CHAPTER 2

MSDC Solution Details and Testing Summary

This chapter discusses Power on Auto Provisioning (PoAP) and fabric protocol scaling.

PoAP

As was discussed earlier, PoAP was used to configure the various logical topologies—one major change for each of 4 cycles (a, b, c, and d) for this phase of testing[1]. Setup and testing is documented below.

The Goals of the PoAP testing can be summarized in 4 bullet points, along with a summary of results:

1. It should be demonstrated that automation of simultaneous initial provisioning of all Leafs, without human intervention, is possible.
- **SUCCESS**. After issuing write erase;reload, no human intervention was needed in order for the switches to load new images/configuration and for the network to reconverge.
2. If failures occur during the PoAP process, there should be troubleshooting steps engineers can take to determine root cause using logs.
- **CONDITIONAL SUCCESS**. Log messages left on bootflash by the PoAP script helped determine root cause of failures in most cases. However some corner cases (bootflash full) prevented logs from being written, and log verbosity is partly dependent on the PoAP script code (which is up to the customer/script author).
    a. Upon failure, PoAP will restart continuously.
    b. On console, abort PoAP process when prompted.
    c. Go through user/pass setup to get to bootflash to read logs.
    d. Problems with PoAP process:
    – PoAP never gets to script execution step
    – bootflash:<ccyymmdd>_<HHMMss>_PoAP_<PID>_init.log files contain log of PoAP process:
      DHCP related problems (DHCP Offer not received, incorrect options in OFFER, etc)
      HTTP/TFTP related problems (couldn't reach server, file not found, etc)
      Check DHCP/TFTP/HTTP/FTP/SFTP server logs for additional information
    e. Errors in script execution:
    – NO STDOUT or STDERR – only what script writes to logfile.

1. Refer to Power On Auto Provisioning (PoAP), page 1-22.

- CCO script writes to bootflash:<ccyymmddHHMMss>_PoAP_<PID>_script.log
- Be verbose in writing to log in scripts b/c no stackdump to use for debugging (but be aware of available space on bootflash)

3. It should be shown that PoAP can take a deterministic amount of time to provision Leafs. This can be a ballpark reference number since actual runtime will depend on the contents of the PoAP script and what a customer is trying to achieve.

- **SUCCESS** Although the actual time to PoAP depends on the PoAP script being implemented, it was observed that a ballpark figure of around 15 minutes. This test was performed using a mix of 3048 and 3064 Leaf devices connected to a 4-wide spine of N7K's using OSPF as the routing protocol. This represents all of the N3K Leaf devices in the 4-wide topology at the time.

  a. Concurrent PoAP of 30 Leaf devices:
  - 14x N3064 Leafs.
  - 16x N3048 Leafs (these were available in lab).
  - Inband PoAP DHCP relay via N7K Spines.
  - Simultaneous PoAP.
  - Single VM for TFTP/FTP/DHCP server
  - PoAP script included image download and switch configuration
  - Runtime: ~15min.

4. The minimum infrastructure needed to support PoAP'ing Leaf devices should be characterized.

- **SUCCESS** Refer to Topology Setup, page 2-2.

PoAP Benefits

Here are a few benefits provided by PoAP:

- Pipelining device configuration
  - Pre-build configurations for Phase N+1 during Phase N.
- Fast reconfiguration of entire topology
  - Phase N complete and configs saved offline.
  - 'write erase' and 'reload' devices and recable testbed.
  - After POAP completes, the new topology fully operational.
- Ensuring consistent code version across testbed/platforms.
- Scripting allows for customization.
- Revision control: config files can be stored in SVN/Git/etc, off-box in a centralized repository, for easy versioning and backup.

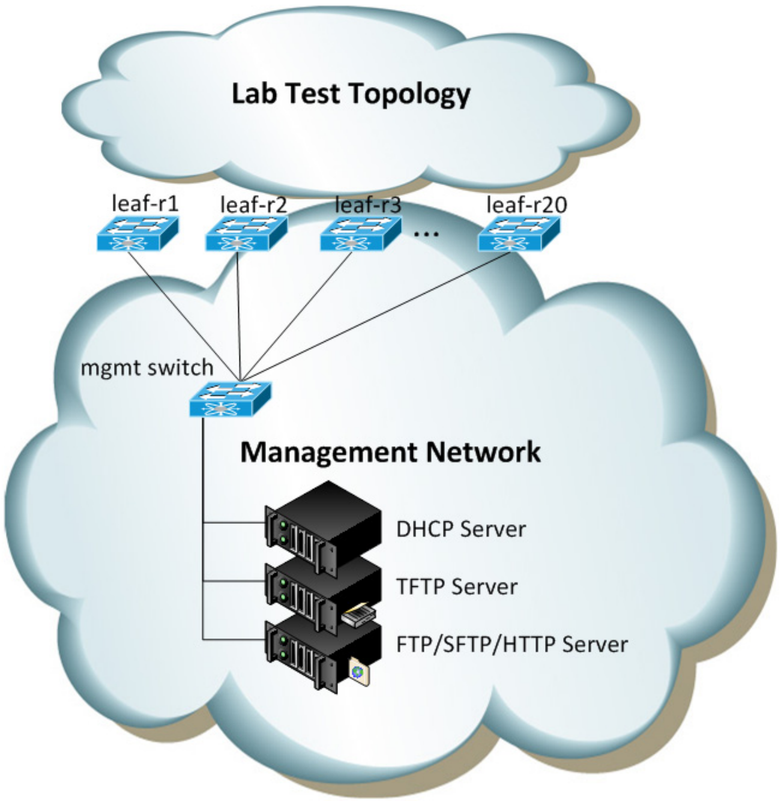
Topology Setup

Each method of enabling PoAP, below, has its pros and cons. One of the most important decisions is how any method scales. MGMT0, page 2-3 and Inband, page 2-3 are two possible ways to enable PoAP in the topology.

MGMT0

Here is a detailed depiction of how PoAP can be used with the mgmt0 interface of each Spine and Leaf node (Figure 2-1).

Figure 2-1 PoAP Across Dedicated Management Network



Pros

- Simple setup (no relay).
- DHCP server can be single homed.
- Single subnet in DHCP config.

Cons

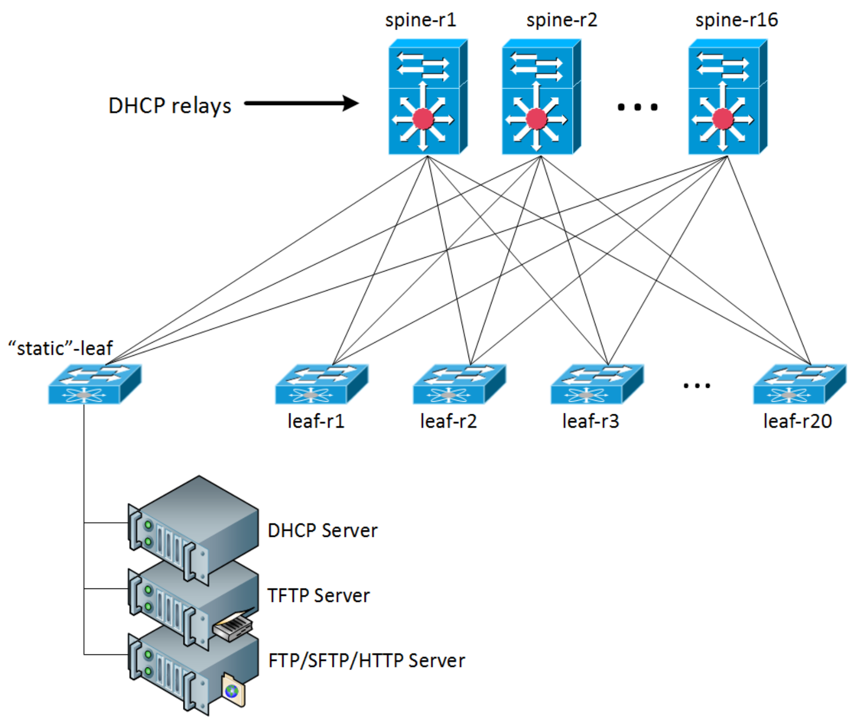
- This is not how most MSDC would deploy. Cost of separate mgmt network at MSDC scales are prohibitive.
- DHCP server could potentially respond to DISCOVERIES from outside the primary network, depending on cabling and configuration.

If using this setup, the PoAP script uses the management VRF.

Inband

In this setup, no mgmt network is used, but rather the normal network (Figure 2-2).

Figure 2-2 PoAP Across Inband Network



Pros

- Customers prefer this method; L3-only, no separate network needed.
- DHCP scope limited to just the main network.

Cons

- Requires DHCP relay on devices.
- When testing, this setup requires extra non-test gear within the topology (dedicated servers).
- DHCP is multi-homed.
- More complex DHCP server configuration.

The test topology used this arrangement for PoAP. The Pros for inband are much higher weighted than all the other cons, and it scales much better than a dedicated L2 network. And with software automation the complexity of DHCP server configuration is easily managed.

Infrastructure

PoAP requires supporting services, such as DHCP, TFTP, FTP/SFTP, and HTTP to properly function. These are discussed below.

DHCP Server

PoAP requires DHCP Offer to contain:

1. IP

2. Subnet
3. routers option
4. domain-name-server option
5. next-server
6. tftp-server-name option
7. bootfile-name option
8. lease time of 1 hour or greater

If PoAP does not get offer with adequate information, init.log will show:

```
poap_dhcp_select_interface_config: No interface with required config
poap_dhcp_intf_ac_action_config_interface_select: Failed in the interface selection to
send DHCPREQUEST for interface 1a013000
```

isc-dhcpd Configuration

Split config into Subnet and Host portions.

