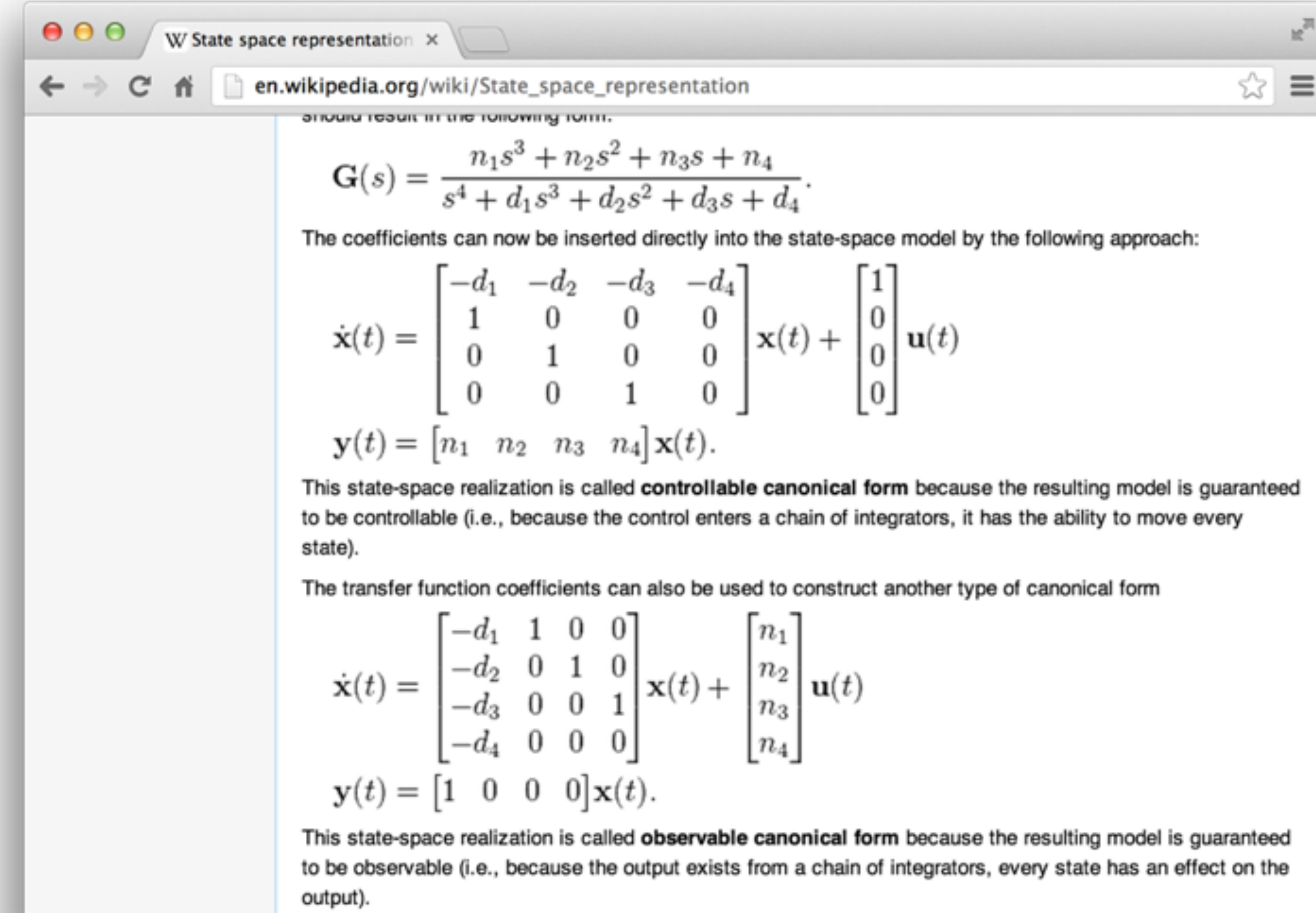
Dynamic provisioning



Feedback control



Controllability and Observability



should result in the following form:

$$\mathbf{G}(s) = \frac{n_1 s^3 + n_2 s^2 + n_3 s + n_4}{s^4 + d_1 s^3 + d_2 s^2 + d_3 s + d_4}.$$

The coefficients can now be inserted directly into the state-space model by the following approach:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} -d_1 & -d_2 & -d_3 & -d_4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \mathbf{u}(t)$$

$$\mathbf{y}(t) = \begin{bmatrix} n_1 & n_2 & n_3 & n_4 \end{bmatrix} \mathbf{x}(t).$$

This state-space realization is called **controllable canonical form** because the resulting model is guaranteed to be controllable (i.e., because the control enters a chain of integrators, it has the ability to move every state).

The transfer function coefficients can also be used to construct another type of canonical form

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} -d_1 & 1 & 0 & 0 \\ -d_2 & 0 & 1 & 0 \\ -d_3 & 0 & 0 & 1 \\ -d_4 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{bmatrix} \mathbf{u}(t)$$

$$\mathbf{y}(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}(t).$$

This state-space realization is called **observable canonical form** because the resulting model is guaranteed to be observable (i.e., because the output exists from a chain of integrators, every state has an effect on the output).

Basic concept is simple, a stable feedback control system requires:

1. ability to influence <u>all</u> important system states (controllable)
2. ability to monitor <u>all</u> important system states (observable)

Controllability and Observability driving example

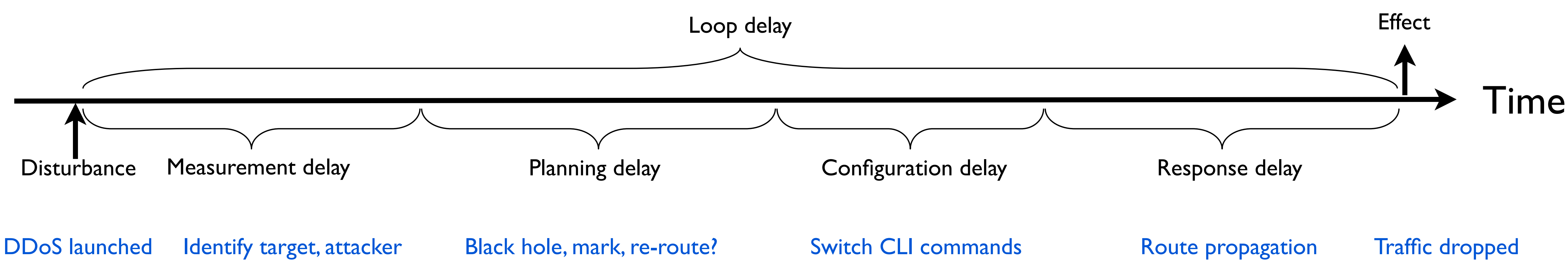
States — location, speed, direction, ...

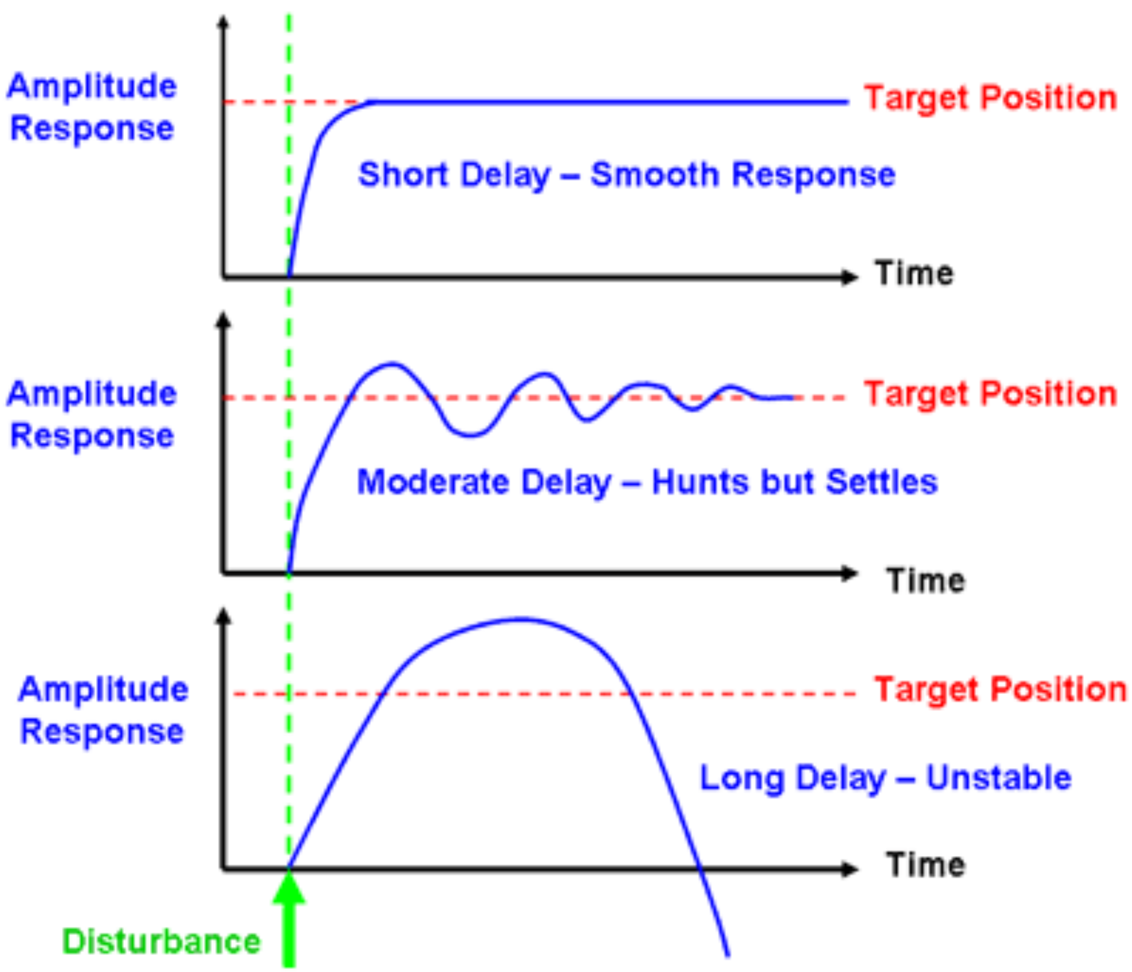
Observability — It's hard to stay on the road if you can't see the road, or keep to the speed limit without a speedometer

Controllability — It's hard to stay on the road or maintain speed if your brakes, engine or steering fail

Effect of delay on stability

Components of loop delay





e.g. Slow reaction time causes tired / drunk / distracted driver to weave, very slow reaction time and they leave the road

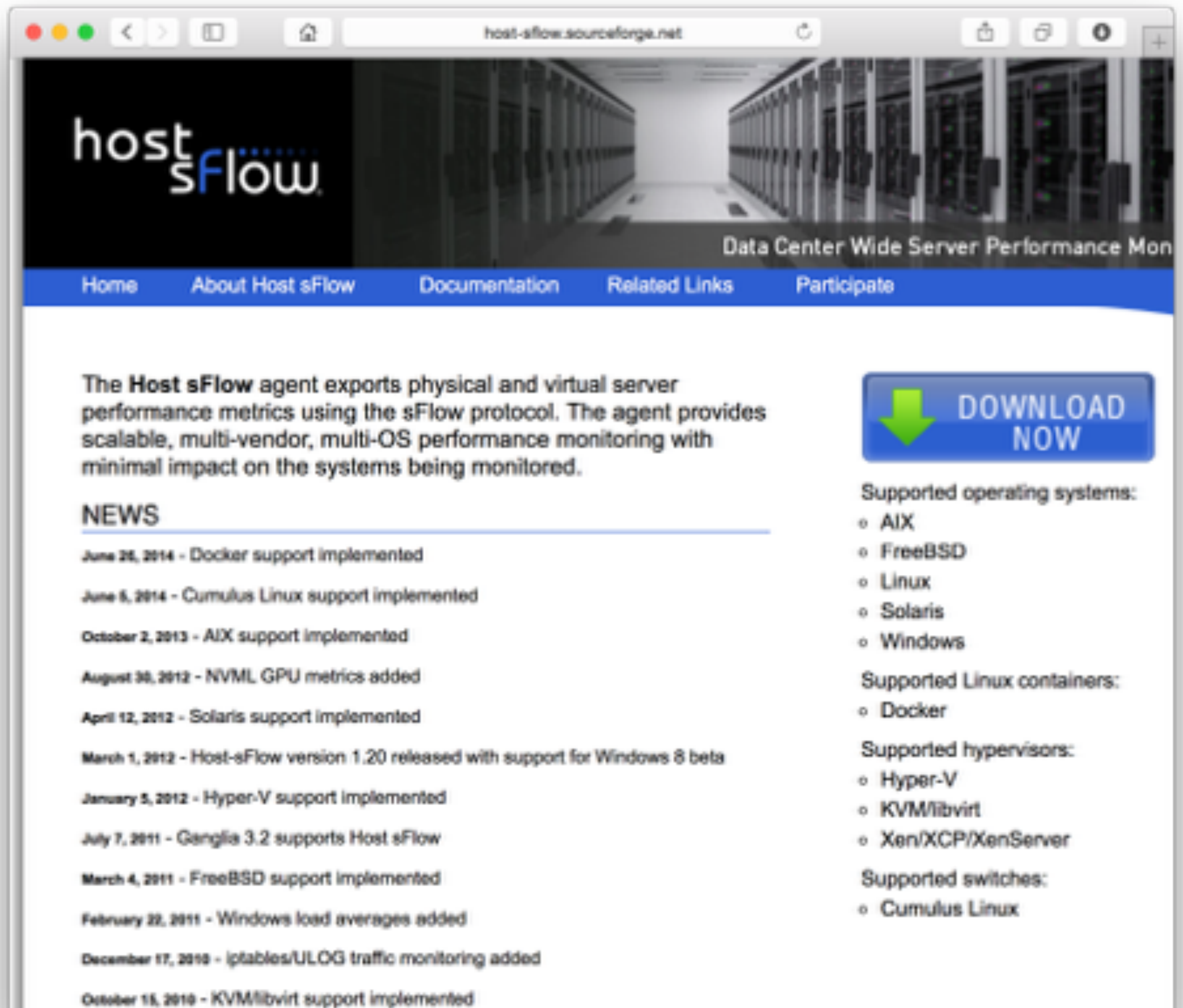
What is sFlow?

