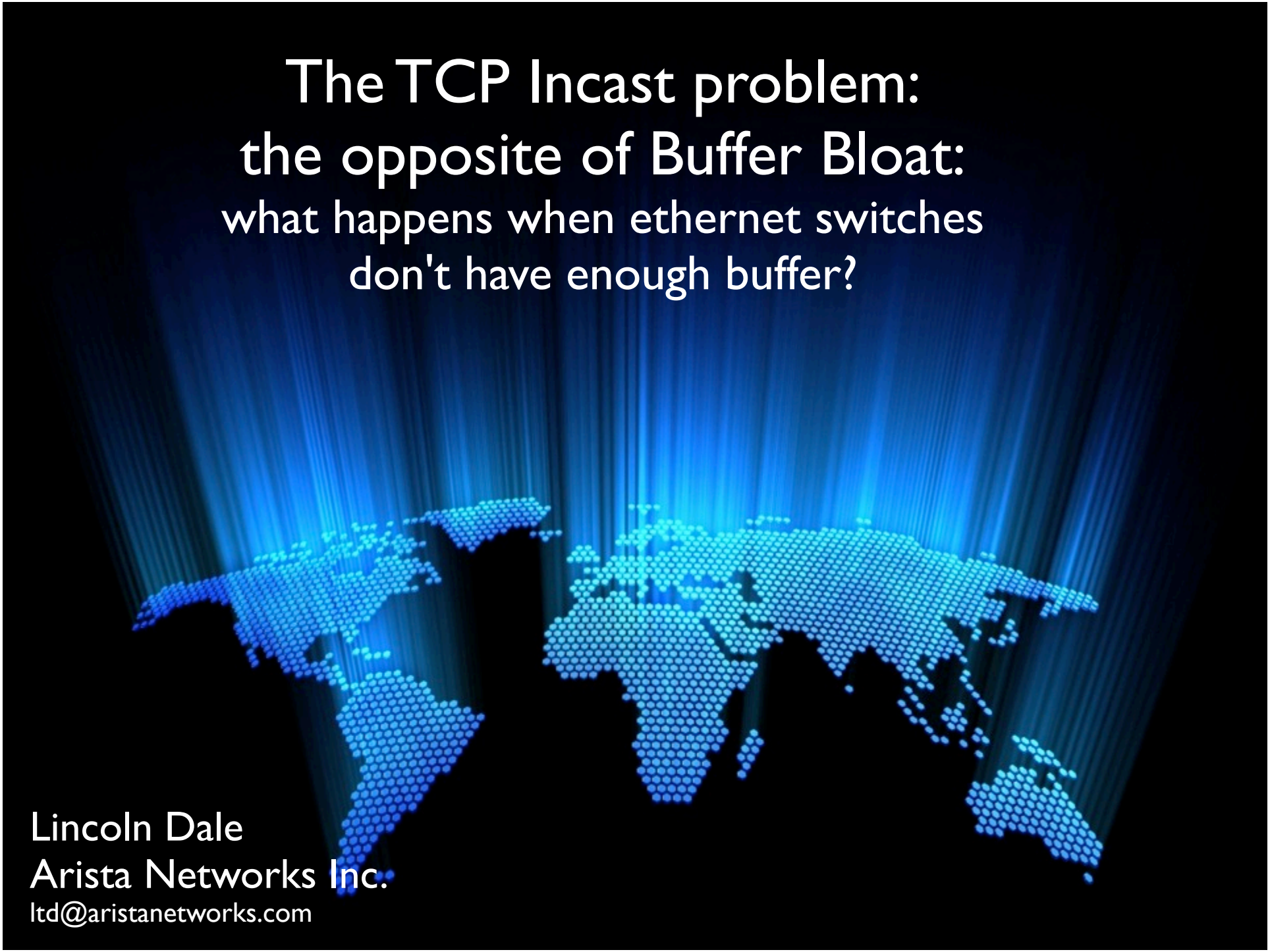
The TCP Incast problem: the opposite of Buffer Bloat:

what happens when ethernet switches don't have enough buffer?



Lincoln Dale
Arista Networks Inc.
ltd@aristanetworks.com

ARISTA

About Arista

How large scale ‘cloud’ network designs are built

Design tradeoffs in how switch silicon is built

Simulations & Reality

Corporate Overview

- Data Center Focus
- Experienced Mgmt and World Class Engineering Team
- Award Winning Products & Differentiators
- Game changing software architecture (EOS)
- Vertical focus on Cloud Networking, HPC, financial services

Executive Team

Jayshree Ullal, President and CEO

- 50 Most Powerful People (Network World, 2005)
- 15yr SVP Cisco for Data Center, Switching, and Services
- Oversaw Catalyst 4500, 6500, and Nexus 7000

Andy Bechtolsheim, Founder, Chairman, and CDO

- Silicon Valley visionary
- Founder of Sun Microsystems
- Founder of Gig Ethernet company Granite Systems
  - Acquired by Cisco in 1996 (Became Catalyst 4xxx)
- Initial investor in Google, Inc.