- Subnets
  - Switch could DHCP from any interface. Need a subnet entry for every network where DHCP Discover could originate. For inband, that is every point-to-point link where dhcp-relay is configured.
  - IP/Subnet/Router unique for each subnet.
  - Use 'group' to specify same next-server, tftp-server, domain-name-server for all subnets.
- Hosts
  - Host entries need to map Serial Number (prepended with \0) to device hostname.

    ```
    host msdc-leaf-r4 {
    option dhcp-client-identifier "\000FOC1546R0SL";
    option host-name              "msdc-leaf-r4";
    }
    ```
  - Use 'group' to specify same filename, bootfile-name for hosts that will use the same PoAP script.
  - Grouping based on platform, network role, testbed, etc.

TFTP/FTP/SFTP/HTTP Server

- PoAP process on switch downloads PoAP script via TFTP/HTTP. Most tftp servers chroot, so filename but not path is required. For http, configure dhcp option tftp-server-name to be "http://servername.domain.com".
- PoAP script then downloads image and config via TFTP, FTP, SFTP, or SCP.
  - Script will need credentials for login and full path to files
- Host specific config files named directly or indirectly[2].
  - Identified directly by hostname when using os.environ['POAP_HOST_NAME']
  - Best Practice: MAC or S/N mapped to hostname in DHCP config
  - Identified indirectly by serial number, mac address, CDP neighbor.

2. As of this writing hostname is only available in Caymen+ (U4.1) and GoldCoast Maintenance.

- Best Practice: symlink conf_<hostname>.cfg to conf_<serialnum/mac_addr>.cfg
- The load on TFTP/FTP/SFTP servers depends on the PoAP script:
  - Generally, devices PoAP'ing look just like any other TFTP/FTP/SFTP client requests.
  - Best practice: make script intelligen enough to NOT download images if they're already present.
  - Be aware of increased log sizes if enabling debugging on servers for troubleshooting.

Demo

The following collection of logfiles demonstrates a successful PoAP event.

- leaf-r1[3]

```
2012 Jun  4 19:53:22  %$ VDC-1 %$ %NOHMS-2-NOHMS_DIAG_ERR_PS_FAIL: System minor alarm
on power supply 1: failed
Starting Power On Auto Provisioning...
2012 Jun  4 19:54:17  %$ VDC-1 %$ %VDC_MGR-2-VDC_ONLINE: vdc 1 has come online
2012 Jun  4 19:54:17 switch %$ VDC-1 %$ %POAP-2-POAP_INITED: POAP process initialized
Done

Abort Power On Auto Provisioning and continue with normal setup ?(yes/no)[n]:
2012 Jun  4 19:54:37 switch %$ VDC-1 %$ %POAP-2-POAP_DHCP_DISCOVER_START: POAP DHCP
Discover phase started
2012 Jun  4 19:54:37 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: Abort Power On Auto
Provisioning and continue with normal setup ?(yes/no)[n]:
```

- DHCP Server and Script Output. The first reboot happens at 19:55. Then config requiring reboot is applied (system URPF, hardware profile, etc). The first second reboot at 19:58:

```
Jun  4 10:54:19 milliways-cobbler dhcpd: DHCPDISCOVER from 54:7f:ee:34:10:c1 via
10.3.1.32
Jun  4 10:54:19 milliways-cobbler dhcpd: DHCPDISCOVER from 54:7f:ee:34:10:c1 via
10.2.1.32
Jun  4 10:54:19 milliways-cobbler dhcpd: DHCPDISCOVER from 54:7f:ee:34:10:c1 via
10.4.1.32
Jun  4 10:54:19 milliways-cobbler dhcpd: DHCPDISCOVER from 54:7f:ee:34:10:c1 via
10.1.1.32
Jun  4 10:54:20 milliways-cobbler dhcpd: DHCPOFFER on 10.3.1.33 to 54:7f:ee:34:10:c1
via 10.3.1.32
Jun  4 10:54:20 milliways-cobbler dhcpd: DHCPOFFER on 10.2.1.33 to 54:7f:ee:34:10:c1
via 10.2.1.32
Jun  4 10:54:20 milliways-cobbler dhcpd: DHCPOFFER on 10.4.1.33 to 54:7f:ee:34:10:c1
via 10.4.1.32
Jun  4 10:54:20 milliways-cobbler dhcpd: DHCPOFFER on 10.1.1.33 to 54:7f:ee:34:10:c1
via 10.1.1.32
Jun  4 10:54:34 milliways-cobbler dhcpd: DHCPREQUEST for 10.3.1.33 (10.128.3.132) from
54:7f:ee:34:10:c1 via 10.3.1.32
Jun  4 10:54:34 milliways-cobbler dhcpd: DHCPACK on 10.3.1.33 to 54:7f:ee:34:10:c1 via
10.3.1.32
2012 Jun  4 19:54:53 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: Using DHCP, information
received over Eth1/19 from 10.128.3.132
2012 Jun  4 19:54:53 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: Assigned IP address:
10.3.1.33
2012 Jun  4 19:54:53 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: Netmask: 255.255.255.254
2012 Jun  4 19:54:53 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: DNS Server: 10.128.3.136
2012 Jun  4 19:54:53 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: Default Gateway: 10.3.1.32
2012 Jun  4 19:54:53 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: Script Server: 10.128.3.132
```

3. This output is from 5.0(3)U3.2. Output is more verbose in 5.0(3)U4.1.

```
2012 Jun  4 19:54:53 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: Script Name:
/poap_script.py
2012 Jun  4 19:55:04 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: The POAP Script download
has started
2012 Jun  4 19:55:04 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: The POAP Script is being
downloaded from [copy tftp://10.128.3.132//poap_script.py bootflash:script.sh vrf
default ]
2012 Jun  4 19:55:06 switch %$ VDC-1 %$ %POAP-2-POAP_SCRIPT_DOWNLOADED: Successfully
downloaded POAP script file
2012 Jun  4 19:55:06 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: Script file size 15738, MD5
checksum b9b180bd70baee9fabb7a253d59e909a
2012 Jun  4 19:55:06 switch %$ VDC-1 %$ %POAP-2-POAP_INFO: MD5 checksum received from
the script file is b9b180bd70baee9fabb7a253d59e909a
2012 Jun  4 19:55:06 switch %$ VDC-1 %$ %POAP-2-POAP_SCRIPT_STARTED_MD5_VALIDATED:
POAP script execution started(MD5 validated)


$ head -n 1 poap_script.py
#md5sum="b9b180bd70baee9fabb7a253d59e909a"
Mon Jun  4 10:54:50 2012 1 10.3.1.33 886 /var/lib/tftpboot/conf_FOC1539R06D.cfg b _ o
r administrator ftp 0 * c
Mon Jun  4 10:54:51 2012 1 10.3.1.33 0 /var/lib/tftpboot/conf_FOC1539R06D.cfg.md5 b _
o r administrator ftp 0 * i
Mon Jun  4 10:54:53 2012 1 10.3.1.33 3060 /var/lib/tftpboot/conf_mgmt_milliways.cfg b
_ o r administrator ftp 0 * c
Mon Jun  4 10:54:55 2012 1 10.3.1.33 0 /var/lib/tftpboot/conf_mgmt_milliways.cfg.md5 b
_ o r administrator ftp 0 * i
Mon Jun  4 10:54:56 2012 1 10.3.1.33 632 /var/lib/tftpboot/conf_proto_ospf.cfg b _ o r
administrator ftp 0 * c
Mon Jun  4 10:54:58 2012 1 10.3.1.33 0 /var/lib/tftpboot/conf_proto_ospf.cfg.md5 b _ o
r administrator ftp 0 * i


2012 Jun  4 19:55:27 switch %$ VDC-1 %$ %POAP-2-POAP_SCRIPT_EXEC_SUCCESS: POAP script
execution success
2012 Jun  4 19:55:30 switch %$ VDC-1 %$ %PFMA-2-PFM_SYSTEM_RESET: Manual system
restart from Command Line Interface
 writing reset reason 9,
```

· leaf-r1. After second reboot, the remainder of the cofiguration is applied:

```
POAP - Applying scheduled configuration...
2012 Jun  4 19:58:36  %$ VDC-1 %$ %VDC_MGR-2-VDC_ONLINE: vdc 1 has come online
Warning: URPF successfully disabled
Warning: Please copy running-config to startup-config and reload the switch to apply
changes
[########################################] 100%
Done
WARNING: This command will reboot the system
2012 Jun  4 19:58:54 switch %$ VDC-1 %$ %PFMA-2-PFM_SYSTEM_RESET: Manual system
restart from Command Line Interface
 writing reset reason 9,
POAP - Applying scheduled configuration...
2012 Jun  4 20:02:01 switch %$ VDC-1 %$ %VDC_MGR-2-VDC_ONLINE: vdc 1 has come online
Please disable the ICMP redirects on all interfaces
running BFD sessions using the command below
'no ip redirects '
% Warning - the verbose event-history buffer may result in a slow down of OSPF
[########################################] 100%
Done
2012 Jun  4 16:02:36 msdc-leaf-r
msdc-leaf-r1 login:
```

PoAP Considerations

The following PoAP considerations are recommended.

- No "default" config using PoAP
  - If no admin user is configured during PoAP - you'll lock yourself out of the box.
  - No CoPP policy applied to box by default – you must have it in your config.
  - Any IP address received via DHCP during PoAP is discarded when PoAP is complete.
- DHCP Relay issues on N7k
  - CSCtx88353 – DHCP Relay; Boot Reply packet not forwarded over L3 interface
  - CSCtw55298 – With broadcast flag set, dhcp floods resp pkt with dmac=ch_addr
- System configuration after aborted PoAP
  - If PoAP initiated because 'write erase', config will be blank
  - If PoAP initiated by 'boot poap enable', config will be in unknown state. Cannot fall-back to previous config.
- Ensure you have enough free space on bootflash for script logs, downloaded images, and downloaded configs.