"In God we trust. All others bring data."
Dr. Edwards Deming

Industry standard measurement technology integrated in switches

http://www.sflow.org/

Open source agents for hosts, hypervisors and applications



Host sFlow project (http://host-sflow.sourceforge.net) is center of an ecosystem of related open source projects embedding sFlow in popular operating systems and applications

Standard counters

Network (maintained by hardware in network devices)

- **MIB-2 ifTable:** ifInOctets, ifInUcastPkts, ifInMulticastPkts, ifInBroadcastPkts, ifInDiscards, ifInErrors, ifUnkownProtos, ifOutOctets, ifOutUcastPkts, ifOutMulticastPkts, ifOutBroadcastPkts, ifOutDiscards, ifOutErrors

Host (maintained by operating system kernel)

- **CPU:** load_one, load_five, load_fifteen, proc_run, proc_total, cpu_num, cpu_speed, uptime, cpu_user, cpu_nice, cpu_system, cpu_idle, cpu_wio, cpu_intr, cpu_sintr, interupts, contexts
- **Memory:** mem_total, mem_free, mem_shared, mem_buffers, mem_cached, swap_total, swap_free, page_in, page_out, swap_in, swap_out
- **Disk IO:** disk_total, disk_free, part_max_used, reads, bytes_read, read_time, writes, bytes_written, write_time
- **Network IO:** bytes_in, packets_in, errs_in, drops_in, bytes_out, packet_out, errs_out, drops_out

Application (maintained by application)

- **HTTP:** method_option_count, method_get_count, method_head_count, method_post_count, method_put_count, method_delete_count, method_trace_count, method_connect_count, method_other_count, status_1xx_count, status_2xx_count, status_3xx_count, status_4xx_count, status_5xx_count, status_other_count
- **Memcache:** cmd_set, cmd_touch, cmd_flush, get_hits, get_misses, delete_hits, delete_misses, incr_hits, incr_misses, decr_hists, decr_misses, cas_hits, cas_misses, cas_badval, auth_cmds, auth_errors, threads, con_yields, listen_disabled_num, curr_connections, rejected_connections, total_connections, connection_structures, evictions, reclaimed, curr_items, total_items, bytes_read, bytes_written, bytes, limit_maxbytes

Scaleable push protocol

Simple

- standard structures - densely packed blocks of counters
- extensible (tag, length, value)
- RFC 1832: XDR encoded (big endian, quad-aligned, binary) - simple to encode/decode
- unicast UDP transport

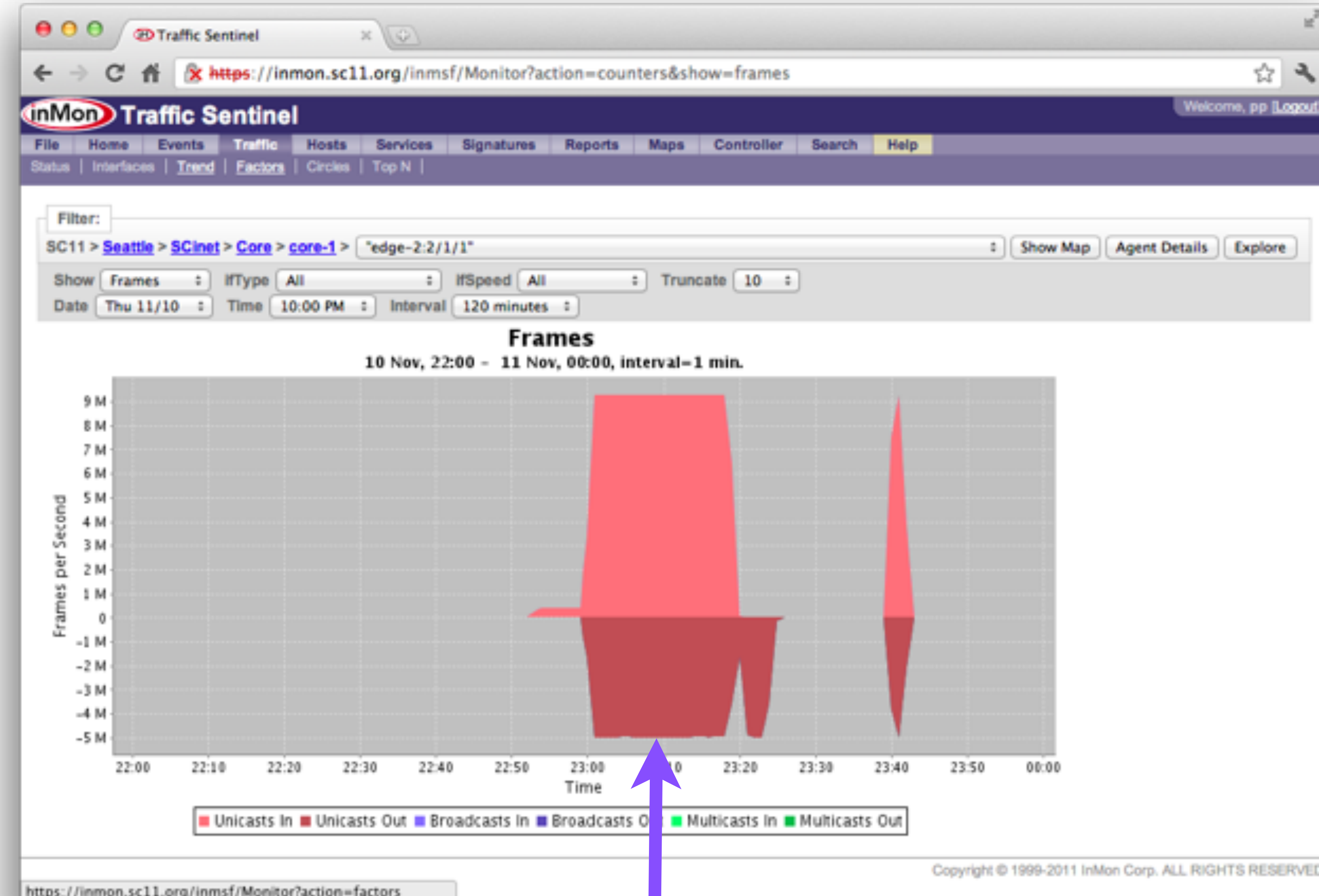
Minimal configuration

- collector address
- polling interval

Cloud friendly

- flat, two tier architecture: many embedded agents → central "smart" collector
- sFlow agents automatically start sending metrics on startup, automatically discovered
- eliminates complexity of maintaining polling daemons (and associated configurations)

Counters aren't enough

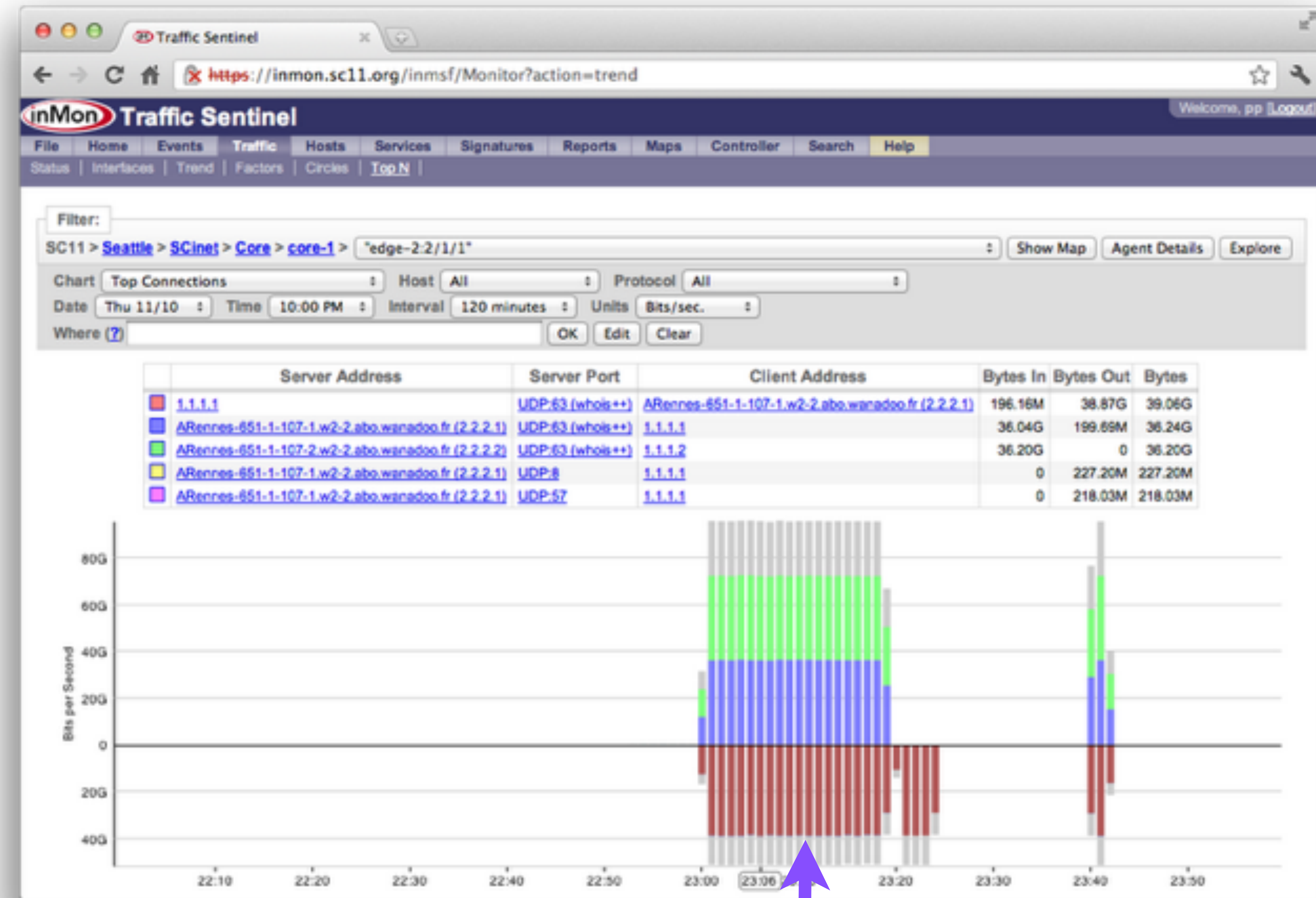


Why the spike in traffic?