The Problem with Networks Today

Single Vendor ASICs are inefficient

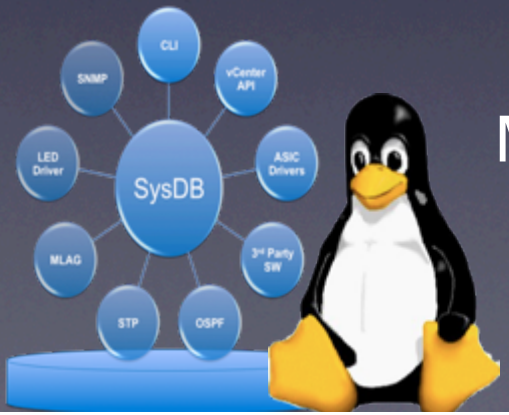
Competitive products designed for email

Network O/Ss are not open

The Arista Solution

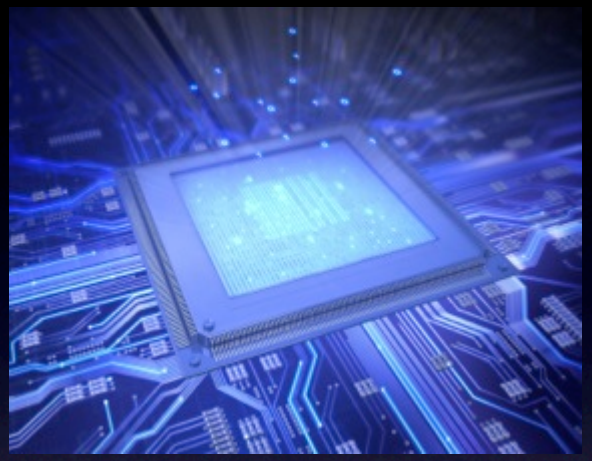
Full custom, faster, denser commercially-available silicon