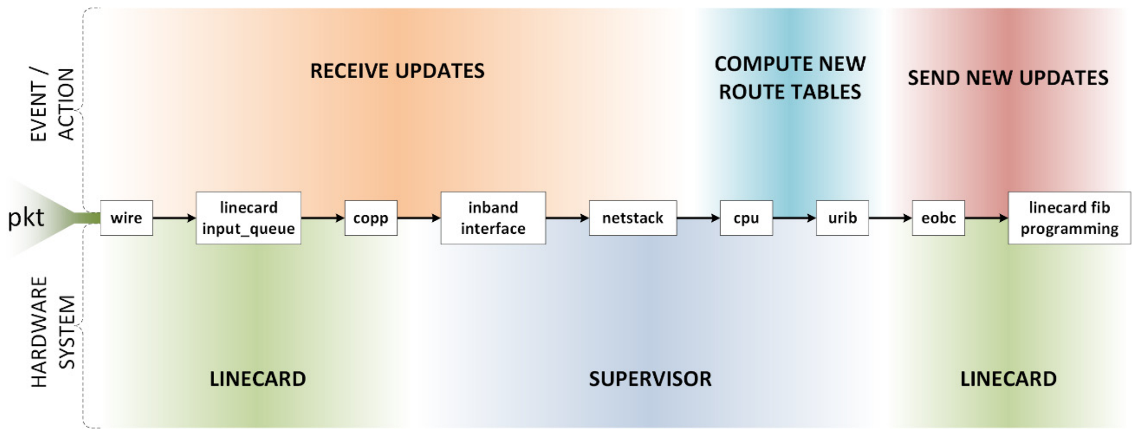
Fabric Protocol Scaling

This section discusses ways to tell if a MSDC is approaching meltdown. Refer to the "Scale" section on page 1-31 for designing MSDC networks to mitigate issues with churn. Figure 2-3 through Figure 2-11 shows and defines the routing and processing subsystems of a packets journey.

Churn

Figure 2-3 is used to describe the day in the life of a packet and how it relates to various routing events and actions.

Figure 2-3 Day in the Life of a Packet Through Routing and Processing Subsystems

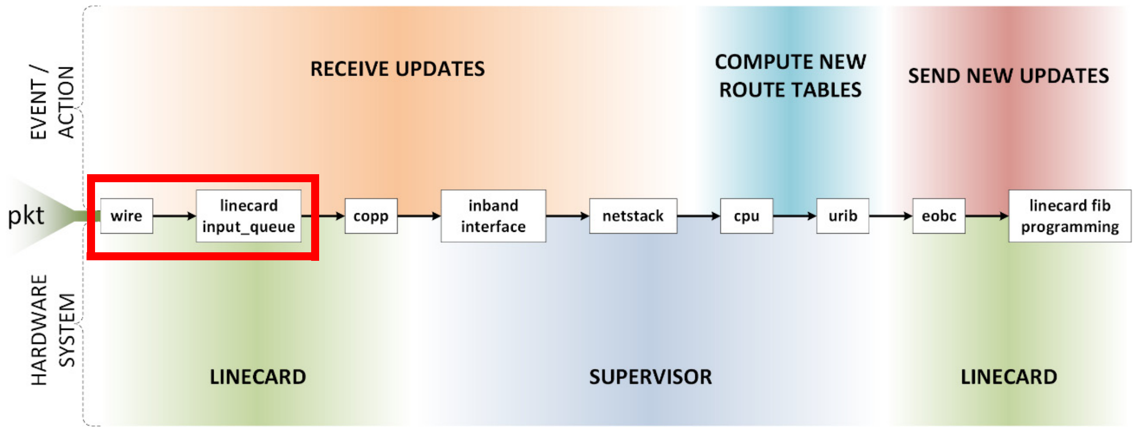


Several terms are used to describe a routing protocol failure; meltdown, cascading failures, etc. The underlying problem in each of these is the network reaches the point where the protocol can no longer keep up. It is so far backed up and sending updates that it becomes the cause of problems instead of routing packets around problems. From an application point of view, this manifests as communication failures between endpoints. But how can one tell from the router point of view that this is occurring? Every routing protocol does three basic things; receive updates, compute new route tables based on these updates, and send out new updates. The most obvious item to check is CPU utilization. If CPU is pegged at 100% computing new route tables, then the limit has obviously been reached. There are, however, other potential breakpoints from when new updates are taken off the wire, to when those updates are processed by the routing protocol, to when new RIB and FIB and generated and pushed to hardware, to when new updates are sent out.

Line Card Input Queues

The first place a packet goes when it comes off the wire is the port's input queue. The architecture of each linecard and platform is different, so the specifics won't be covered here.[4]

Figure 2-4 Line Card Input Queues

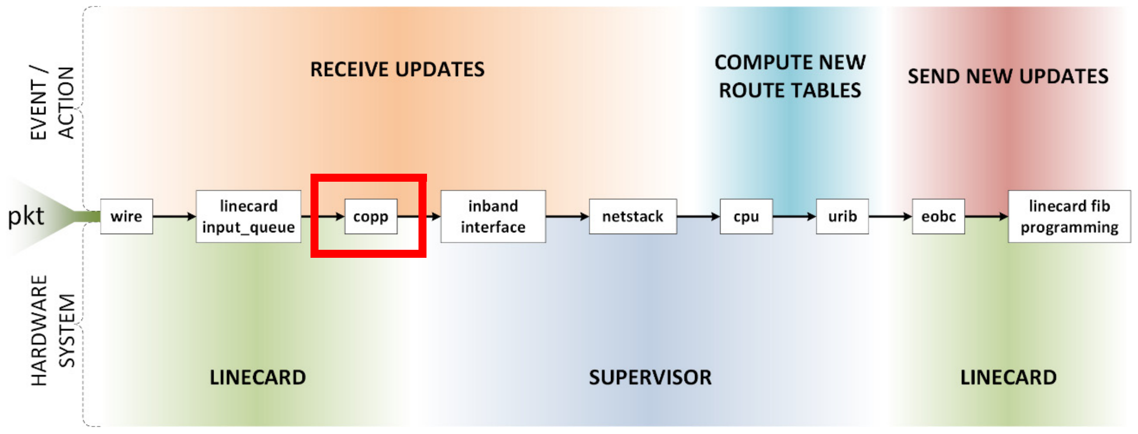


CoPP

Control Plane Policing (CoPP) protects the supervisor from becoming overwhelmed by DDOS type attacks using hardware rate-limiters. The CoPP configuration is user customizable. The default N7k CoPP policy puts all routing protocol packets into the copp-system-p-class-critical class. By default this class is given the strict policy of 1 rate and 2 color and has a BC value of 250ms. The default N3k CoPP policy divides the routing protocol packets into several classes based on each protocol. Should the routing protocol exceed configured rates, packets will be dropped. Dropped Hello's can lead to entire neighbor session being dropped. Dropped updates/LSAs can lead to increased load due to retransmissions or inconsistent routing state.

4. Refer to Appendix C, "F2/Clipper Linecard Architecture,"

Figure 2-5 CoPP Path



CoPP Commands

On the N7k the show policy-map interface control-plane class copp-system-p-class-critical command displays counters for default CoPP class regulating routing protocol traffic. A violated counter that is continuously incrementing indicates network churn rate is approaching meltdown.

```
msdc-spine-r9# show pol int cont class copp-system-p-class-critical | begin mod
      module 3 :
        conformed 14022805664 bytes; action: transmit
        violated 0 bytes; action: drop

      module 4 :
        conformed 8705316310 bytes; action: transmit
        violated 0 bytes; action: drop
```

On the N3k, the show policy-map interface control-plane command displays counters for all CoPP classes. A routing protocol class DropPackets counter that is continuously incrementing indicates the network churn rate is approaching meltdown.