(100Gbit link carrying 14,000,000 packets/second)

- Counters tell you there is a problem, but not why.
- Counters summarize performance by dropping high cardinality attributes:
  - IP addresses
  - URLs
  - Memcache keys
- Need to be able to efficiently disaggregate counter by attributes in order to understand root cause of performance problems.
- How do you get this data when there are millions of transactions per second?

sFlow also exports random samples

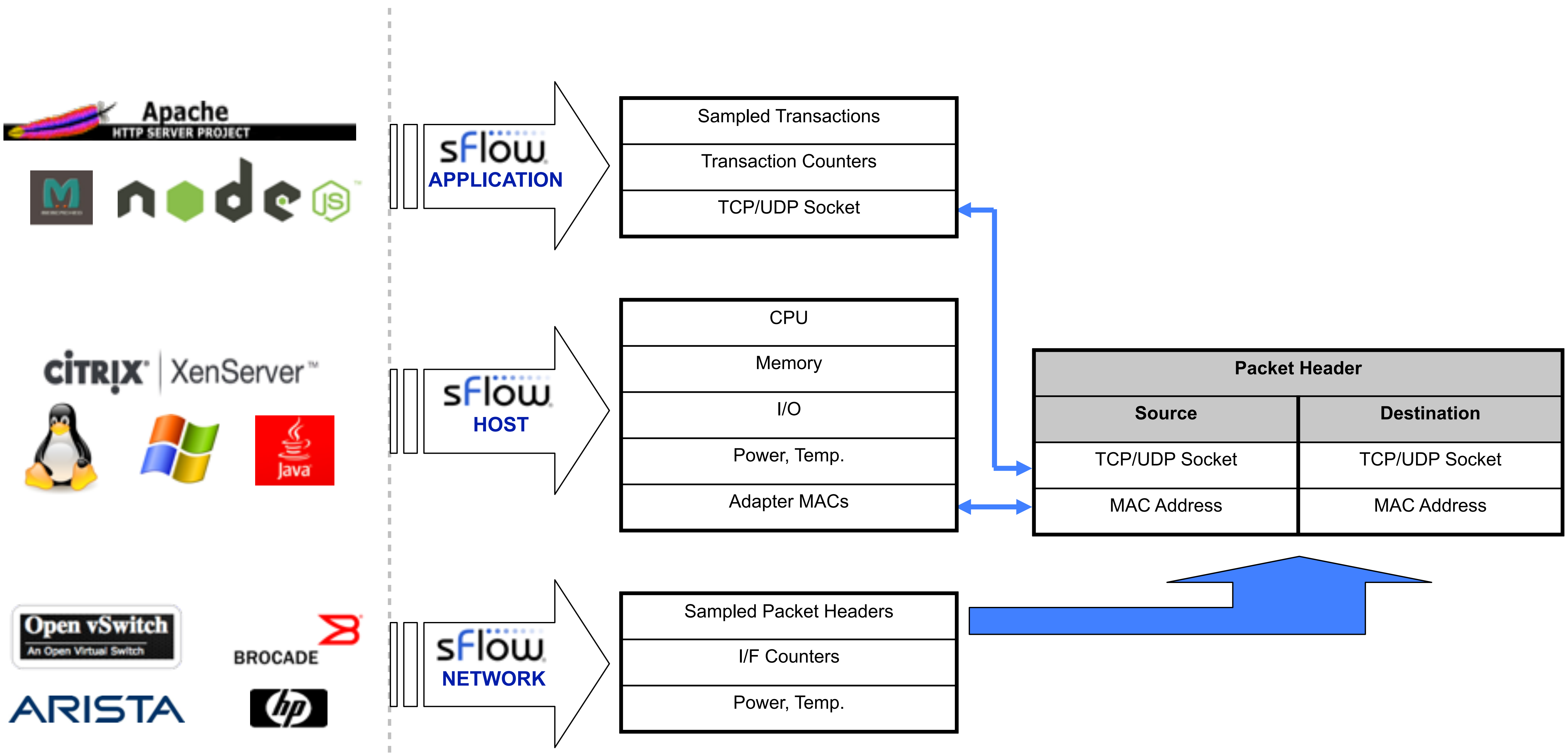


- Random sampling is lightweight
- Critical path roughly cost of maintaining one counter:
  `if(--skip == 0) sample();`
- Sampling is easy to distribute among modules, threads, processes without any synchronization
- Minimal resources required to capture attributes of sampled transactions
- Easily identify top keys, connections, clients, servers, URLs etc.
- Unbiased results with known accuracy

Break out traffic by client, server and port

(graph based on samples from100Gbit link carrying 14,000,000 packets/second)

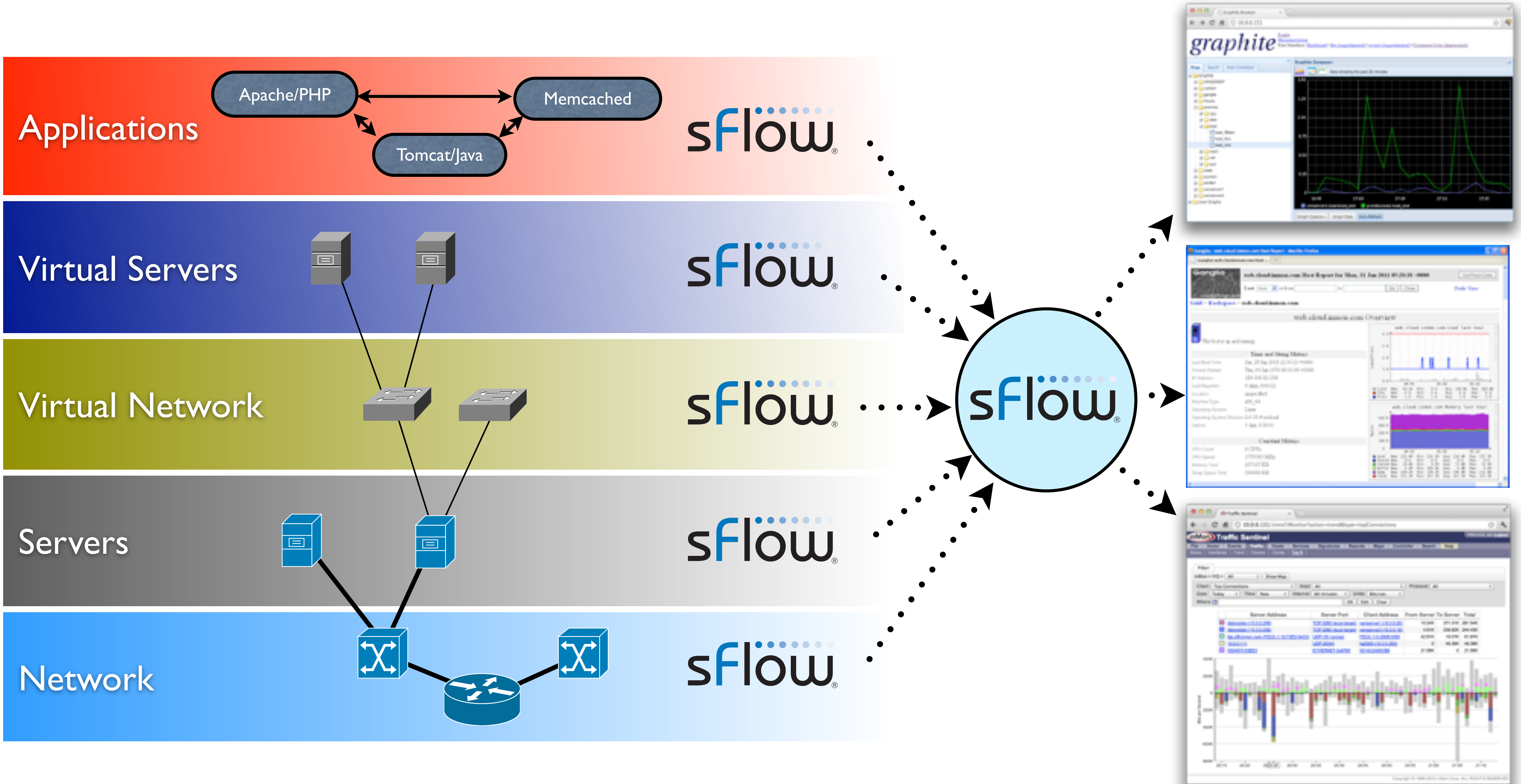
Integrated data model



Independent agents

sFlow analyzer joins data for integrated view

Comprehensive data center wide visibility



Embedded monitoring of all switches, all servers, all applications, all the time

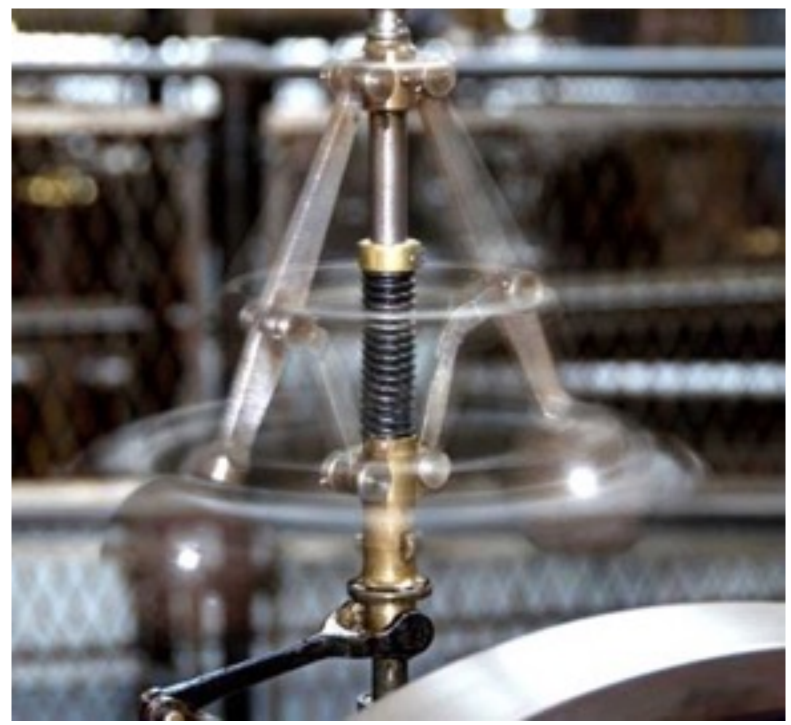
Consistent measurements shared between multiple management tools

sFlow, OVS and Host sFlow timeline

Date	sFlow spec.	OVS	Host sFlow
Jan 2010	sFlow Version 5 (published July 2004)	sampled_header, extended_switch, if_counters	
May 2010			project started
July 2010	sFlow Host Structures (published July 2010)		host_descr, host_adapters, host_cpu, host_memory, host_disk_io, host_net_io
Sept 2010			OVS integration
Oct 2010	sFlow Host Structures (published July 2010)		XenServer, KVM integration host_parent, virt_node, virt_cpu, virt_memory, virt_disk_io, virt_net_io
Jan 2012			Hyper-V integration
June 2014	sFlow OpenFlow Structures (published Nov 2014)		Docker integration, Cumulus Linux integration port_name
Nov 2014	sFlow LAG Counter structures (published Sept 2012) sFlow OpenFlow Structures (published Nov 2014)	lag_port_stats, of_port, port_name	