Designed for data center traffic patterns

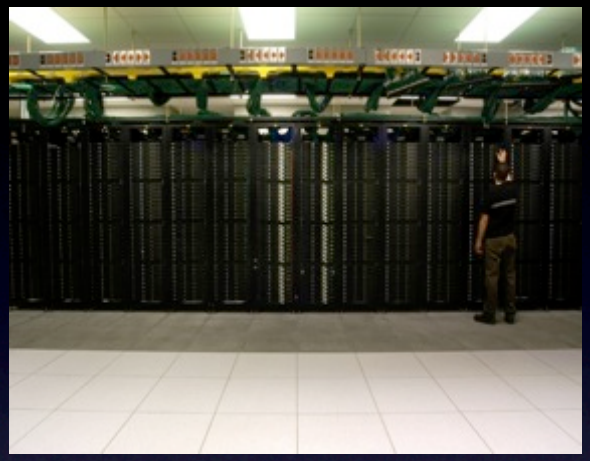


Modern, open, and scalable O/S

Strategy



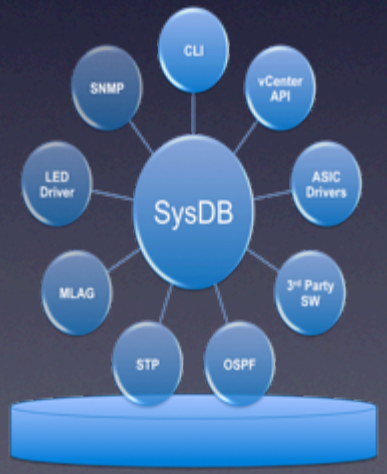
Best Full Custom Silicon Available



100% Data Center Focus

Indirect Sales & Support

Data Center Optimized Designs



Modular EOS vs Spaghetti Code

ARISTA

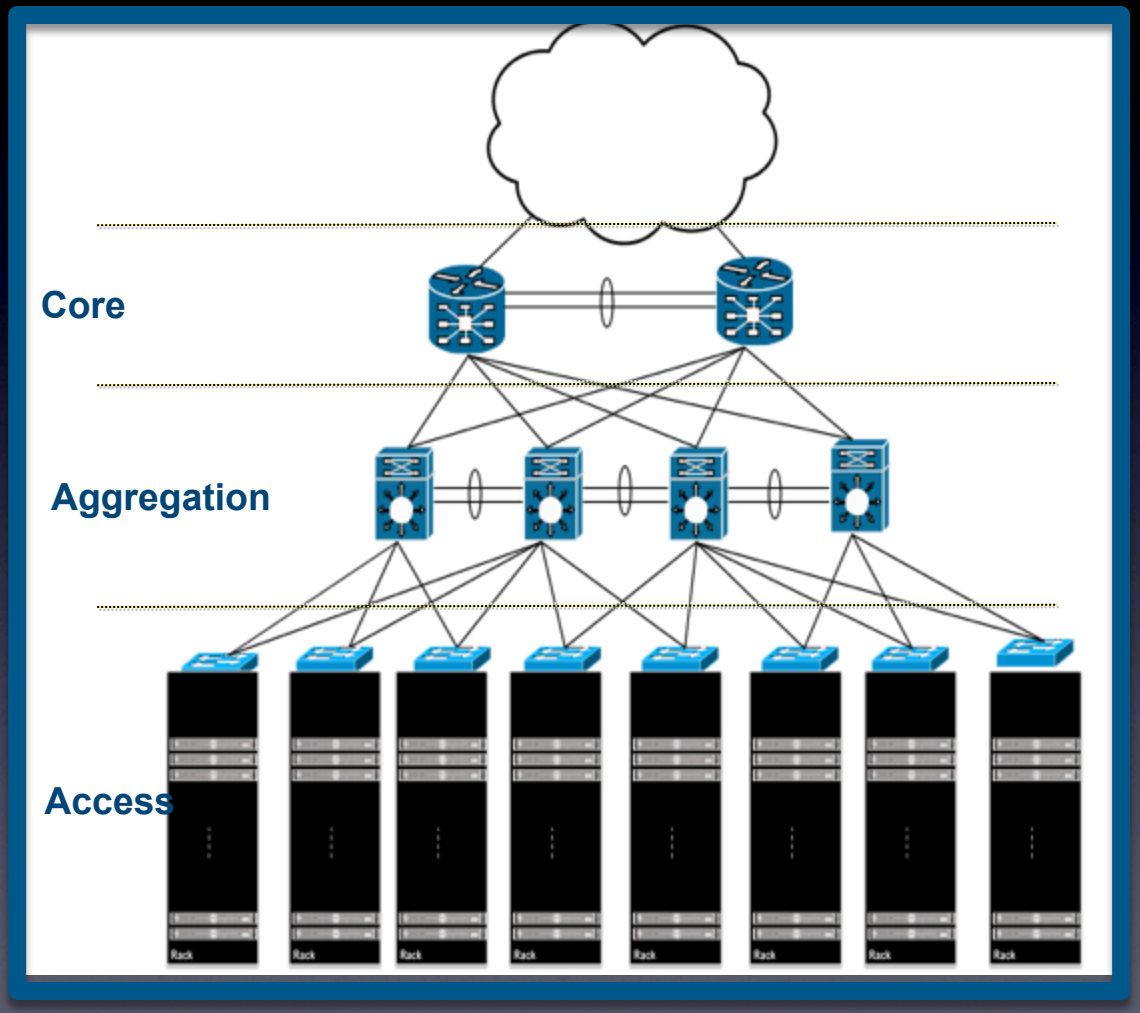
About Arista

How large scale 'cloud' network designs are built

Design tradeoffs in how switch silicon is built

Simulations & Reality

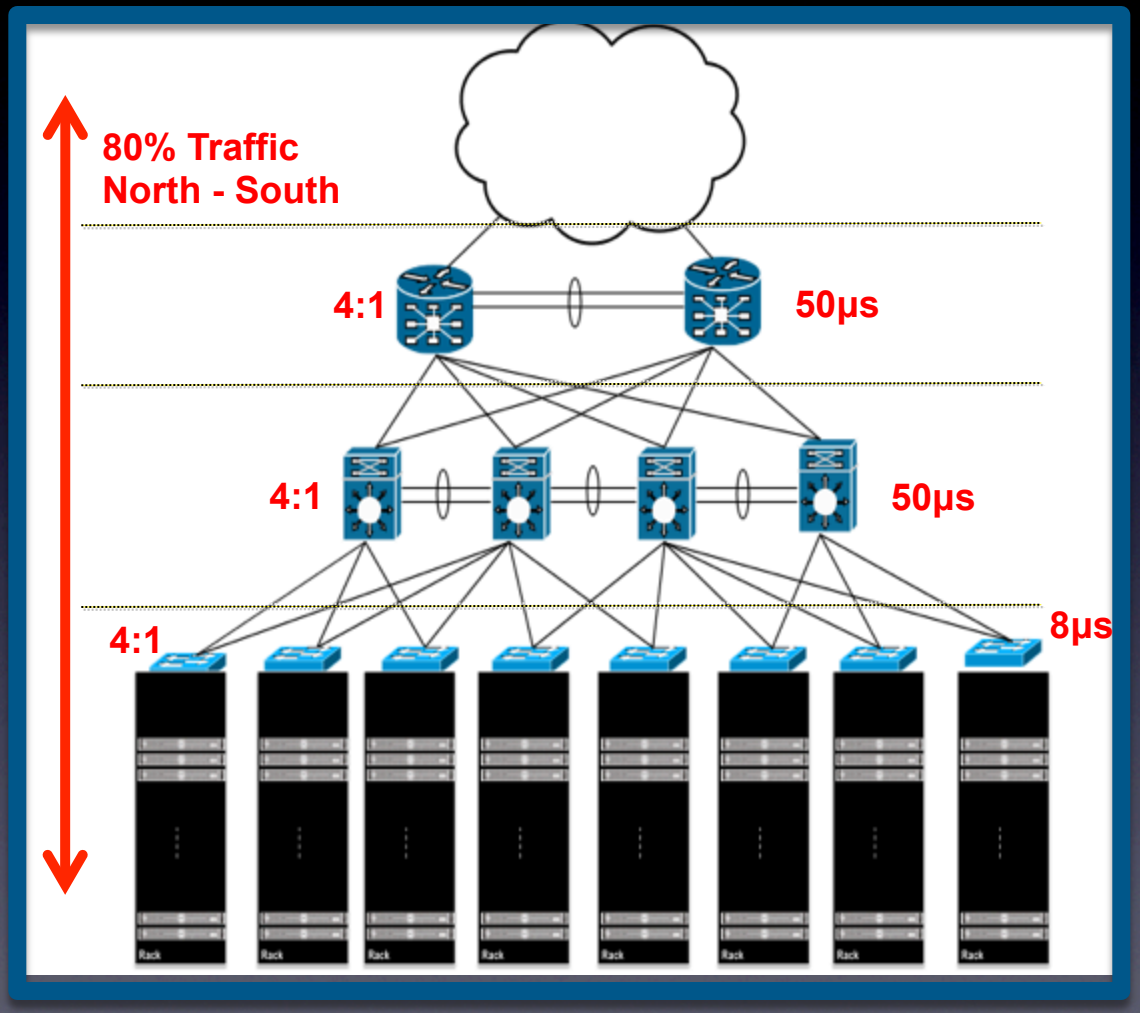
Legacy Datacenter Designs



Generic Attributes

- Mostly Client-Server Traffic
- WAN Latency impacts user experience
- Only 5 to 10% servers transmit at once
- Different architecture for different applications