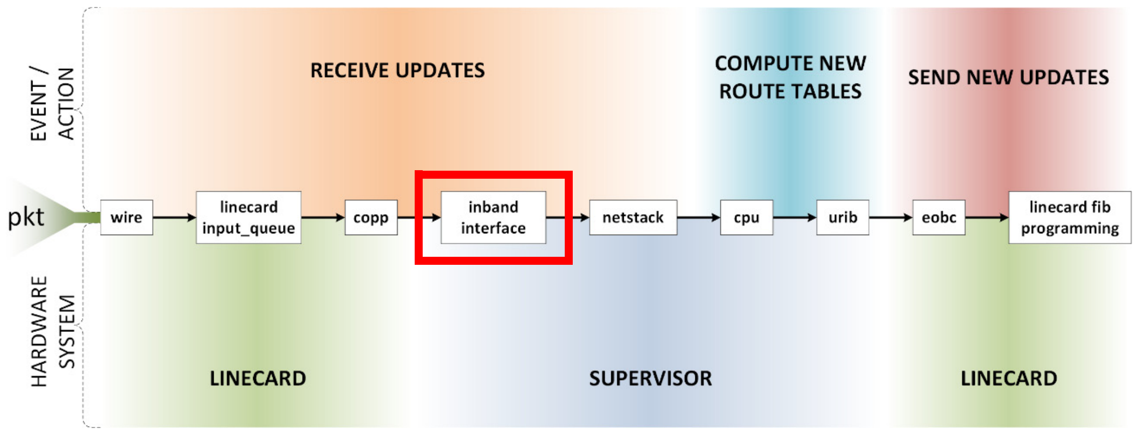
```
msdc-leaf-r21# show policy-map interface control-plane  | begin copp-s-igmp
    class-map copp-s-igmp (match-any)
      match access-grp name copp-system-acl-igmp
      police pps 400
        OutPackets    0
        DropPackets   0
    class-map copp-s-eigrp (match-any)
      match access-grp name copp-system-acl-eigrp
      match access-grp name copp-system-acl-eigrp6
      police pps 200
        OutPackets    0
        DropPackets   0
    class-map copp-s-pimreg (match-any)
      match access-grp name copp-system-acl-pimreg
      police pps 200
        OutPackets    0
        DropPackets   0
    class-map copp-s-pimautorp (match-any)
      police pps 200
        OutPackets    0
        DropPackets   0
    class-map copp-s-routingProto2 (match-any)
      match access-grp name copp-system-acl-routingproto2
      police pps 1300
```

```
          OutPackets    0
          DropPackets   0
      class-map copp-s-v6routingProto2 (match-any)
        match access-grp name copp-system-acl-v6routingProto2
        police pps 1300
          OutPackets    0
          DropPackets   0
      class-map copp-s-routingProto1 (match-any)
        match access-grp name copp-system-acl-routingproto1
        match access-grp name copp-system-acl-v6routingproto1
        police pps 1000
          OutPackets    1208350
          DropPackets   0
      class-map copp-s-arp (match-any)
        police pps 200
          OutPackets    9619
          DropPackets   0
      class-map copp-s-ptp (match-any)
        police pps 1000
          OutPackets    0
          DropPackets   0
      class-map copp-s-bfd (match-any)
        police pps 350
          OutPackets    24226457
          DropPackets   0
  <snip>
```

Supervisor Inband Interface

After making it through CoPP, control plane packets are sent to the supervisor via its inband interface. As the level of network churn increases, it is expected the number of Updates/LSAs sent and received by the device should also increase. A corresponding increase is seen in RX and TX utilization on the inband interface. Should this interface become overwhelmed, throttling occurs and packets will be dropped. Dropped Hello's may lead to entire neighbor sessions being dropped. Dropped updates/LSAs may also lead to increased load due to retransmissions or inconsistent routing state.

Figure 2-6 Inband Interface Path



Supervisor Inband Interface Commands

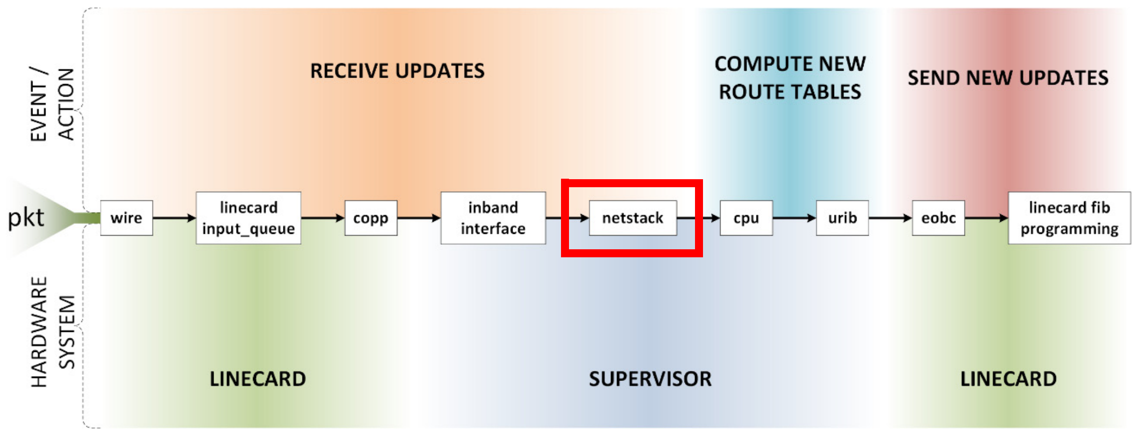
On the N7k, the inband rate limit for Sup1 is 32kpps, while the limit for Sup2 is 64kpps. The show hardware internal cpu-mac inband stats command gives a vast array of statics regarding the inband interface, specifically statistics about throttling. Seeing the rate limit reached counter incrementing indicates the network churn rate is approaching meltdown.

```
msdc-spine-r1# show hard int cpu-mac inband stats | be Throttle | head
Throttle statistics
-----------------------------+---------
Throttle interval ........... 2 * 100ms
Packet rate limit ........... 32000 pps
Rate limit reached counter .. 0
Tick counter ................ 2217856
Active ...................... 0
Rx packet rate (current/max)  261 / 3920 pps
Tx packet rate (current/max)  618 / 4253 pps
```

Netstack

Netstack is the set of NX-OS processes that implement all protocol stacks required to send and receive control plane packets. Routing protocols register with the IP Process to receive their Hello and Update packets. MTS is used to pass these updates between IP Process and routing protocols. When routing protocols are too busy processing previous messages or doing route recalculations to receive these messages, they can be dropped. Dropped Hello's can lead to entire neighbor session being dropped. Dropped updates/LSAs can lead to increased load due retransmissions or inconsistent routing state. Each routing protocol registers as a client of IP process to receive these messages. Statistics are available on a per-client basis.

Figure 2-7 Netstack Path



Netstack Output Commands

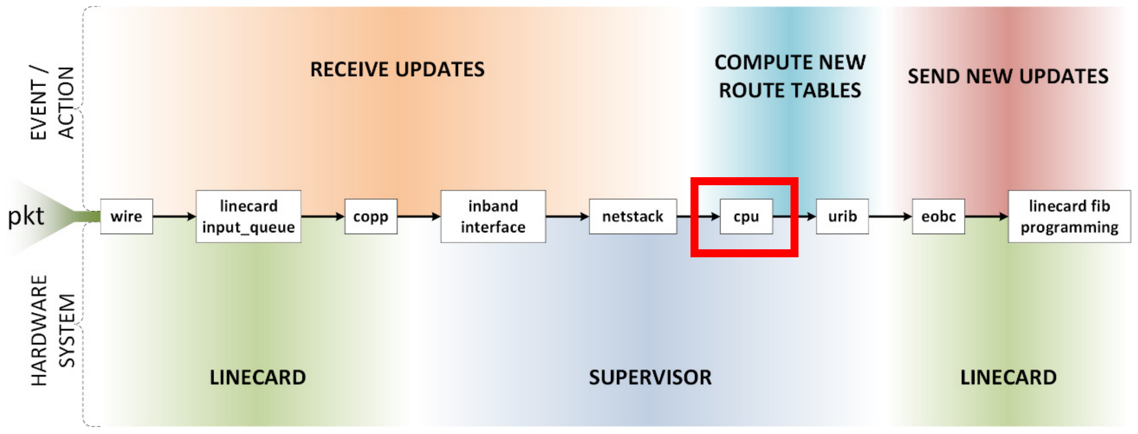
The show ip client command lists all the processes that have registered to receive IP packets. Seeing the failed data messages counter incrementing is an indication that the network churn rate is approaching meltdown.

```
msdc-spine-r9# show ip client ospf

Client: ospf-msdc, uuid: 1090519321, pid: 4242, extended pid: 4242
  Protocol: 89, client-index: 12, routing VRF id: 65535
  Data MTS-SAP: 324, flags 0x3
  Data messages, send successful: 737284, failed: 0

msdc-spine-r8# show ip client tcpudp

Client: tcpudp, uuid: 545, pid: 4416, extended pid: 4416
  Protocol: 1, client-index: 6, routing VRF id: 65535
  Data MTS-SAP: 2323, flags 0x1
  Data messages, send successful: 462, failed: 0
  Recv fn: tcp_process_ip_data_msg (0x8369da6)

Client: tcpudp, uuid: 545, pid: 4416, extended pid: 4416
  Protocol: 2, client-index: 7, routing VRF id: 65535
  Data MTS-SAP: 2323, flags 0x1
  Data messages, send successful: 0, failed: 10
  Recv fn: tcp_process_ip_data_msg (0x8369da6)

Client: tcpudp, uuid: 545, pid: 4416, extended pid: 4416
  Protocol: 6, client-index: 4, routing VRF id: 65535
  Data MTS-SAP: 2323, flags 0x1
  Data messages, send successful: 14305149, failed: 0
  Recv fn: tcp_process_ip_data_msg (0x8369da6)

Client: tcpudp, uuid: 545, pid: 4416, extended pid: 4416
  Protocol: 17, client-index: 5, routing VRF id: 65535
  Data MTS-SAP: 2323, flags 0x1
  Data messages, send successful: 588710, failed: 0
  Recv fn: tcp_process_ip_data_msg (0x8369da6)

Client: tcpudp, uuid: 545, pid: 4416, extended pid: 4416
  Protocol: 112, client-index: 8, routing VRF id: 65535
  Data MTS-SAP: 2323, flags 0x1
  Data messages, send successful: 0, failed: 0
  Recv fn: tcp_process_ip_data_msg (0x8369da6)
```

CPU Utilization

Once the update has reached its final destination, the routing protocol requires compute time on the supervisor to run its SPF or best-path algorithms. As the network converges more frequently, the more load will be put on CPU. However, each platform has a different type of CPU so load will be different on each platform. Also, the location of the device in the network has an impact (routers in an OSPF totally stubby area are insulated from churn in other areas). Thus CPU utilization is one metric to carefully examine, but monitoring all devices is required until it is determined which platform+roles will be high water marks. If the network melts before any devices have pegged the CPU, then one of the other breakpoints are being reached first.