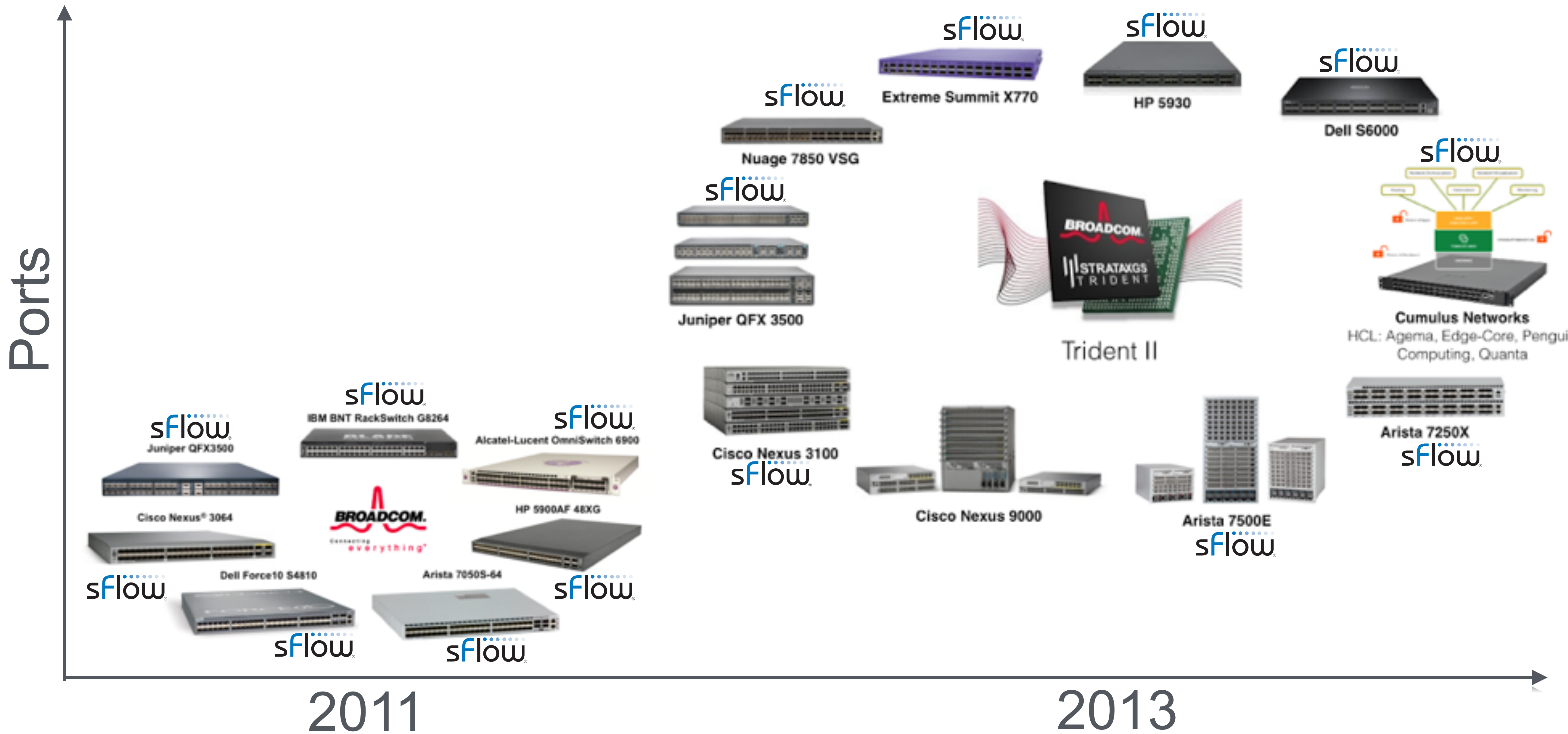
Likely future OVS sFlow additions: sFlow Tunnel Structures (published November 2012), extended_mpls from sFlow Version 5 (published July 2004)

Software Defined Networking



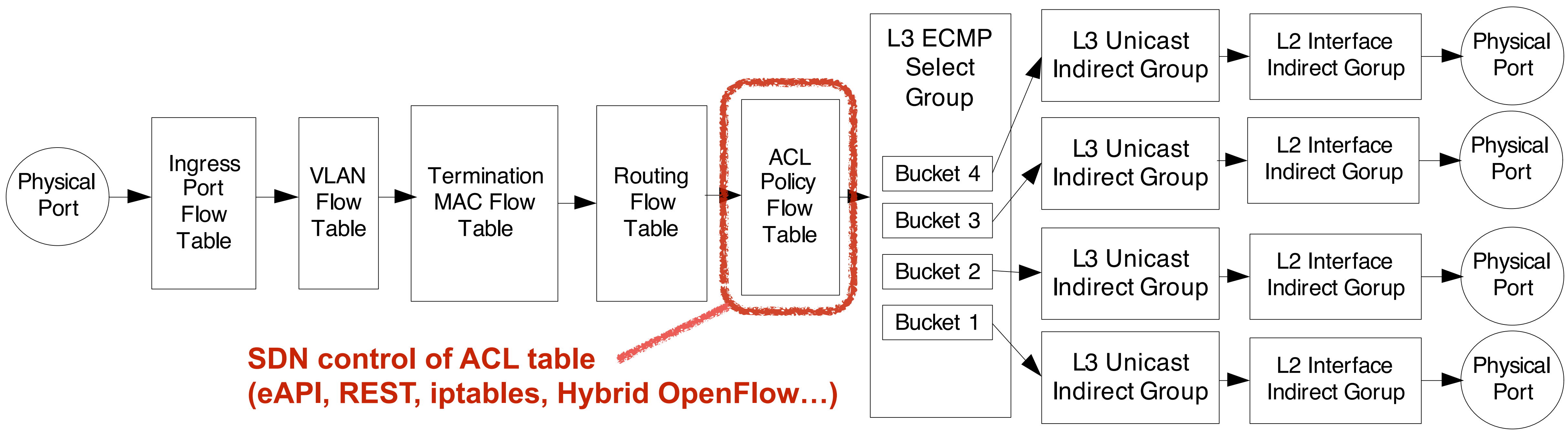
"You can't control what you can't measure"
Tom DeMarco

Measurement standard supported by merchant silicon



Commodity, bare metal, white box, merchant silicon based hardware delivers standard platform (instrumentation / forwarding pipeline) → simplifies creation of SDN solutions

Traffic control with hybrid SDN



https://www.broadcom.com/collateral/wp/OF-DPA-WP102-R.pdf

- **Simple**, no change to normal forwarding behavior - BGP, OSPF, SPB, TRILL, LAG/MLAG etc. used to control L2 / L3 forwarding tables

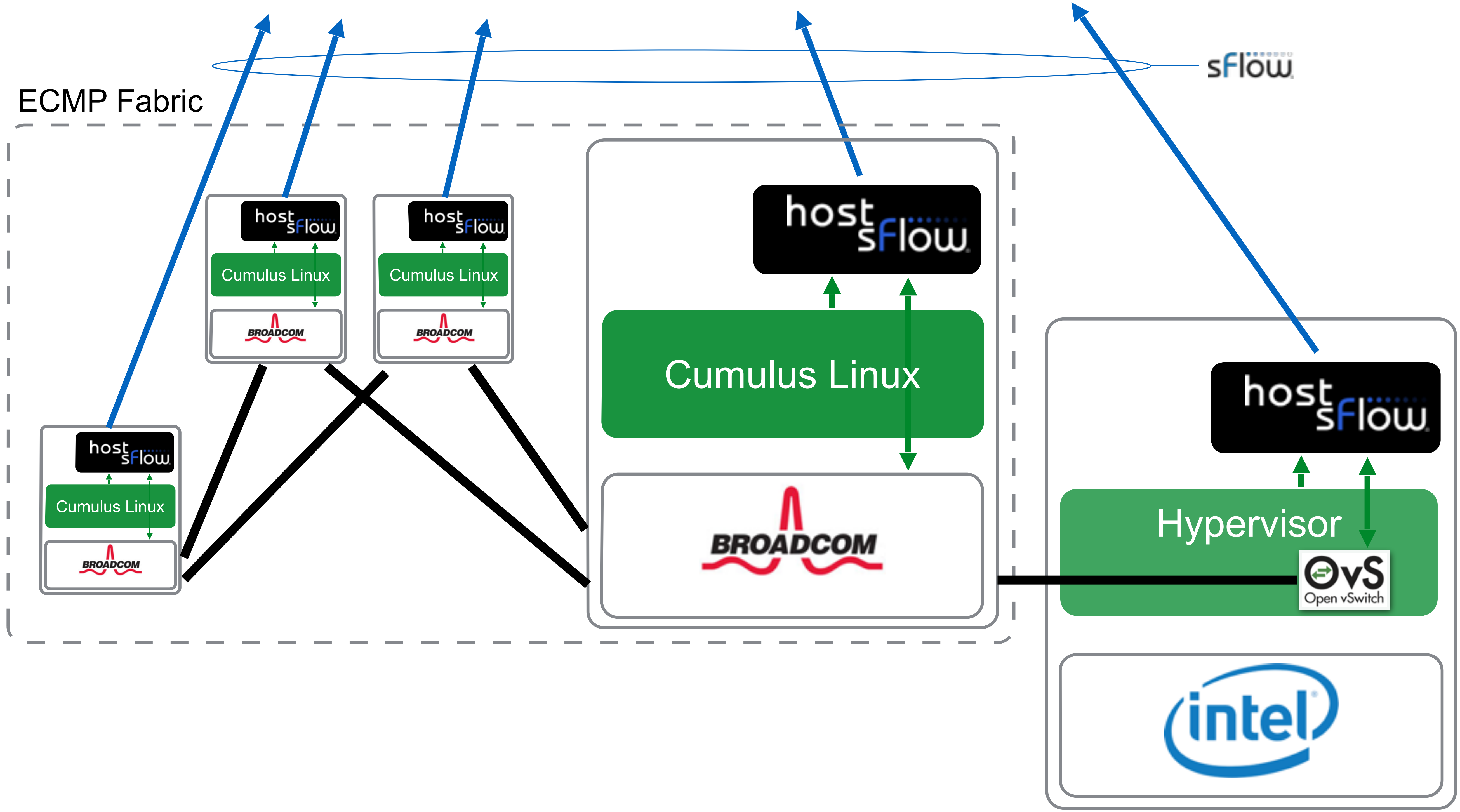
- **Efficient**, hardware multipath forwarding efficiently handles most flows. SDN used to control ACL table and selectively override forwarding of specific flows (block, mark, steer, rate-limit), maximizing effectiveness of limited general match capacity.
  Note: very few ACLs needed in fabric since policy has shifted to edge - mainly required to protect control plane

- **Scaleable**, flows handled by existing control plane, SDN only used when controller wants to make an exception.

- **Robust**, if controller fails, network keeps forwarding

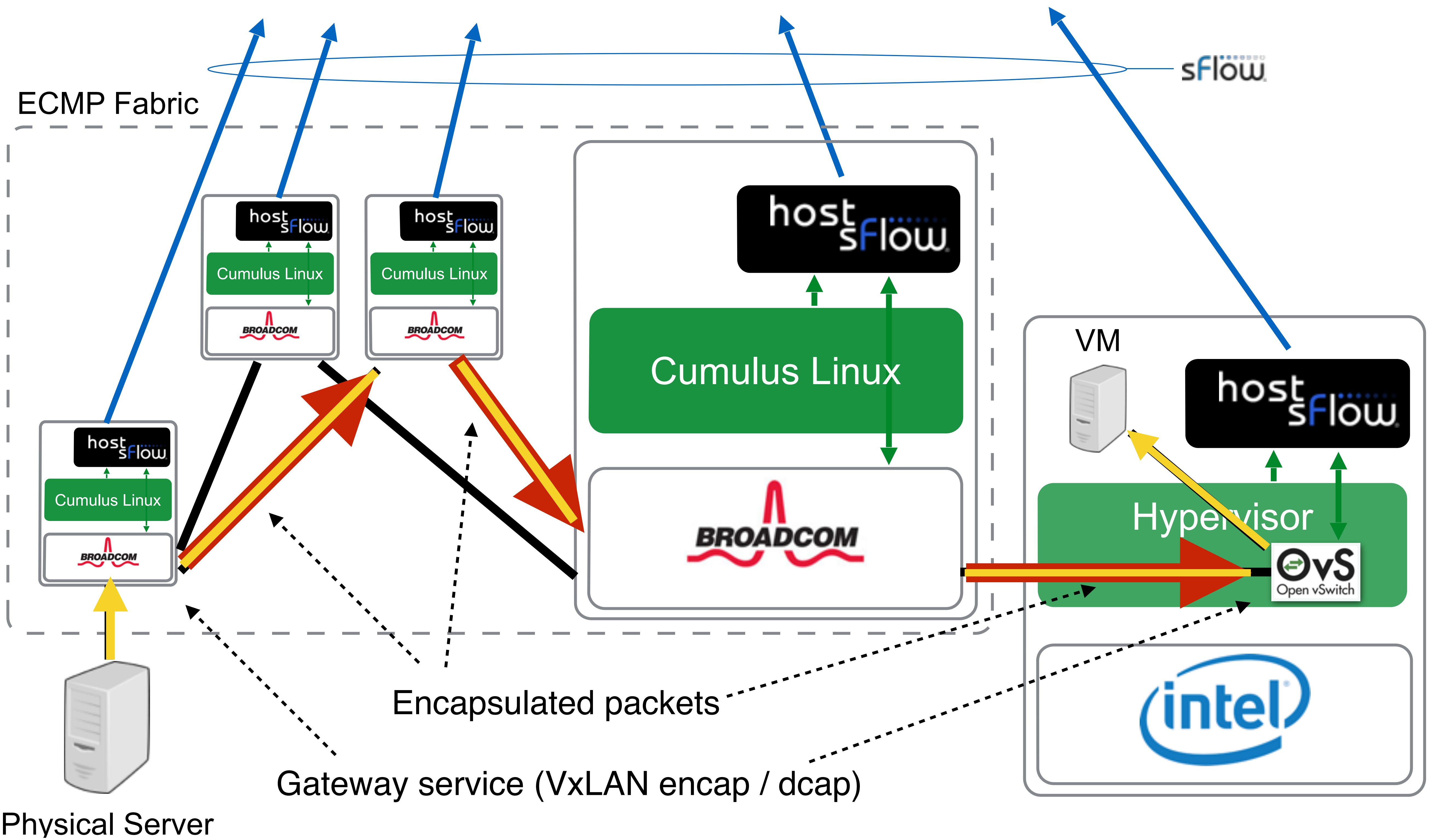
Converging switch / server platforms

sFlow unifies monitoring of host OS, virtual machines / containers, on switches and servers, physical and virtual switches to provide end-to-end visibility through a common set of tools