Limitations of Legacy Designs



Architectural Limitations

- North-South Traffic Flow
- 64:1 to 200:1 Oversubscription
- 75 - 150µs Inter-rack latency
- 20 Racks Maximum Scaling

Requirements for Cloud/Scale-Out Architectures

Facebook:

- Low Inter-Rack Latency
- Non-Blocking Access to Cache/DB

} Optimal User Experience

Windows Azure

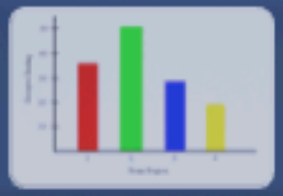
- Cloud based Compute & Storage
- Same Performance as local cluster

} Same performance as local cluster

Oil & Gas Exploration

- Non-Blocking Access
- Low Latency

} Time == Money



Hadoop

- Any-To-Any Access
- Non-Blocking

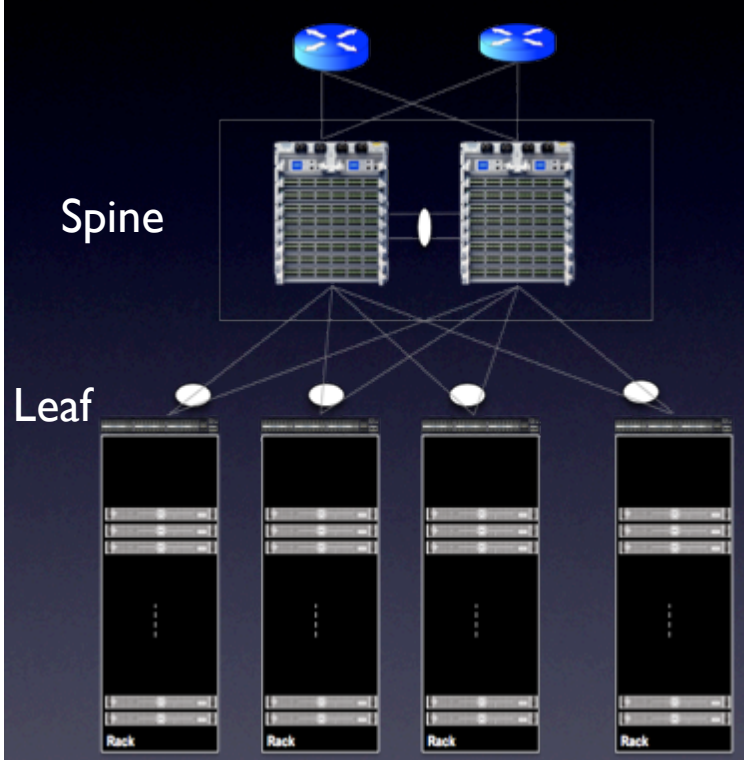
} Smart business decisions

Private Cloud

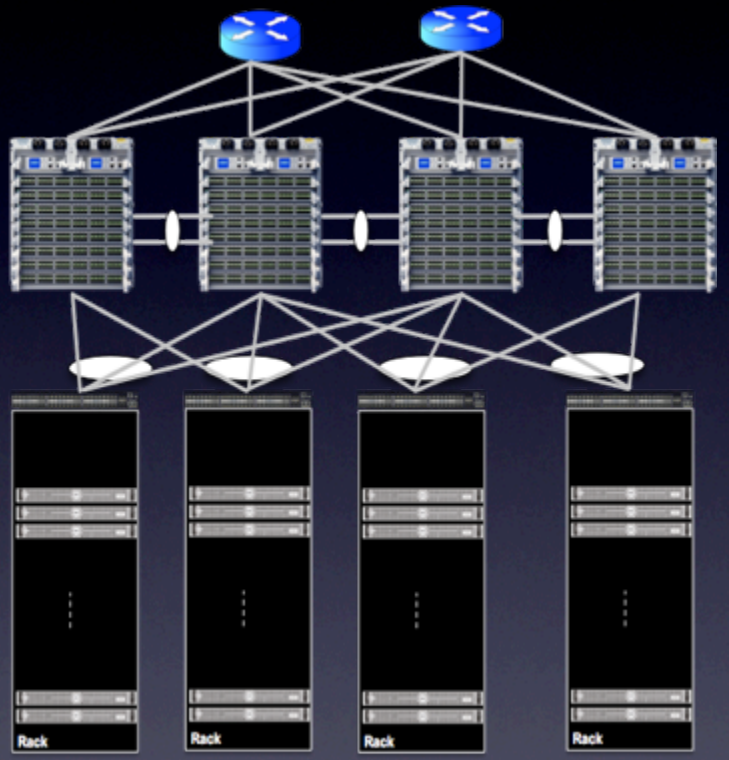
- Uniform design for all apps
- Virtualized