Figure 2-8 CPU Usage



CPU Utilization Commands

The following CPU usage commands were used:

- show process cpu sort
- show process cpu hist
- show system resources module all

```
msdc-spine-r1# show proc cpu sort | exc 0.0%

PID    Runtime(ms)  Invoked   uSecs  1Sec    Process
-----  -----------  --------  -----  ------  -----------
 3929          229        87   2641    6.8%  netstack
 4347      4690520   3655116   1283    2.9%  statsclient
 3824      5842819   2004444   2914    2.0%  diagmgr
 4223      9112189  35562230    256    2.0%  stp
   26       507049   1086599    466    0.9%  kide/1
 3983     33557935   1148416  29221    0.9%  sac_usd
 4034      5259725   1575385   3338    0.9%  oc_usd
 4218      1484069   4998255    296    0.9%  diag_port_lb
 4235      1991337   1127732   1765    0.9%  udld

CPU util  :   5.0% user,    4.5% kernel,    90.5% idle
Please note that only processes from the requested vdc are shown above
msdc-spine-r1# show proc cpu hist

     1     11  226 2  111 211    111      4554 353 2  2 1 3
    696787708864288269140716978855989375663843527196860868197579
100
 90
 80
 70              #
 60              #                         #    #
 50              #                         ##   #
 40              #                        ####  #
 30              #                        #### ### #      #
 20  #      #  ### #      ###    ###      #### ### #  # # #
 10 ################################# ##########################
    0....5....1....1....2....2....3....3....4....4....5....5....
              0    5    0    5    0    5    0    5    0    5

               CPU% per second (last 60 seconds)
```

```
                      # = average CPU%


                                       1            1     11
    777877697797678967989767785988798980787586978798098788009679
    166077546715148676827549868699342800060935474641066850000773
100                *  * *      *     * *            **     **
 90        *  *   **  * *    * *** * * * *    *   * *** * ***  *
 80  ***** *****  ** ***** *** *** ***** *  * *************** **
 70 ************ ********* *** ************ ********************
 60 ************************************************************
 50 ************************************************************
 40 **#****#**********#******#*#*****#******#*#*****#*****##***#
 30 **##*#*##*#***#***#*##***#*###***#*###**#*###***#*#***##**##
 20 ###############*############################################
 10 ############################################################
    0....5....1....1....2....2....3....3....4....4....5....5....
              0    5    0    5    0    5    0    5    0    5

               CPU% per minute (last 60 minutes)
              * = maximum CPU%   # = average CPU%


    111111111111 1111111 111111111111111111111111111111111111111111111111111
    000000000000900000009000000000000000000000000000000000000000000000000000
    000000000000900000006000000000000000000000000000000000000000000000000000
100 **************************##########################********************
 90 **************************##########################********************
 80 **************************###########################*******************
 70 *************************############################*******************
 60 *************************############################*******************
 50 ******#*****************################################################
 40 ******#****************#################################################
 30 ###*###*####***********#################################################
 20 ########################################################################
 10 ########################################################################
    0....5....1....1....2....2....3....3....4....4....5....5....6....6....7.
              0    5    0    5    0    5    0    5    0    5    0    5    0

                   CPU% per hour (last 72 hours)
                  * = maximum CPU%   # = average CPU%

msdc-spine-r1# show system resources module all
CPU Resources:
-----------------------------------------------------------
  CPU utilization:  Module  5 seconds  1 minute  5 minutes
-----------------------------------------------------------
                       1        25        15         14
                       2        21        15         15
                       3        26        23         21
                       4        14        14         14
                       5        21        15         14
                       6        11        13         13
                       7        11        13         13
                       8        11        12         12
                      10        27        18         19
                      11        23        13         12
                      12        17        11         12
                      13        10        13         12
                      14        10        13         13
                      15        11        12         13
                      16        11        12         12
                      17        11        13         13
```

```
-----------------------------------------------------------
  Processor memory:   Module   Total(KB)   Free(KB)  % Used
-----------------------------------------------------------
                         1      2075900    1339944      35
                         2      2075900    1340236      35
                         3      2075900    1333976      35
                         4      2075900    1339780      35
                         5      2075900    1341112      35
                         6      2075900    1344648      35
                         7      2075900    1344492      35
                         8      2075900    1344312      35
                        10      8251592    6133856      25
                        11      2075900    1344604      35
                        12      2075900    1344904      35
                        13      2075900    1344496      35
                        14      2075900    1344496      35
                        15      2075900    1344808      35
                        16      2075900   •show process cpu sort
```

- show process cpu hist
- show system resources module all

```
msdc-spine-r1# show proc cpu sort | exc 0.0%

PID    Runtime(ms)  Invoked   uSecs  1Sec    Process
-----  -----------  --------  -----  ------  -----------
 3929          229        87   2641    6.8%  netstack
 4347      4690520   3655116   1283    2.9%  statsclient
 3824      5842819   2004444   2914    2.0%  diagmgr
 4223      9112189  35562230    256    2.0%  stp
   26       507049   1086599    466    0.9%  kide/1
 3983     33557935   1148416  29221    0.9%  sac_usd
 4034      5259725   1575385   3338    0.9%  oc_usd
 4218      1484069   4998255    296    0.9%  diag_port_lb
 4235      1991337   1127732   1765    0.9%  udld

CPU util  :   5.0% user,   4.5% kernel,   90.5% idle
Please note that only processes from the requested vdc are shown above
msdc-spine-r1# show proc cpu hist

     1     11  226 2  111 211    111      4554 353 2  2 1 3
    696787708864288269140716978855989375663843527196860868197579
100
 90
 80
 70              #
 60              #                         #    #
 50              #                         ##   #
 40              #                        ####  #
 30              #                        #### ### #      #
 20  #      #  ### #      ###    ###      #### ### #  # # #
 10 ################################# ##########################
    0....5....1....1....2....2....3....3....4....4....5....5....
              0    5    0    5    0    5    0    5    0    5

               CPU% per second (last 60 seconds)
                      # = average CPU%


                                       1            1     11
    777877697797678967989767785988798980787586978798098788009679
    166077546715148676827549868699342800060935474641066850000773
100                *  * *      *     * *            **    **
```

```
 90        *  *   **  * *    * *** * * * *    *   * *** * ***  *
 80  ***** *****  ** ***** *** *** ***** *   * *************** **
 70 ************ ********* *** ************ ********************
 60 ************************************************************
 50 ************************************************************
 40 **#****#**********#******#*#*****#******#*#*****#*****##***#
 30 **##*#*##*#***#***#*##***#*###***#*###**#*###***#*#***##**##
 20 ###############*############################################
 10 ############################################################
    0....5....1....1....2....2....3....3....4....4....5....5....
              0    5    0    5    0    5    0    5    0    5

               CPU% per minute (last 60 minutes)
              * = maximum CPU%   # = average CPU%


    111111111111 1111111 111111111111111111111111111111111111111111111111111
    000000000000900000009000000000000000000000000000000000000000000000000000
    000000000000900000006000000000000000000000000000000000000000000000000000
100 **************************##########################********************
 90 **************************##########################********************
 80 **************************###########################*******************
 70 *************************############################*******************
 60 *************************############################*******************
 50 ******#*****************################################################
 40 ******#****************#################################################
 30 ###*###*####***********#################################################
 20 ########################################################################
 10 ########################################################################
    0....5....1....1....2....2....3....3....4....4....5....5....6....6....7.
              0    5    0    5    0    5    0    5    0    5    0    5    0

                   CPU% per hour (last 72 hours)
                  * = maximum CPU%   # = average CPU%

msdc-spine-r1# show system resources module all
CPU Resources:
-----------------------------------------------------------
  CPU utilization:  Module  5 seconds  1 minute  5 minutes
-----------------------------------------------------------
                       1        25        15         14
                       2        21        15         15
                       3        26        23         21
                       4        14        14         14
                       5        21        15         14
                       6        11        13         13
                       7        11        13         13
                       8        11        12         12
                      10        27        18         19
                      11        23        13         12
                      12        17        11         12
                      13        10        13         12
                      14        10        13         13
                      15        11        12         13
                      16        11        12         12
                      17        11        13         13

-----------------------------------------------------------
  Processor memory:   Module   Total(KB)   Free(KB)  % Used
-----------------------------------------------------------
                         1      2075900    1339944      35
                         2      2075900    1340236      35
                         3      2075900    1333976      35
                         4      2075900    1339780      35
```

```
                        5      2075900    1341112      35
                        6      2075900    1344648      35
                        7      2075900    1344492      35
                        8      2075900    1344312      35
                       10      8251592    6133856      25
                       11      2075900    1344604      35
                       12      2075900    1344904      35
                       13      2075900    1344496      35
                       14      2075900    1344496      35
                       15      2075900    1344808      35
                       16      2075900    1344416      35
                       17      2075900    1344536      35
msdc-spine-r1# 1344416      35
                       17      2075900    1344536      35
msdc-spine-r1#
```

URIB

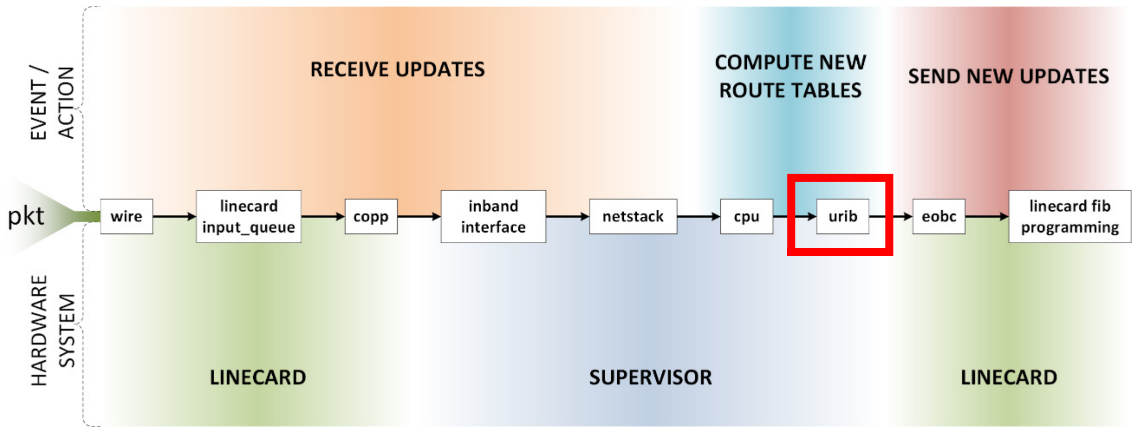
When there is a lot of network instability urib-redist can run out of shared memory waiting for acks caused by routing changes. urib-redist uses 1/8 of the memory allocated to urib, which can be increased by modifying the limit for 'limit-resource u4route-mem' (urib).