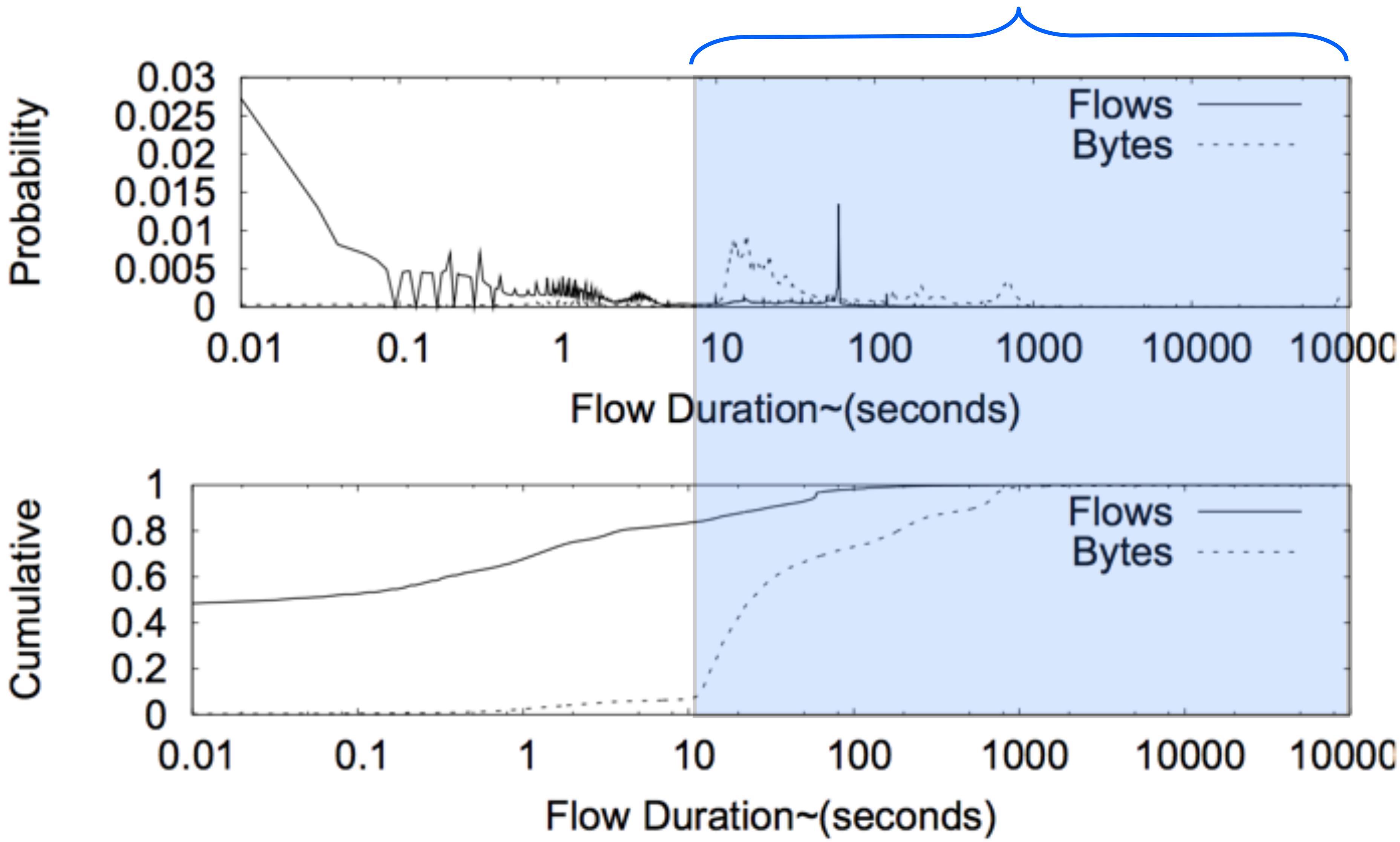
Network virtualization

Real-time visibility into physical and virtual network edge and path across fabric (overlay / underlay addresses, protocols etc. visible in sampled packet headers)



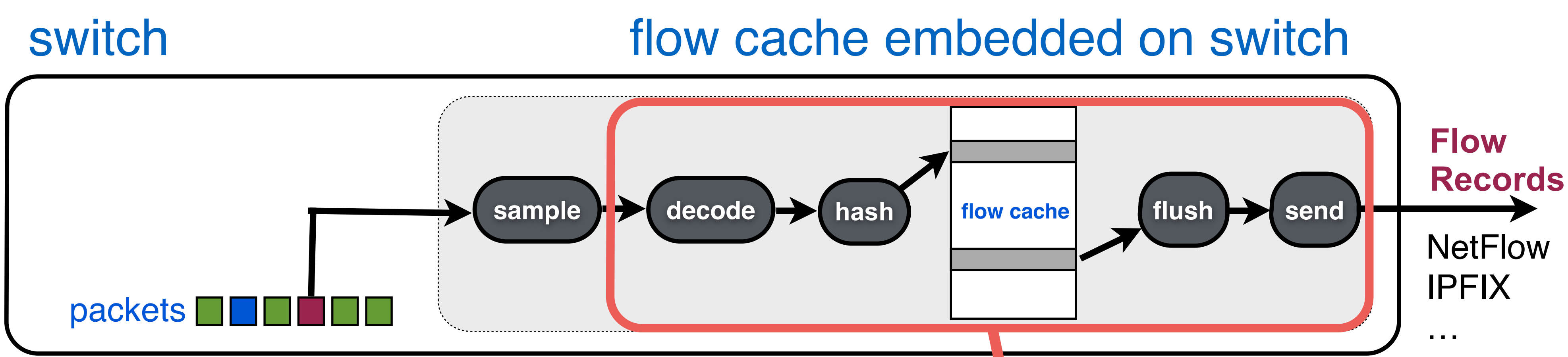
Large "Elephant" flows

Elephant flows are the small number of long lived large flows responsible for majority of bytes on network



http://research.microsoft.com/en-us/UM/people/srikanth/data/imc09_dcTraffic.pdf

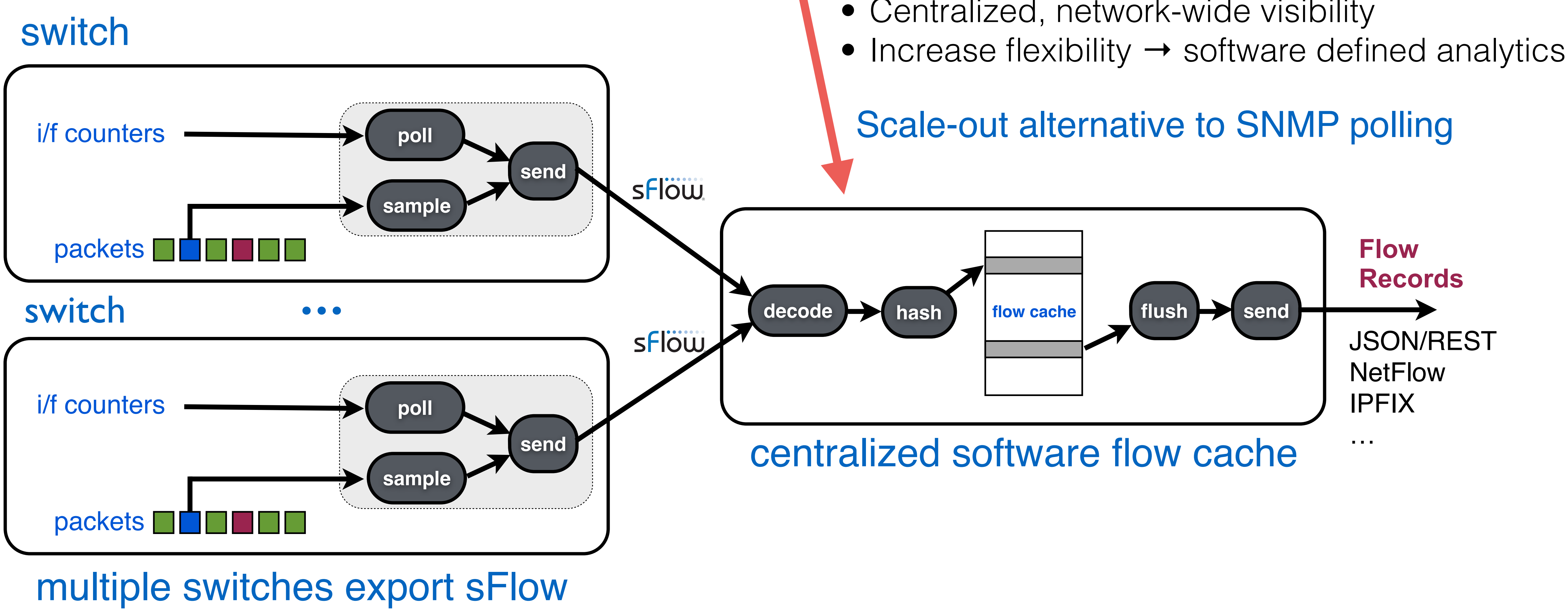
Traffic analytics with sFlow



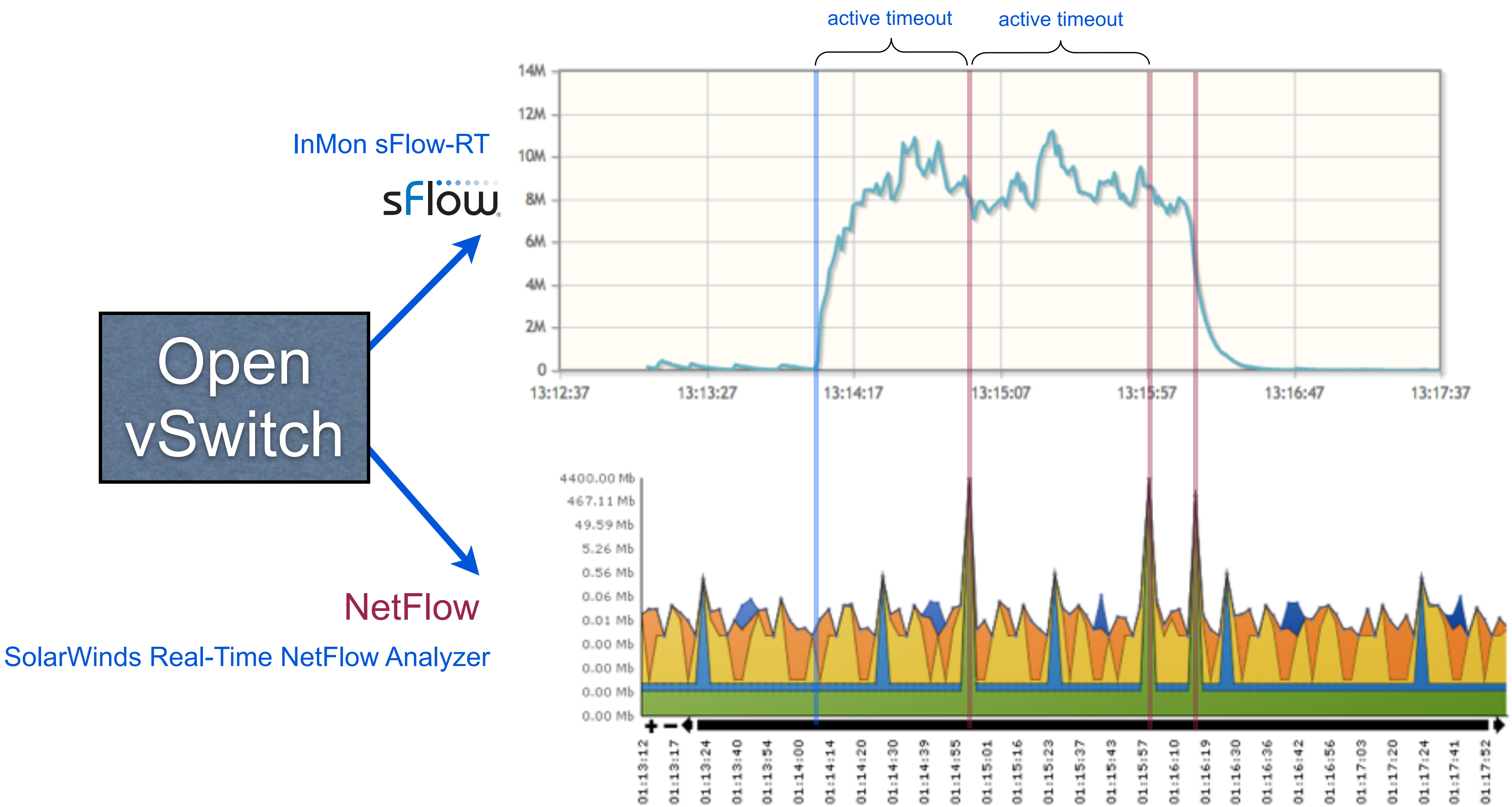
Move flow cache from ASIC to external software