} Rapid service delivery

Two fundamental ways to Scale: L2 or L3

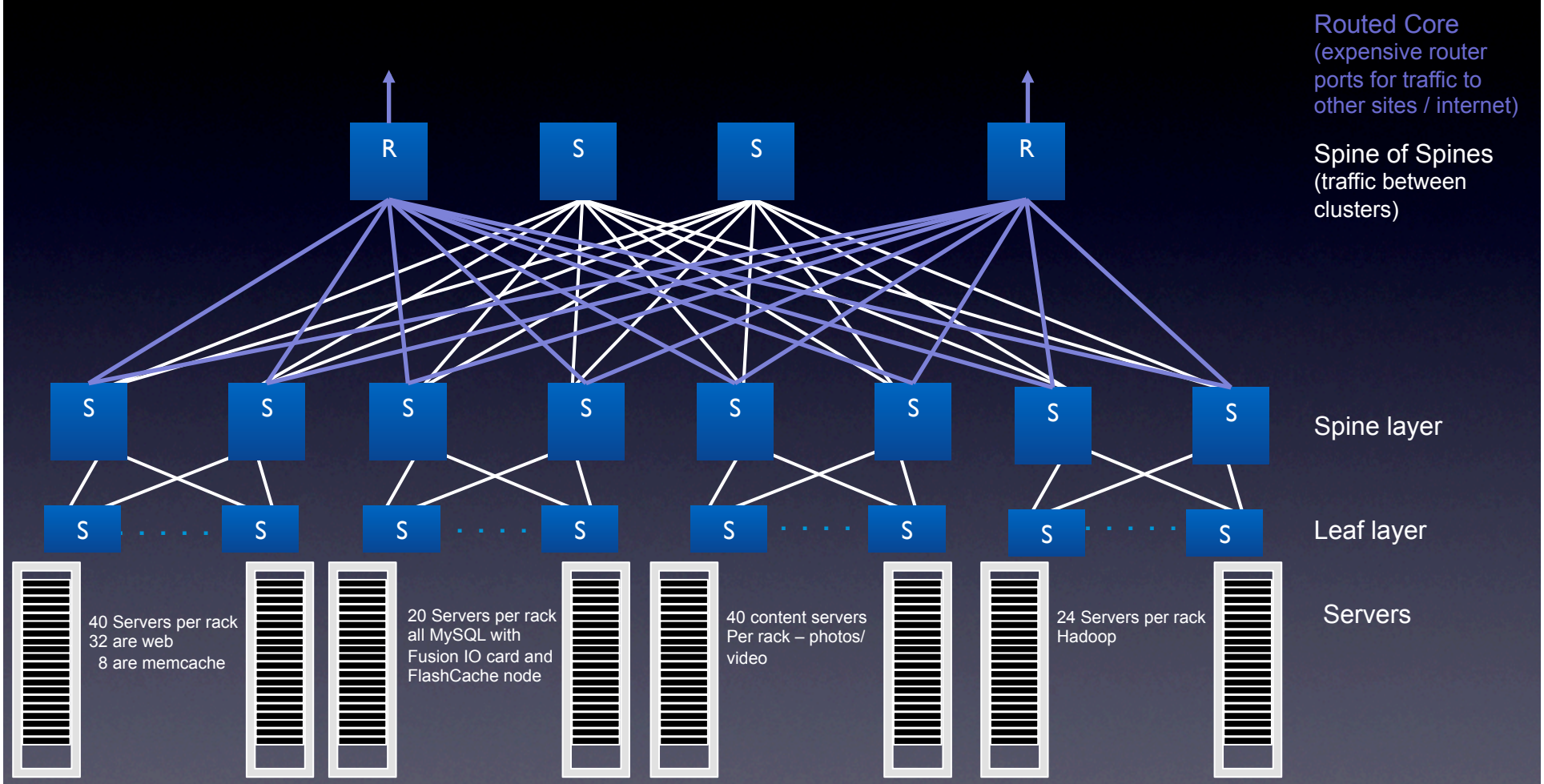


MLAG Spine (L2)
two-way active



ECMP Spine (L3 - OSPF/BGP)
today: up to 16-way ECMP

A typical Large Scale design



ARISTA

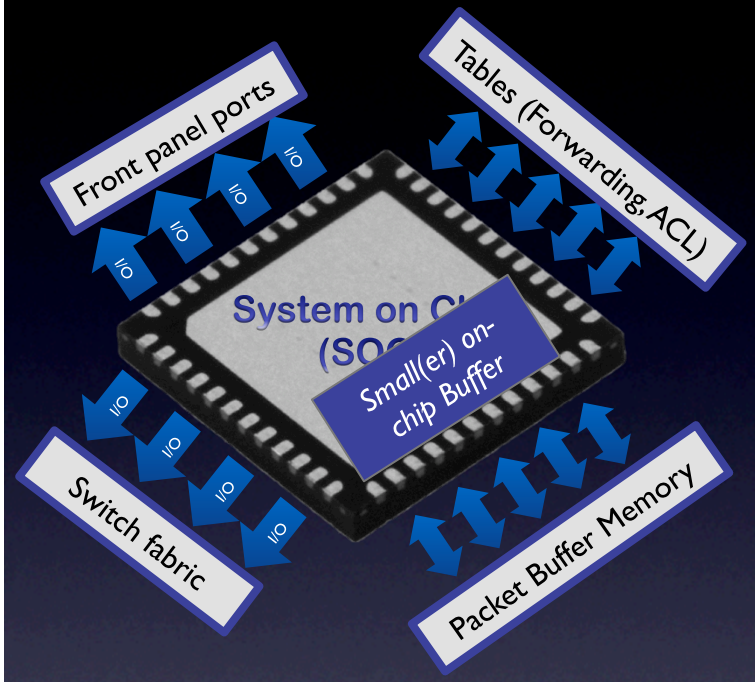
About Arista

How large scale ‘cloud’ network designs are built

Design tradeoffs in how switch silicon is built

Simulations & Reality

Switch silicon design – a matter of tradeoffs



Desire: put everything on a single die

Reality: Constrained by

- Size of die (cost/manufacturability)
- Power (can you cool it, reliability)
- # of pins/bumps (exotic packaging costs)
- Table sizes (larger = better)
- Feature bloat

Moore's law is helping. But it cannot change the laws of physics!

ARISTA

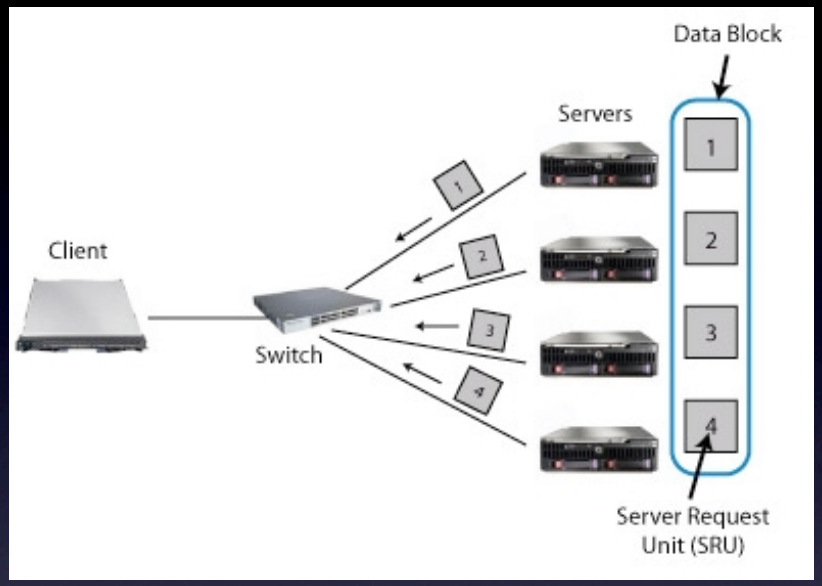
About Arista

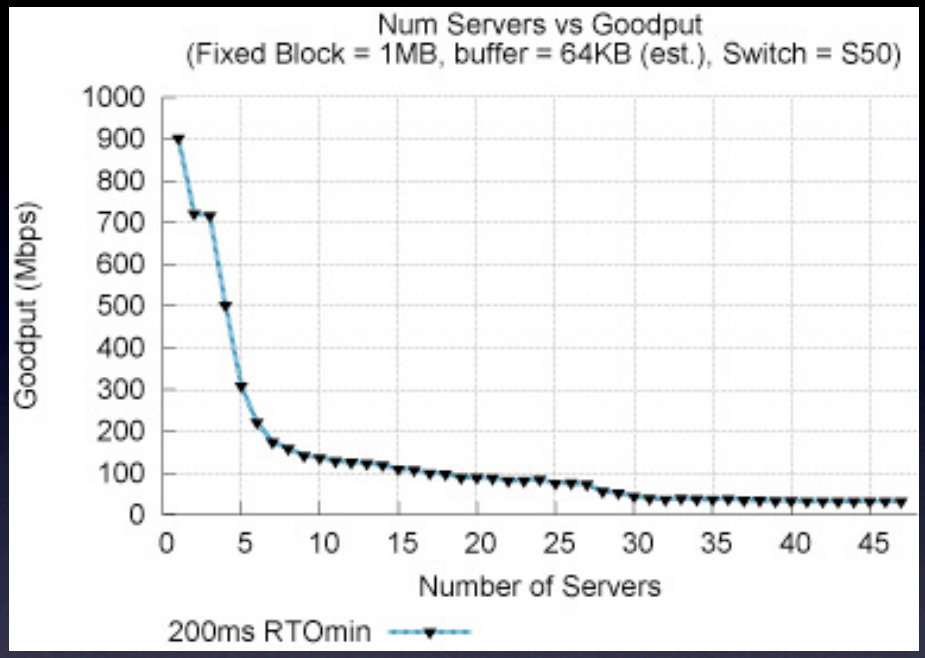
How large scale ‘cloud’ network designs are built

Design tradeoffs in how switch silicon is built

Simulations & Reality

The classic TCP incast problem





Source: http://www.pdl.cmu.edu/Incast/

Simulating TCP incast on a larger scale

NS3 Network simulation tool (http://www.nsnam.org/) can be used to model real world traffic
Good paper talking about its capabilities:
http://typo3.trilogy-project.eu/fileadmin/publications/Other/Lacage-NS3.pdf

NS3 is NOT like yesterday's NS2

- It can source/sink traffic based on any number of criteria
- An accurate model of a buffered crossbar / VoQ switch can be built
- Traffic between end devices can use a real TCP stack (e.g. recent linux-2.6 kernel, FreeBSD, OpenBSD etc)
- pcap files can be generated & therefore analyzed with tools like tcptrace

Perhaps most important of all, we have proven its model is accurate.