This data shows urib-redist with 12292 allocated, which is 1/8 of urib (98308)

```
n7k# show processes memory shared
Component           Shared Memory      Size             Used  Available    Ref
                          Address   (kbytes)         (kbytes)   (kbytes)  Count
smm                    0X50000000      1028                4       1024     41
cli                    0X50101000     40964*           25151      15813     12
npacl                  0X52902000        68                2         66      2
u6rib-ufdm             0X52913000       324*             188        136      2
u6rib                  0X52964000      2048+ (24580)      551       1497     11
urib                   0X54165000      7168+ (98308)     5161       2007     22
u6rib-notify           0X5A166000      3076*             795       2281     11
urib-redist            0X5A467000     12292*           11754        538     22
urib-ufdm              0X5B068000      2052*               0       2052      2
```

Protocols often express interest in notifications whenenever there is a change in the status of their own routes or routes of others (redistribution). Previously, no flow control in this notification mechanism existed, that is, urib kept sending notifications to protocols without checking whether the protocol was able to process the notifications or not. These notifications use shared memory buffers which may encounter situations where shared memory was exhausted. Part of this feature, urib will now allow only for a fixed number of unacknowledged buffers. Until these buffers are acknowledged additional notifications will not be sent.

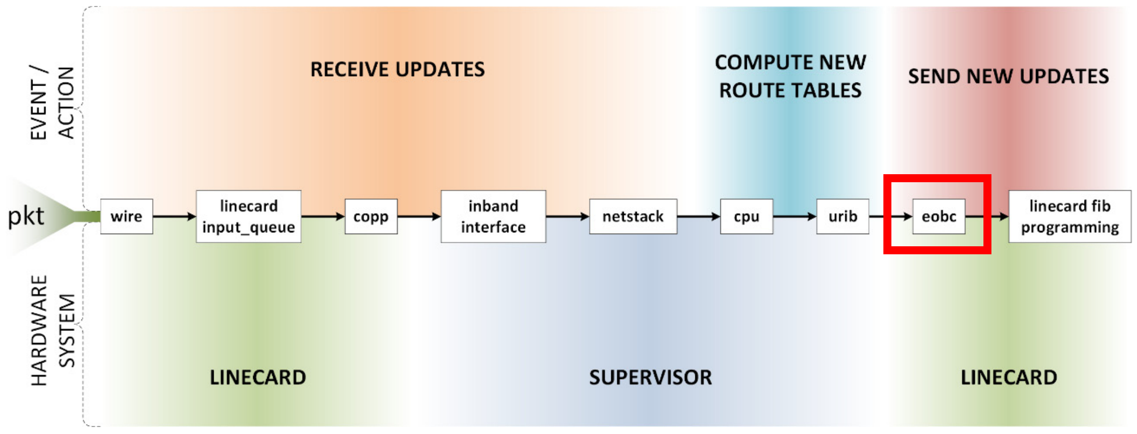
Figure 2-9 URIB Path



EOBC

Once a new FIB has been generated from the RIB, updates are sent to the forwarding engine on each linecard via the Ethernet Out of Band Channel (EOBC) interface on the supervisor. Many other internal system processes utilize the EOBC as well. As the level of network churn increases, it is expected the number of FIB updates increase. Thus it is expected an increase in RX and TX utilization on the EOBC interface to happen. Should this interface become overwhelmed, throttling will occur and packets will be dropped. This delays programming new entries into the forwarding engine, causing packet misrouting and increased convergence times.

Figure 2-10 EOBC Path



EOBC Commands

On the N7k, the EOBC rate limit for SUP1 is 16kpps, while the limit for SUP2 is significantly higher. The show hardware internal cpu-mac eobc stats command gives a vast array of statics regarding the EOBC interface. Statistics about throttling are specifically sought after. Seeing the Rate limit reached counter incrementing indicates the network churn rate is approaching meltdown.

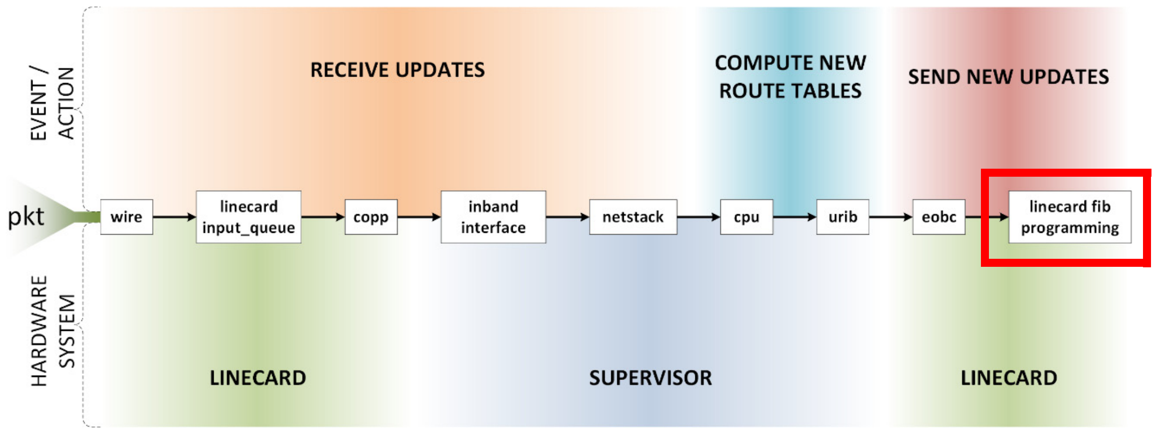
```
msdc-spine-r8# show hard int cpu-mac eobc stats | be Throttle | head
Throttle statistics
```

```
-----------------------------+---------
Throttle interval ........... 3 * 100ms
Packet rate limit ........... 16000 pps
Rate limit reached counter .. 0
Tick counter ................ 6661123
Active ...................... 0
Rx packet rate (current/max)  30 / 6691 pps
Tx packet rate (current/max)  28 / 7581 pps
```

Linecard FIB Programming

Each linecard and platform has its own programming algorithms for its forwarding engines. The architecture of each is different, so the specifics won't be covered here.[5]

Figure 2-11 Linecard FIB Programming



OSPF

Open Shortest Path First (OSPF) testing focused around control plane scale at a real MSDC customer network, herein to be referred as ACME_1[6]. ACME_1 has an OSPF network that runs at a higher scale than Cisco originally published for the N7K platform as supported, and is growing at a rapid pace.

This testing verification ensures Nexus 7000 capabilities of handling ACME_1s specific scenario.

This version of ACME_1 testing includes the following primary technology areas:

- OSPF Scale
- Unicast Traffic
- ECMP

DDTS caveats discovered and/or encountered in this initial testing effort are identified in the "Defects Enountered" section of the external test results document.[7]

5. Refer to Appendix C, "F2/Clipper Linecard Architecture,"
6. To protect the names of the innocent, as well as comply with MNDA requirements, ACME_1 will be used. If other real MSDC customers are referred to in this document, they will be notated as "ACME_2", "ACME_3", etc.
7. For a detailed discussion of testing results, please refer to the document "Cisco ACME_1 Control Plane Scale Testing, Phase 1 Test Results". This guide is intended to provide a summary only of overall considerations.

Table 2-1shows project scale number for OSPF scale parameters.

Table 2-1 Project Scale Number for OSPF Scale Parameters

OSPF Scale Parameters	Value
Area 0 Type-1 LSA	>1000
Type-5 External	20,000->30,000
Neighbors	~45

All routing protocols are susceptible to scale limitation in the number of routes in the table and the number of peers to which they are connected. Link state protocols like OSPF are also susceptible to limitations in the number of routers and links within each area. The ACME_1 topology pushes all these limits, as is typical of most MSDC customers.

Summary of Test plan

OSPF Scale testing focused on 7 major considerations in this phase:

1. OSPF Baselining
2. Type-5 LSA Rout Injections/Withdrawals
3. Domain Stability
4. External Influences on OSPF Domain Stability
5. Unicast Traffic Patterns
6. ECMP
7. BFD

Each test group (test set) had a series of individual tests. The reader may refer to a subsequent document detailing all tests and results upon request.