- Reduce ASIC cost / complexity
- Fast response (data not sitting on switch)
- Centralized, network-wide visibility
- Increase flexibility → software defined analytics

Scale-out alternative to SNMP polling



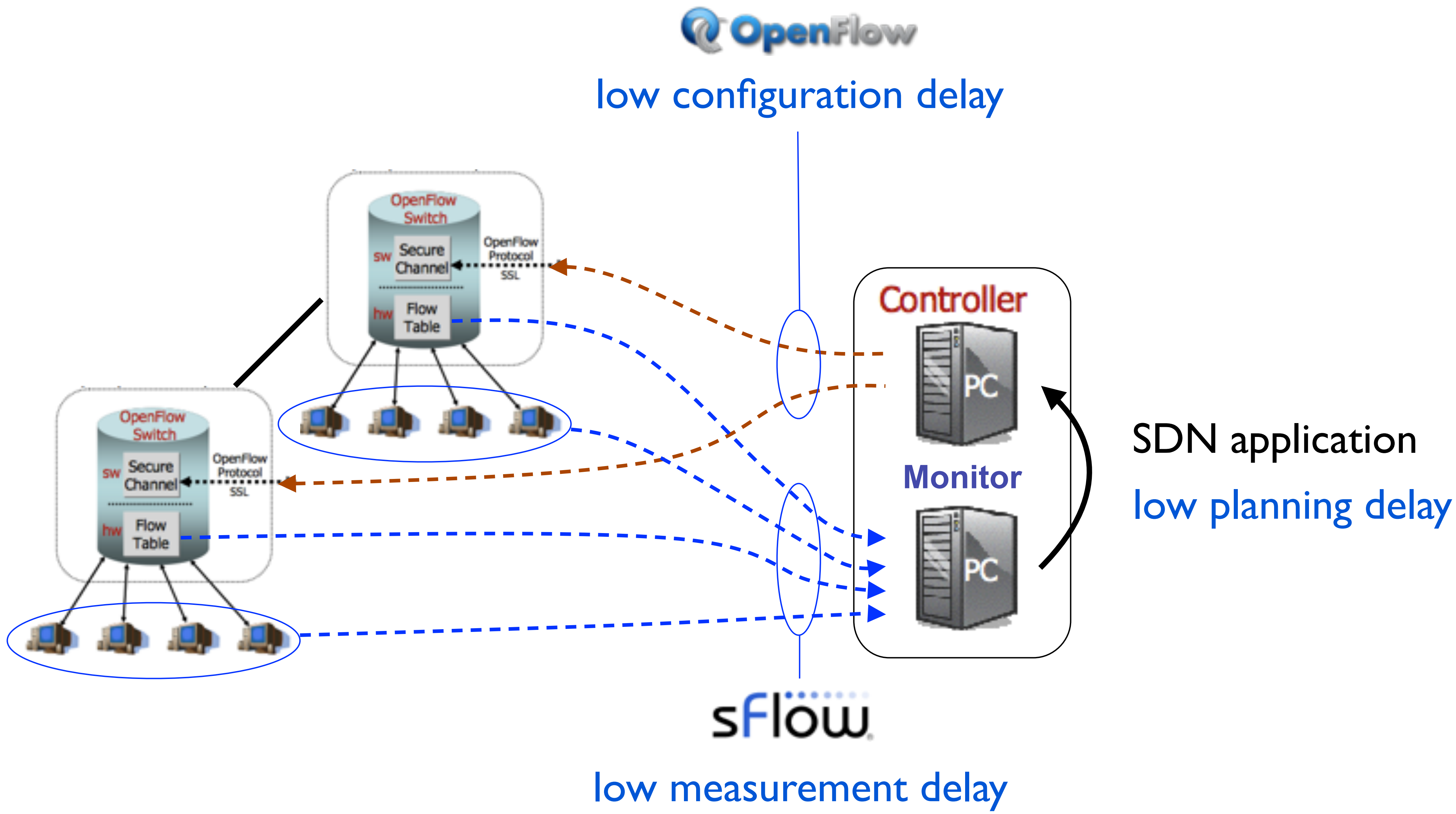
Rapid detection of large flows



- sFlow does not use flow cache, so realtime charts more accurately reflect traffic trend
- NetFlow spikes caused by flow cache active-timeout for long running connections

Flow cache active timeout delays large flow detection, limits value of signal for real-time control applications

Feedback control loop with sFlow and OpenFlow



Together, sFlow and OpenFlow provide the observability and controllability to enable SDN applications targeting low latency control problems like Elephant flow traffic engineering

Connect switches to central control plane

e.g. connect Open vSwitch to OpenFlow controller

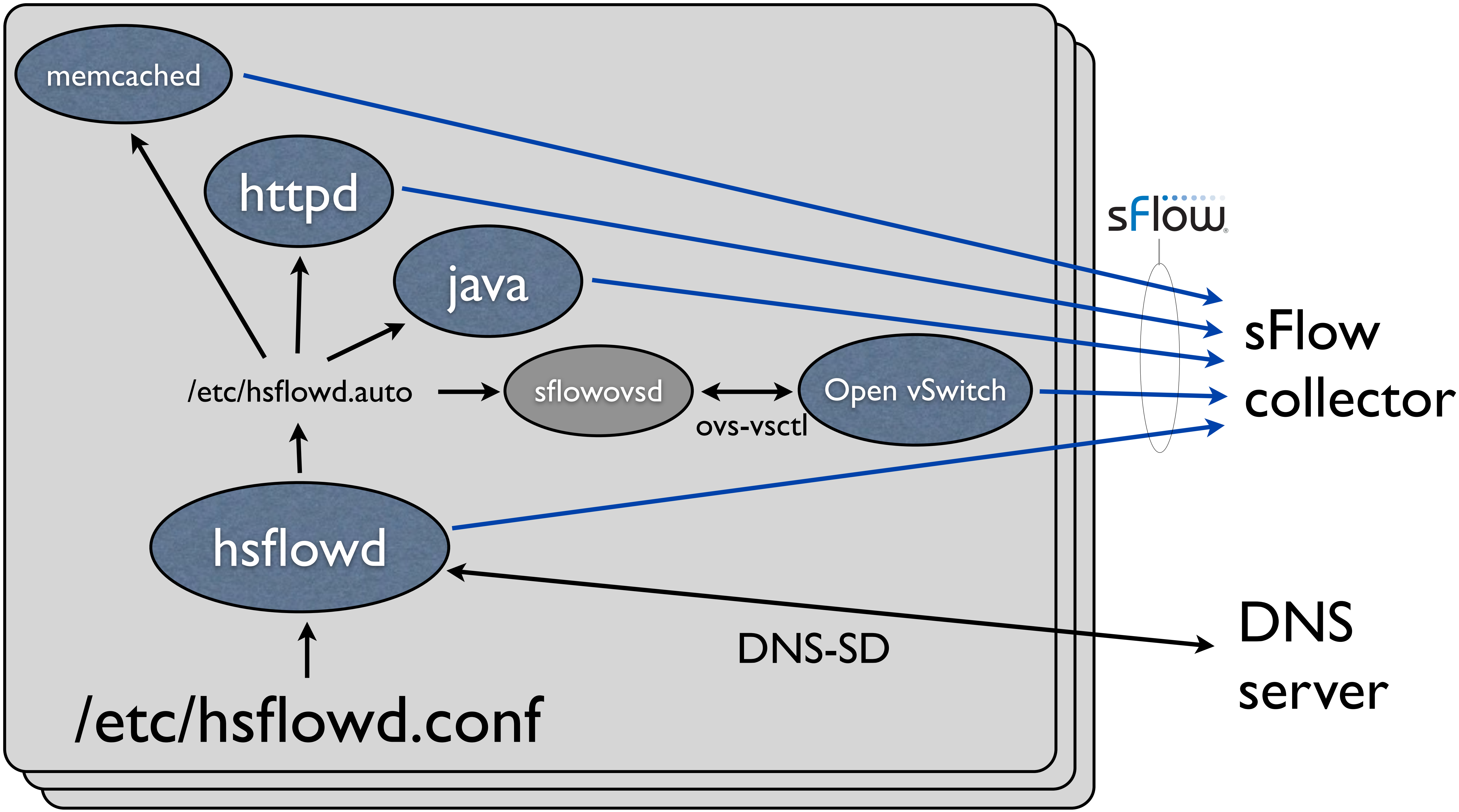
```
ovs-vsctl set-controller br0 tcp:10.0.0.1:6633
```

e.g. connect Open vSwitch to sFlow analyzer

```
ovs-vsctl – –id=@sflow create sflow agent=eth0 \
target=\"10.0.0.1:6343\" sampling=1000 polling=20 \
– set bridge br0 sflow=@sflow
```

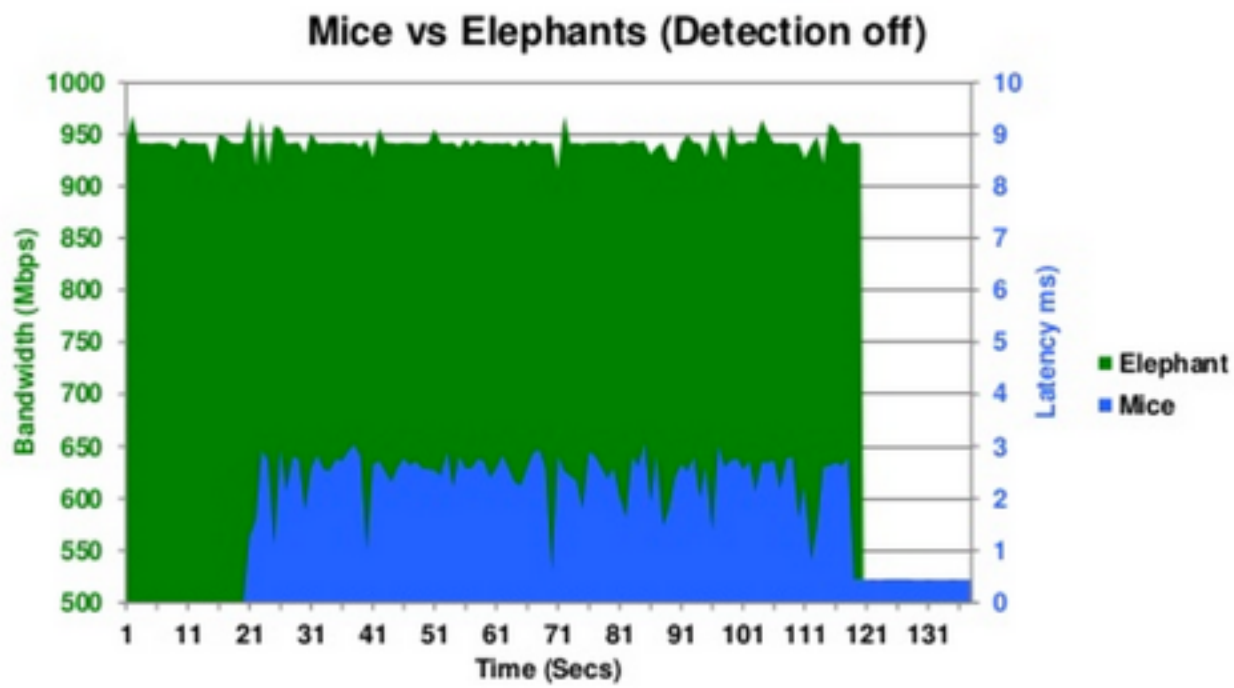
Minimal configuration to connect switches to controllers, intelligence resides in external software