Simulating TCP incast on a larger scale

Recreated a 'cloud' network topology with 'many to many' TCP flows

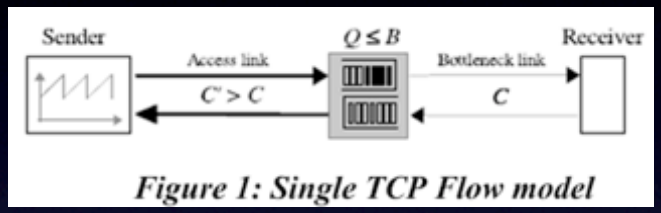


Figure 1: Single TCP Flow model

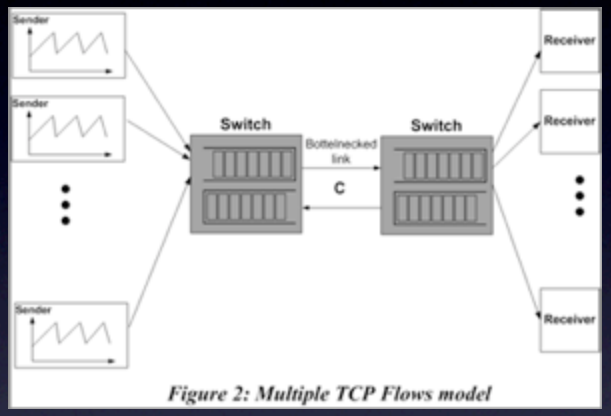


Figure 2: Multiple TCP Flows model

512 'servers' in a spine/leaf network design

- Servers attached to leaf
- 3:1 oversubscription leaf:spine (e.g. for every 12 server ports there is 4 spine links – 48:16)
- Results in 176 ports on spine with each spine dealing with Nx1533 simultaneous TCP streams/port (48 servers x 511 TCP streams x N hashed over 16-way ECMP)

NS3 model 0/3

We created two things in NS3

1. An accurate 'model' of how a switch actually works
   - Modelled on the csma-bridge.cc code included but added
   - Input buffering with virtual output queueing
   - Input buffering with 'cells' of fixed size & frame descriptors
   - Arbitration from egress to ingress (scheduling of traffic)
   - Switching latency & serialization inside-the-switch if components are based on store-and-forward
   - Most importantly of all, that an output port can only be transmitting one frame out a port at one time.
   - (the default csma-bridge code assumed there was never output congestion [even for two ports sending to one at the same time] and a frame always left the output port the same nanosecond it arrived at the input port)
2. A test harness to simulate many senders/receivers and thousands of TCP flows the exact same way you would have it for a hadoop/ bigdata type setup

NS3 model 1/3

```
int
main (int argc, char *argv[])
{
  ns3::Time::SetResolution(Time::NS); // NS

  // cubic is the default congestion algorithm in Linux 2.6.26
  std::string tcpCong = "cubic";

  // this is the default error rate of our link, that is, the the probability of a single
  // byte being 'corrupted' during transfer.
  double errRate = 0; // 0.000001;
  unsigned int runtime = 10;   // how long the sender should be running, in seconds.
  unsigned int nodes = 4;      // number of nodes
  unsigned int streams_per_node = 5;
  unsigned int mtu = 1500;    // mtu

  // the name of the NSC stack library that should be used
  std::string nscStack = "liblinux2.6.26.so";

  CommandLine cmd;
  cmd.AddValue ("TCP_CONGESTION", "Linux 2.6.26 Tcp Congestion control algorithm to use", tcpCong);
  cmd.AddValue ("error-rate", "Error rate to apply to link", errRate);
  cmd.AddValue ("runtime", "How long the applications should send data (default 120 seconds)", runtime);
  cmd.AddValue ("nodes", "number of nodes (default 48)", nodes);
  cmd.AddValue ("streams_per_node", "number of streams per node to every other node (default 10)", streams_per_node);
  cmd.AddValue ("mtu", "mtu to use (1500 default)", mtu);
  cmd.AddValue ("nscstack", "Set name of NSC stack (shared library) to use (default liblinux2.6.26.so)", nscStack);
  cmd.Parse (argc, argv);

  Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue (mtu));
  Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue ("1Gbps"));

  std::cout << "LTD simulation.\n";
  std::cout << "  " << nodes << " nodes, " << streams_per_node << " streams/node, " << runtime << " runtime\n";

  // Explicitly create the nodes required by the topology (shown above).
  NodeContainer n;
  n.Create (nodes);

  // create the 'switch' in the middle
  NodeContainer csmaSwitch;
  CsmaHelper csma;
  csmaSwitch.Create (1);
  csma.SetChannelAttribute ("DataRate", DataRateValue (1000000000));
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (0)));
```