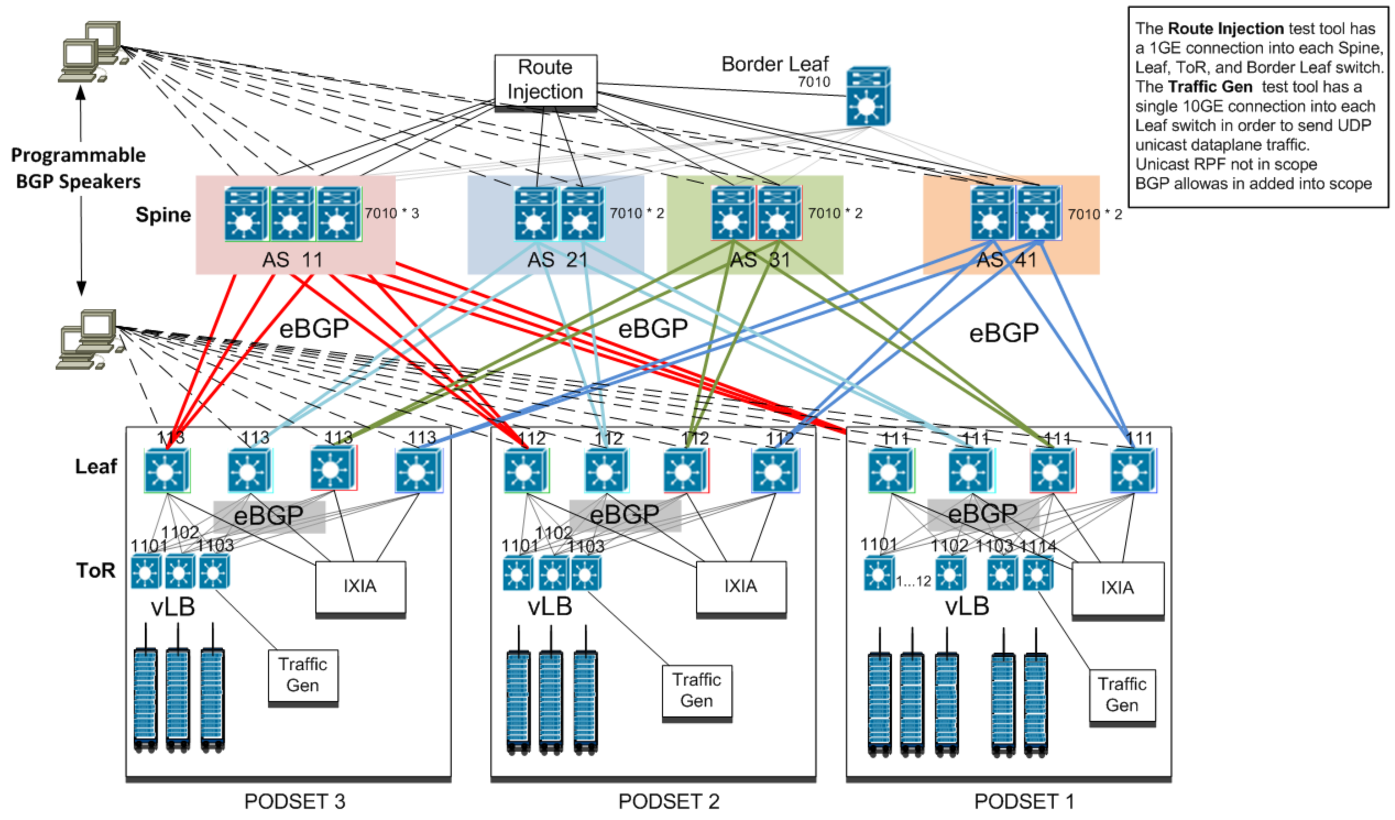
Summary of Results

OSPF testing results demonstrated that the network remains stable up to 30k LSAs, and can scale to 60k LSAs if BFD is enabled. OSPF and OSPF with BFD enabled showed some instability in a few instances with steady-state flaps and LSA propagation delays; however, both those issues are addressed in NX-OS 6.2.

BGP

Another MSDC customer, ACME_2, was selected to examine alternative BGP arrangements for increasing scale of an MSDC without compromising convergence. Both resiliency and reliability were also top concerns needing attention, and are discussed below. The test topology was not a straightforward three-stage Clos, but rather closer to a “reduced” five-stage Clos with multiple Spine “networks”, never the less, the same high-level topological principles apply (Figure 2-12). It was run within the test topology.

Figure 2-12 BGP Testing: Resilliency and Reliability


Programmable
BGP Speakers
Route
Injection
Border Leaf
7010
The Route Injection test tool has a 1GE connection into each Spine, Leaf, ToR, and Border Leaf switch.
The Traffic Gen test tool has a single 10GE connection into each Leaf switch in order to send UDP unicast dataplane traffic.
Unicast RPF not in scope
BGP allowas in added into scope
Spine
7010 * 3
AS 11
7010 * 2
AS 21
7010 * 2
AS 31
7010 * 2
AS 41
eBGP
eBGP
eBGP
Leaf
113
112
111
ToR
eBGP
1101
1102
1103
1114
1...12
vLB
IXIA
Traffic
Gen
PODSET 3
PODSET 2
PODSET 1

The system was composed of 3 physical Podsets[8], Podsets 1, 2 and 3. Each Podset consisted of 4 Nexus 3064 Leaf nodes and a mixture of Nexus 3064/3048 ToRs. Podset 1 had over a dozen TORs while Podset 2 and 3 had 3 ToRs. IXIA IXNetwork was used to bring the total number of real and simulatied ToRs to 17 for each Podset. Route-maps were configured on each ToR to advertise four /24 directly connected prefixes. A 300x VM Hadoop cluster was also connected to Podset 1 (also used for TCP incast and buffer utilization testing). Each VM connected to the ToR via a /30 connected subnet, configured through DCHP.

Note /30 masks were used to provide location awareness for Hadoop nodes.

Based on the DHCP forwarding address, backend servers map requests to specific racks, and position in the rack. Inband management was used for the Hadoop cluster, out of band was utilized for network devices. Each Leaf node connected to a single Spine. Depending on the Leaf node there were either two or three parallel connections to the Spine layer (ACME_2 requirement). IXNetwork was used to simulate up to 32 BGP spine sessions for each Leaf node.

Scaling was done to 140 POD sets at the Spine layer using combinations of real and simulated equipment. Each Spine node connected three non-simulated Leaf nodes, and the remaining nodes, 137 of them, were simulated using IXIA. All Leafs advertised 68 /24 ipv4 prefixes to each Spine node, and each Spine node received over 9000 BGP prefixes, in total, from the Leaf layer.

8. A Podset would be comprised of hundreds of servers. ToRs for each rack were N3064s. Pod sets connect to an infrastructure based on the three-stage Clos topology. For the purposes of testing, a smaller-scale version of the customer has in production was used.

With the exception of the programmable BGP Speakers (pBS), BFD was enabled across the topology for each BGP session. BFD is enabled for all ToR <-> Leaf, Leaf <-> Spine, and Spine <-> Border connections.

pBSes were simulated using IXIA. Each Spine and Leaf node peered with a pBS. There were 32 BGP sessions with the pBS, per device, broken down into two groups, with each group consisting of sixteen BGP sessions. All 32 BGP sessions advertised hundreds of /32 VIPs used for service loadbalancing to the server. For all VIPS advertised, Group1 advertises prefix with MED 100 while Group 2 advertised MED 200. Each VIP had 16 equal cost paths in the route table; NH reachability for all VIPs point to the physical IP address of the load balancer(s).

To reach the final goal of 16,000 IPV4 prefixes, IXIA injected 4700 prefixes at the Border Leaf layer. Nexus 3000 limits the route size to 8K in hardware if uRPF is enabled (default). To get to the target of 16K routes, urpf had to be disabled on Leaf and ToR nodes.

Two types of traffic were used in testing:

1. Background server-to-server traffic
   a. Podset 2 <-> Podset 1
   b. Podset 3 <-> Podset 1
   c. Podset 3 <-> Podset 2
2. VIP traffic from servers to loadbalancers
   a. Podset 2 -> VIP
   b. Podset 1 -> VIP
   c. Podset 3 -> VIP

With the entire system configured as outlined above, these were the 3 major test sets executed:

1. Baseline tests
2. Route Convergence
3. Multi-Factor Reliability

Note Test sets are defined as a broad characterization of individual tests; in other words, Test set 1 had 17 individual tests (BGP steady state with and without churn, BGP soft clearing, Link Flapping, ECMP path addition and reduction, etc), Test set 2 had 7, Test set 3 had 6.

Summary of Results

All platforms must be considered when examining routing scale limits. For the N7K[9]; 2 session limits exist when running BGP with and without BFD. BFD is limited to 200 sessions per module, and 1000 sessions were supported per system. For BGP, 1000 neighbors per system were supported. Limits for N3K were less than N7K.

Observations

- Peering at both Spine and Leaf provides greater granularity of available hardware loadbalancing. However, peering at the Spine, requires customizing route-maps to change next-hop which is less scalable.

4. SDU validated these numbers in testing:

9. http://www.cisco.com/en/US/docs/switches/datacenter/sw/verified_scalability/b_Cisco_Nexus_7000_Series_NX-OS_Verified_Scalability_Guide.html#concept_2CDBB777A06146FA934560D7CDA37525

- The overall test topology as a whole:
  - **N7K**—Up to 128 sessions of BGP+BFD were validated per linecard. Note: BGP Updates do not terminate on the linecard, unlike BFD sessions. Thus the 128 session limit is what BFD could do. Per system, tests were scaled to 768 sessions (768 IXIA sessions + 12 real sessions). All were run rith BFD at 500ms timers.
  - **N3K**—16 BGP sessions on leaf-r1, the remaining Leafs at 8 sessions.