Host sFlow and configuration

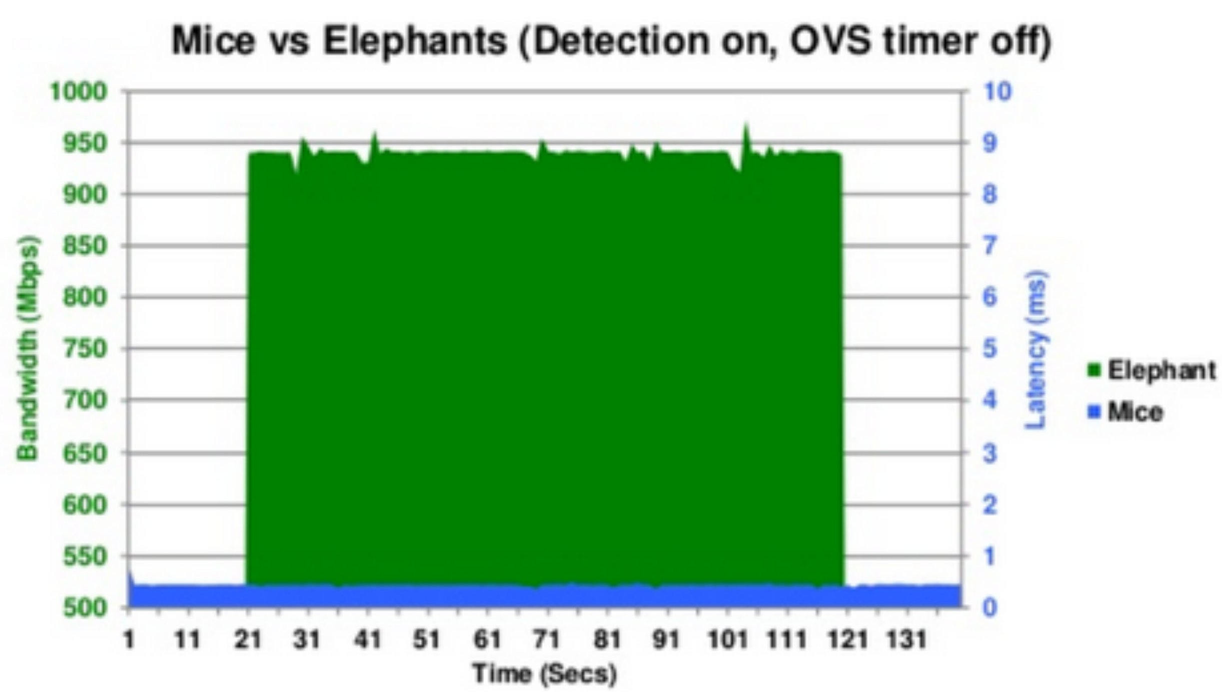


http://blog.sflow.com/2012/01/host-sflow-distributed-agent.html

Benefits of marking



Without marking, large flows “Elephants” impact latency of small flows “Mice”



Marking and priority queuing, minimal impact on large flow throughput and greatly improved latency for small flows

VMware / Cumulus demonstration

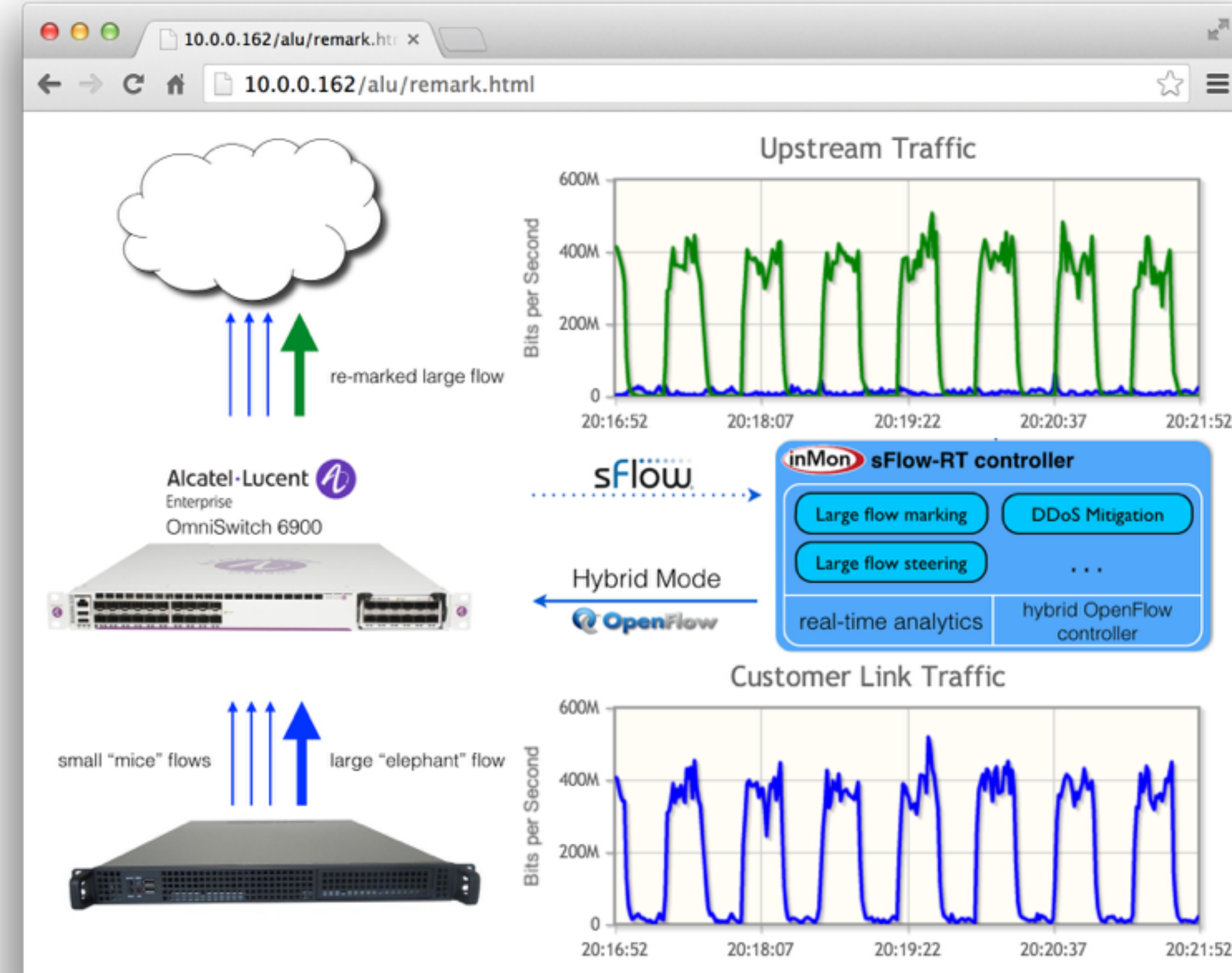
http://www.slideshare.net/martin_casado/elephants-and-mice-elephant-detection-in-the-vswitch-with-hardware-handling

sFlow settings for large flow detection

Define large flow as flow consuming 10% of link bandwidth for 1 second

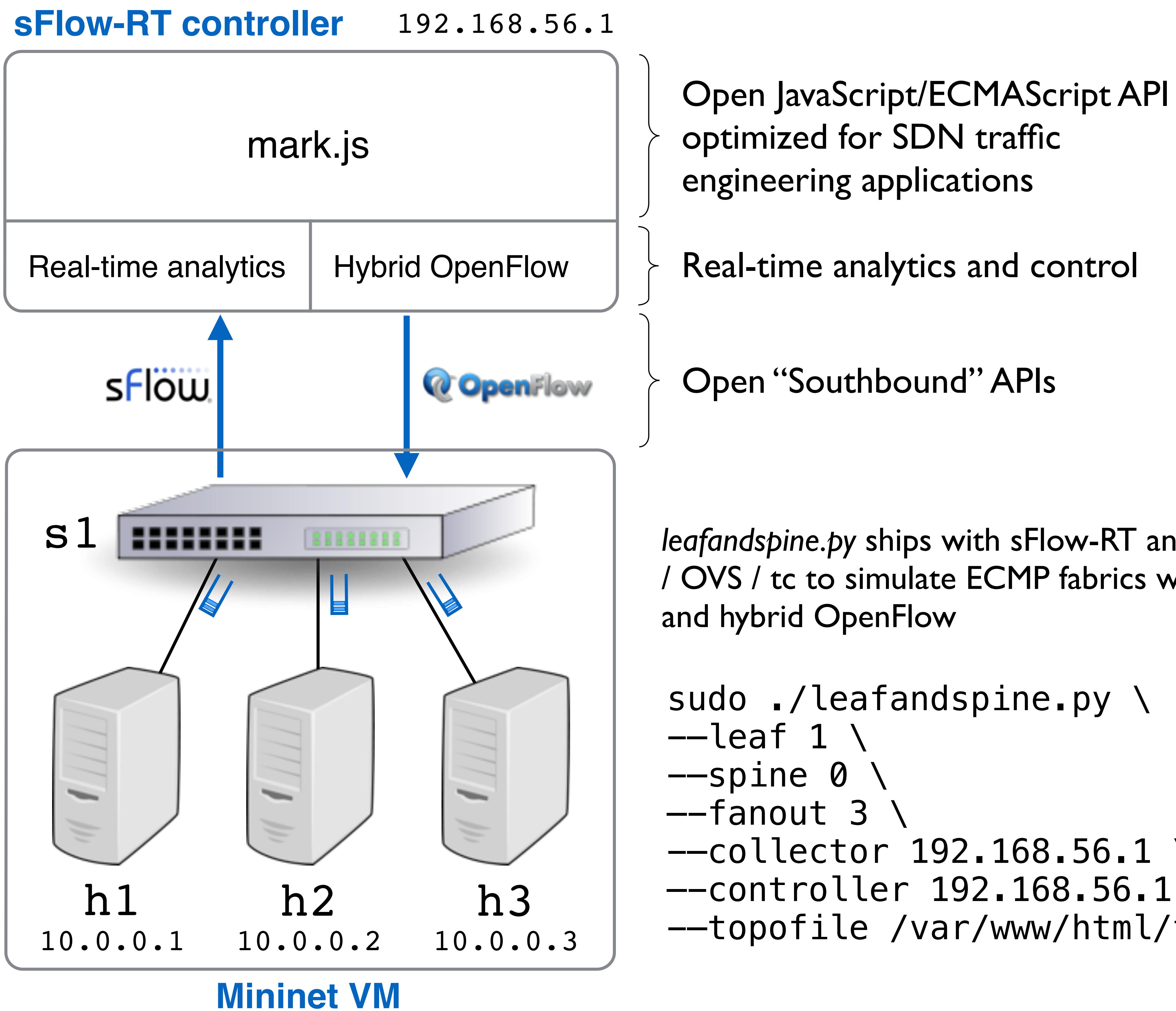
Link Speed	Large Flow	Sampling Rate	Polling Interval
10 Mbit/s (Mininet)	>= 1Mbit/s	1-in-10	20 seconds
1 Gbit/s	>= 100 Mbit/s	1-in-1,000	20 seconds
10 Gbit/s	>= 1 Gbit/s	1-in-10,000	20 seconds
25 Gbit/s	>= 2.5 Gbit/s	1-in-25,000	20 seconds
40 Gbit/s	>= 4 Gbit/s	1-in-40,000	20 seconds
50 Gbit/s	>= 5 Gbit/s	1-in-50,000	20 seconds
100 Gbit/s	>= 10 Gbit/s	1-in-100,000	20 seconds