NS3 model 2/3

```
// Create the csma links, from each node to the switch
NetDeviceContainer terminalDevices;
NetDeviceContainer switchDevices;

for (unsigned int i = 0; i < nodes; i++)
  {
    NetDeviceContainer link = csma.Install (NodeContainer (n.Get (i), csmaSwitch));
    terminalDevices.Add (link.Get (0));
    switchDevices.Add (link.Get (1));
  }

// Create the bridge netdevice, which will do the packet switching
Ptr<Node> switchNode = csmaSwitch.Get (0);
BridgeHelper bridge;
bridge.SetDeviceAttribute ("Mtu", UintegerValue(mtu));
bridge.SetDeviceAttribute ("Latency", UintegerValue(6000));   // 6 usec
bridge.SetDeviceAttribute ("Bandwidth", UintegerValue(1000)); // 10G
bridge.SetDeviceAttribute ("CellSize", UintegerValue (384));
bridge.SetDeviceAttribute ("CellCount", UintegerValue (409)); // 384 bytes x 4K cells = 1.5MB

bridge.Install (switchNode, switchDevices);

InternetStackHelper internet;
// The next statement switches the nodes to 'NSC'-Mode.
// It disables the native ns-3 TCP model and loads the NSC library.
//  internet.SetTcp ("ns3::NscTcpL4Protocol","Library",StringValue (nscStack));
internet.Install (n);

if (tcpCong != "cubic") // make sure we only fail if both --nscstack and --TCP_CONGESTION are used
  {
    // This uses ns-3s attribute system to set the 'net.ipv4.tcp_congestion_control' sysctl of the
    // stack.
    // The same mechanism could be used to e.g. disable TCP timestamps:
    // Config::Set ("/NodeList/*/$ns3::Ns3NscStack<linux2.6.26>/net.ipv4.tcp_timestamps", StringValue ("0"));
    Config::Set ("/NodeList/*/$ns3::Ns3NscStack<linux2.6.26>/net.ipv4.tcp_congestion_control", StringValue (tcpCong));
  }
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.0.0.0", "255.255.0.0");
Ipv4InterfaceContainer ipv4Interfaces = ipv4.Assign (terminalDevices);

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

NS3 model 3/3

```
  uint16_t servPort = 8080;
  PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), servPort));
  ApplicationContainer sinkApp = sinkHelper.Install (n);
  sinkApp.Start (Seconds (0.0));
  // this makes sure that the receiver will run 0.5s longer than the sender application.
  sinkApp.Stop (Seconds (runtime + 0.5));

  // This sets up two TCP flows, one from A -> B, one from B -> A for each node
  for (unsigned int i = 0; i < nodes; i++)
    {
      for (unsigned int j = 0; j < nodes; j++)
        {
          if (i != j)
          {
             for (unsigned int k = 0; k < streams_per_node; k++)
             {
               Address remoteAddress (InetSocketAddress (ipv4Interfaces.GetAddress (i), servPort));
               OnOffHelper clientHelper ("ns3::TcpSocketFactory", remoteAddress);
              clientHelper.SetAttribute ("DataRate", DataRateValue (DataRate ("1Gbps")));
              clientHelper.SetAttribute ("PacketSize", UintegerValue (1460));
               clientHelper.SetAttribute ("OnTime", RandomVariableValue (ConstantVariable (1)));
               clientHelper.SetAttribute ("OffTime", RandomVariableValue (ConstantVariable (0)));
               ApplicationContainer clientApp = clientHelper.Install (n.Get (j));
               clientApp.Start (Seconds (0.2));
               clientApp.Stop (Seconds (runtime));
             }
          }
        }
      }

  // This tells ns-3 to generate pcap traces.
  std::stringstream fname;
  fname << "ltd_" << runtime << "_" << nodes << "x" << streams_per_node << "_" << mtu;
  csma.EnablePcapAll (fname.str(), false);

  AsciiTraceHelper ascii;
  //csma.EnableAsciiAll (ascii.CreateFileStream ("ltd.tr"));

  Simulator::Stop (Seconds (runtime + 2));
  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```

Simulating TCP incast on a larger scale

# simutaneous TCP connections	# ports	Peak buffer utilization (ingress buffer)
20K	16 ports	1.74 MB/port
50K	100 ports	3.66 MB/port
120K	200 ports	~3.54 MB/port
300K	400 ports	7.55 MB/port

Simulating 512 'servers' in a spine/leaf network design

- Servers attached to leaf
- 3:1 oversubscription leaf:spine (e.g. for every 12 server ports there is 4 spine links – 48:16)
- Results in 176 ports on spine with each spine dealing with Nx1533 simultaneous TCP streams/port (48 servers x 511 TCP streams x N hashed over 16-way ECMP)

A typical 'cloud' / 'social networking' / 'search engine' has >100K TCP flows/ server.

- >4.8M TCP flows seen by each spine switch downlink port to leaf switch!

Real world buffer utilization

Real World Data	Customer	Real Buffer Utilization Observations	Max Buffer Used
	HPC	Storage Cluster – Medium	33 MB
	Animation	Storage Filer (NFS)	6.2 MB
	Software vendor	Engineering Build Servers (Perforce)	14.9 MB
	Online shopping	Hadoop 2K servers – Big Data	52.3 MB
	Educational	Enterprise Data Center (Virtualization)	52.4 MB

10MB = 1msec buffer @ 10G

This is real world data we have seen from production spine switches.

Switch silicon keeps a high-water-mark record of maximum VoQ buffer utilization for buffering to an output port

Reality: buffers DO matter

ARISTA

Thank You!