Marking demo with physical switch



http://blog.sflow.com/2014/03/performance-optimizing-hybrid-openflow.html

Mininet demonstration



leafandspine.py ships with sFlow-RT and configures Mininet / OVS / tc to simulate ECMP fabrics with priority queueing and hybrid OpenFlow

```
sudo ./leafandspine.py \
--leaf 1 \
--spine 0 \
--fanout 3 \
--collector 192.168.56.1 \
--controller 192.168.56.1 \
--topofile /var/www/html/topology.json
```

mark.js

```
// Define large flow as greater than 1Mbits/sec for 1 second or longer
var bytes_per_second = 1000000/8, duration_seconds = 1;

// get topology from Mininet
setTopology(JSON.parse(http("http://192.168.56.101/topology.json")));

// define TCP flow cache
setFlow('tcp',
 {keys:'ipsource,ipdestination,tcpsourceport,tcpdestinationport', filter:'direction=ingress',
  value:'bytes', t:duration_seconds}
);

// set threshold identifying Elephant flows
setThreshold('elephant', {metric:'tcp', value:bytes_per_second, byFlow:true, timeout:4});

// set OpenFlow marking rule when Elephant is detected
var idx = 0;
setEventHandler(function(evt) {
 if(topologyInterfaceToLink(evt.agent,evt.dataSource)) return;
 var port = ofInterfaceToPort(evt.agent,evt.dataSource);
 if(port) {
  var dpid = port.dpid;
  var id = "mark" + idx++;
  var k = evt.flowKey.split(',');
  var rule= {
    match:{in_port: port.port, dl_type:2048, ip_proto:6, nw_src:k[0], nw_dst:k[1], tcp_src:k[2], tcp_dst:k[3]},
    actions:["set_ip_dscp=8","output=normal"], priority:1000, idleTimeout:2
  };
  logInfo(JSON.stringify(rule,null,1));
  setOfRule(dpid,id,rule);
 }
},['elephant']);
```

Run iperf between h2 and h3

peter — pp@mininet: ~ — ssh — 80×24

```
*** Cleanup complete.
pp@mininet:~$ ./start.sh
*** Creating network
*** Adding controller
Unable to contact the remote controller at 192.168.56.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(10.00Mbit) (10.00Mbit) (h1, s1) (10.00Mbit) (10.00Mbit) (h2, s1) (10.00Mbit) (1
0.00Mbit) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1 (10.00Mbit) (10.00Mbit) (10.00Mbit)
*** Configuring sFlow collector=192.168.56.1
*** Starting CLI:
mininet> xterm h1
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['9.56 Mbits/sec', '10.8 Mbits/sec']
mininet>
```

Ping between h1 and h3 shows increased response time (consistent with 3ms result from physical switch)

Node: h1

```
root@mininet:~# ./pingtest 10.0.0.3 10
 0.036 ms
* 0.139 ms
* 0.188 ms
* 0.153 ms
* 0.142 ms
******************* 1.90 ms
*************** 1.51 ms
************ 1.20 ms
*********** 1.11 ms
********************************** 3.49 ms
********************************** 3.49 ms
 0.088 ms
 0.067 ms
 0.066 ms
 0.072 ms
 0.071 ms
 0.073 ms
 0.073 ms
 0.083 ms
 0.071 ms
```

Start sFlow-RT controller running mark.js application

sflow-rt — java — 80×24

```
PETERs-MacBook-Pro-4:sflow-rt peter$ ./start.sh
2014-11-13T17:50:13-0800 INFO: Listening, OpenFlow port 6633
2014-11-13T17:50:14-0800 INFO: Listening, sFlow port 6343
2014-11-13T17:50:14-0800 INFO: Listening, http://localhost:8008
2014-11-13T17:50:14-0800 INFO: mark.js started
```

Repeat iperf between h2 and h3
(no significant change in throughput)

peter — pp@mininet: ~ — ssh — 80×24

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(10.00Mbit) (10.00Mbit) (h1, s1) (10.00Mbit) (10.00Mbit) (h2, s1) (10.00Mbit) (1
0.00Mbit) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1 (10.00Mbit) (10.00Mbit) (10.00Mbit)
*** Configuring sFlow collector=192.168.56.1
*** Starting CLI:
mininet> xterm h1
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['9.56 Mbits/sec', '10.7 Mbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['9.56 Mbits/sec', '10.7 Mbits/sec']
mininet>
```

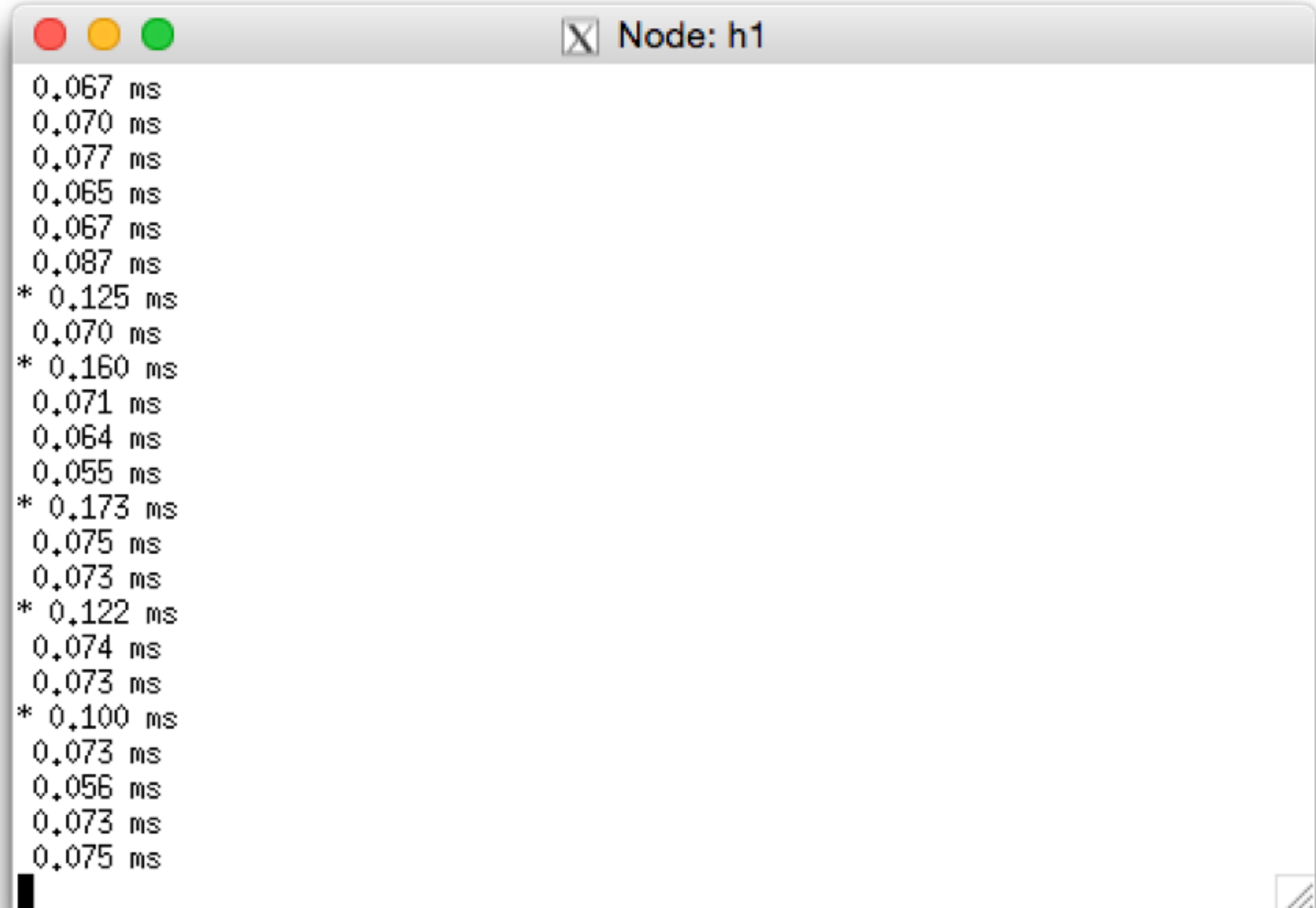
Controller detects large flow and applies OpenFlow rule (flow marked DSCP 8, placing flow in low priority queue)

sflow-rt — java — 80×24

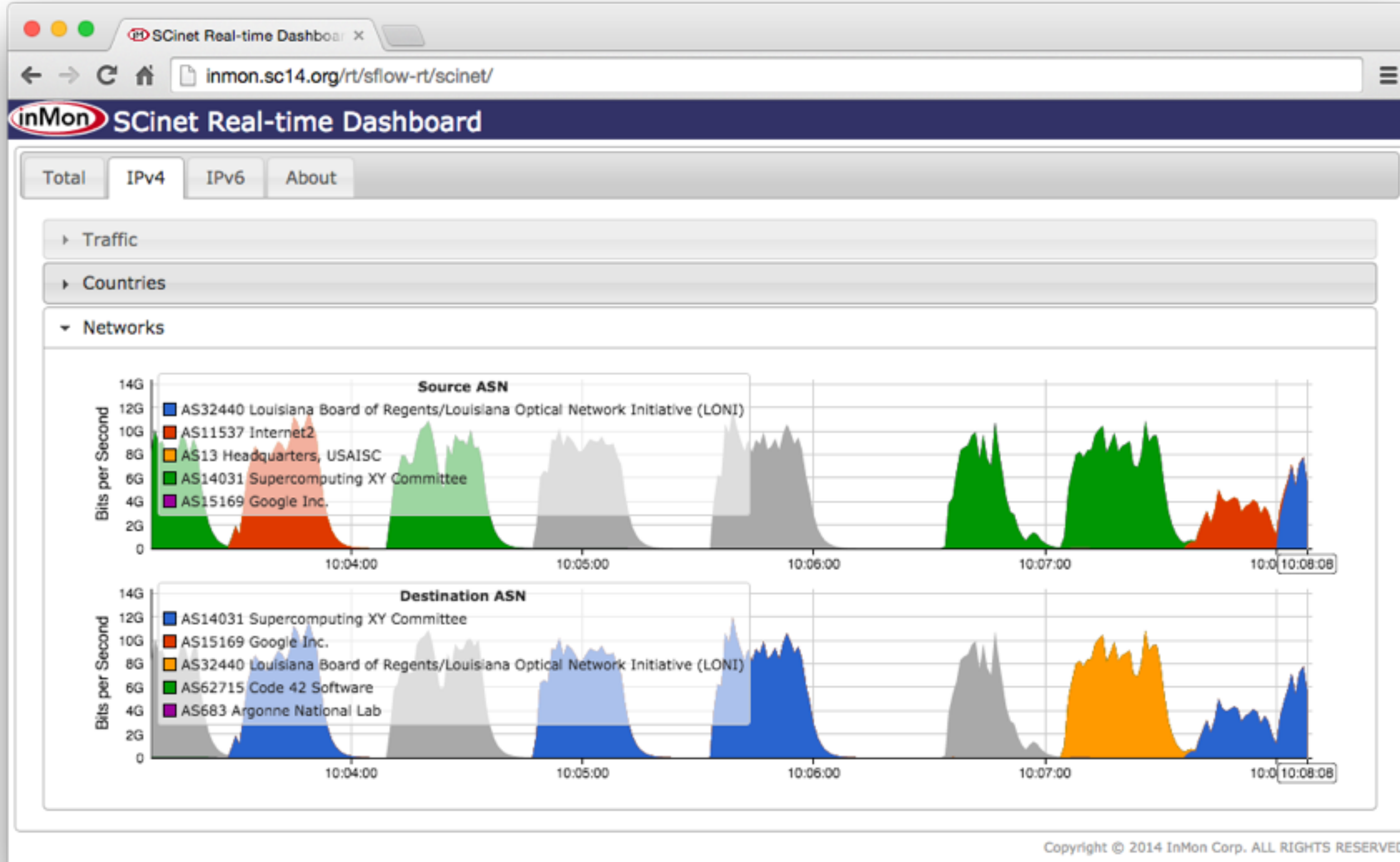
```
2014-11-13T17:21:21-0800 INFO: mark.js started
2014-11-13T17:21:21-0800 INFO: OF: connected to 192.168.56.101:60288 using OF 1.
0
2014-11-13T17:21:21-0800 INFO: OF1.0: 192.168.56.101:60288 = datapath 0000000000
000001
2014-11-13T17:21:21-0800 INFO: OF1.0: datapath 0000000000000001 added to control
ler
2014-11-13T17:21:27-0800 INFO: {
```


```
 "match": {
  "in_port": "2",
  "dl_type": 2048,
  "ip_proto": 6,
  "nw_src": "10.0.0.2",
  "nw_dst": "10.0.0.3",
  "tcp_src": "34410",
  "tcp_dst": "5001"
 },
 "actions": [
  "set_ip_dscp=8",
  "output=normal"
 ],
 "priority": 1000,
 "idleTimeout": 2
}
```

Ping between h1 and h3 shows steady response time (consistent with 0.2ms result from physical switch)



Live Dashboard at SC14 in New Orleans this week



http://inmon.sc14.org/rt/sflow-rt/scinet/

Questions?