5. Convergence with BGP (w/ BFD enabled) was well below the 10 second target.
6. Convergence with BGP alone (without BFD) did not converge under the targeted 10 seconds.
7. FIB overflow can cause inconsistency or unpredictable convergence. It should be avoided if possible or worked around. This is due to new entries learned after FIB exhaustion that would be otherwise forced to software route. Once mapped in software these would never reprogram back into the FIB, unless they were lost and relearned. The workaround is to clear all IP routes, forcing a TCAM reload/reprogram. This workaround causes temporary neighbor-loss with BFD configured (when we used 500/3 timers). This workaround can be done manually or through an EEM script, like this:

```
event manager applet fib-exception
  event syslog pattern "<put-to-FIB-exception-gone-syslog>"
  action 1.0 cli clear ip route *
  action 1.1 syslog msg FIB Re-downloaded to HW
```

   Features are available in IOS-XR which would benefit NX-OS development, which address FIB issues encountered above.
8. FIB and MAC tables are not coupled. Recommendation is to configure identical aging timer to maintain synchronization. Options are; either increase MAC aging or decrease ARP aging. Primarily applies to unidirectional flow.
9. If BFD is implemented in the network, BFD echo packets needs to be assigned to priority queue to ensure network stability under load.
10. URPF must be disabled to support 16K routes in hardware on the N3K.
11. To work around an ECMP polarization issue, hashing algorithms must be different between ToR and Leaf layers. A new CLI command was created to configure different hash offsets to avoid the ECMP polarization.

Refer to subsequent testing documentation for complete details about ACME_2 testing.

BFD

Bidirectional Forwarding Detection (BFD), a fast failure detection technology, was found to allow for relaxed routing protocol timers. This in turn creates room for scaling routing protocols.

Summary of Results

BFD testing occurred between test instrumentation hardware and the Spine. 384 sessions were validated at the spine with both BGP and OSPF. A 500ms interval was configured based on overall system considerations for other LC specific processes.

Incast Simulation and Conclusions

Since SDU-MSDC's objective was provide meaningful network architecture guidance in this space, it is necessary to simulate as close to the real thing as possible. This presents difficulties in MSDC space because of the sheer volume of servers (endpoints, or nodes) that are required to make the problem appear in the first place.

Servers

Servers are distributed throughout the fabric with 10G connectivity. Refer to Server and Network Specifications, page A-1 for server specifications, configurations, and Hadoop applications details.

Intel recommends the following based on real world applications:
http://www.intel.com/content/dam/doc/application-note/82575-82576-82598-82599-ethernet-controllers-interrupts-appl-note.pdf

Note File transfer buffering behaviors were observed – kernel controls how frequently data is dumped from cache; with default kernel settings, the kernel wasn't committing all memory available, thus there was a difference between committed memory vs. what it's able to burst up to. As a result, VMs that hadn't committed everything behaved worse than those that did. To keep all experiments consistent, all VMs were configured to have all memory 100% "committed".

TCP receive buffers were configured at 32MB. It was set higher because the goal was to remove receive window size as a potential limitation on throughput and to completely rely on CWND. This is not realistic for a production deployment, but it made tracking key dependencies easier. Refer to Incast Utility Scripts, IXIA Config, page E-1 for relevant sysctl.conf items.

The formula for TCP receive window is:

$$\frac{tcp_{rmem} Bytes}{2^{tcp\_adv\_win\_scale}}$$

Below shows TCP RX window is set correctly:

```
[root@r09-p02-vm01 tmp]# more /proc/sys/net/ipv4/tcp_adv_win_scale
2
```

Based on theeformula, 75% of buffer size is used for TCP receive window (25MB window scale factor 10). This value is never reached as CWND is always the limiting factor.

Note Regarding window size, as of linux kernel 2.6.19 and above, CUBIC is the standard implementation for congestion control.

Other TCP parameters were as follows:

- TCP selective ACK is enabled:

```
[root@r09-p02-vm01 ipv4]# more tcp_sack
1
```

- IP forward disabled:

```
[root@r09-p02-vm01 ipv4]# more ip_forward
0
```

- Misc settings:

```
[root@r09-p02-vm01 ipv4]# more tcp_congestion_control
Cubic
[root@r09-p02-vm01 ipv4]# more tcp_reordering
3
```

- RTT averaged 0.5ms as reported by ping.

All VMs were configured with 4 VCPU and 20G memory.   Since the Hadoop jobs were not CPU bound, one vcpu would have been sufficient. IO was the biggest bottleneck especially when less than 20G assigned and during cluster failure; hence moving to 20G masked that. For comparison purposes, to copy a 1G file from hdfs to local disk iowait peak was at 75% with 3G memory, barely over 1% @20G. This is because linux page cache relies on pdflush to write data of cache to disk, and this is nominally 30 seconds or 10% dirty pages. Depending on the type of job write interval can be tuned up or down, as required :

Note This link outlines additional issues to be aware of when hot plugging vcpu: https://bugzilla.redhat.com/show_bug.cgi?id=788562

To manage failures and their impact to Incast events, two scripts were written to track the status of a job: “fail-mapper.sh” and “find-reducer.sh”. fail-mapper.sh reloads 15% of the VMs immediately before the reduce phase, and find-reducer.sh launches tcpdump on the reducer. Tcpdump output was used to analyze TCP windowing behavior during Incast events.

Following logic was implemented in fail-mapper.sh:

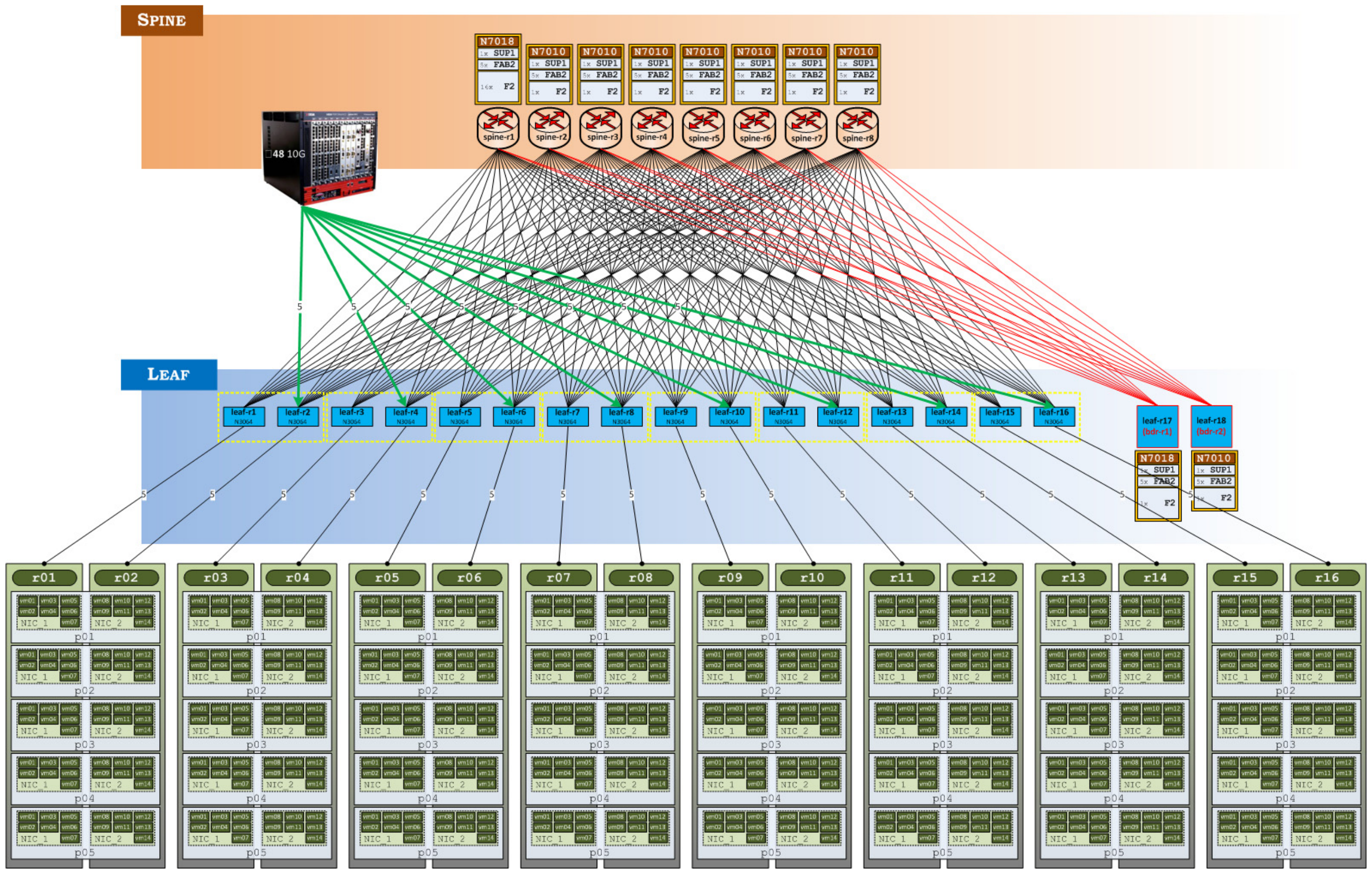
1. User inputs two job ids (example 0051, 0052)
2. Query each map task and generate a unique list of VMs responsible for each job. There will be two lists generated, one per job.
3. Compare the two lists, generate a third list by suppress common VMs.
4. Query the job status, once map tasks reaches 100% completion (96% for cascading failure), reload 15% of the VMs based on #3.

Find-reducer.sh determines the location of the reducer and launches tcpdump.

Topology

Figure 2-13 shows a standard 3-stage folded Clos topology, with 8 Spines and 16 Leafs.

Figure 2-13 Incast Lab Setup



Note Physical servers are arranged in logical racks, numbered “r01-r16”. Even though a physical server spans two logical racks, it is the physical NICs (and the VMs mapped to them) that are actually assigned to a logical rack. For example, the first server shown in the top-leftmost position has NIC_1 which is “in” rack r01 and NIC_2 in r02.

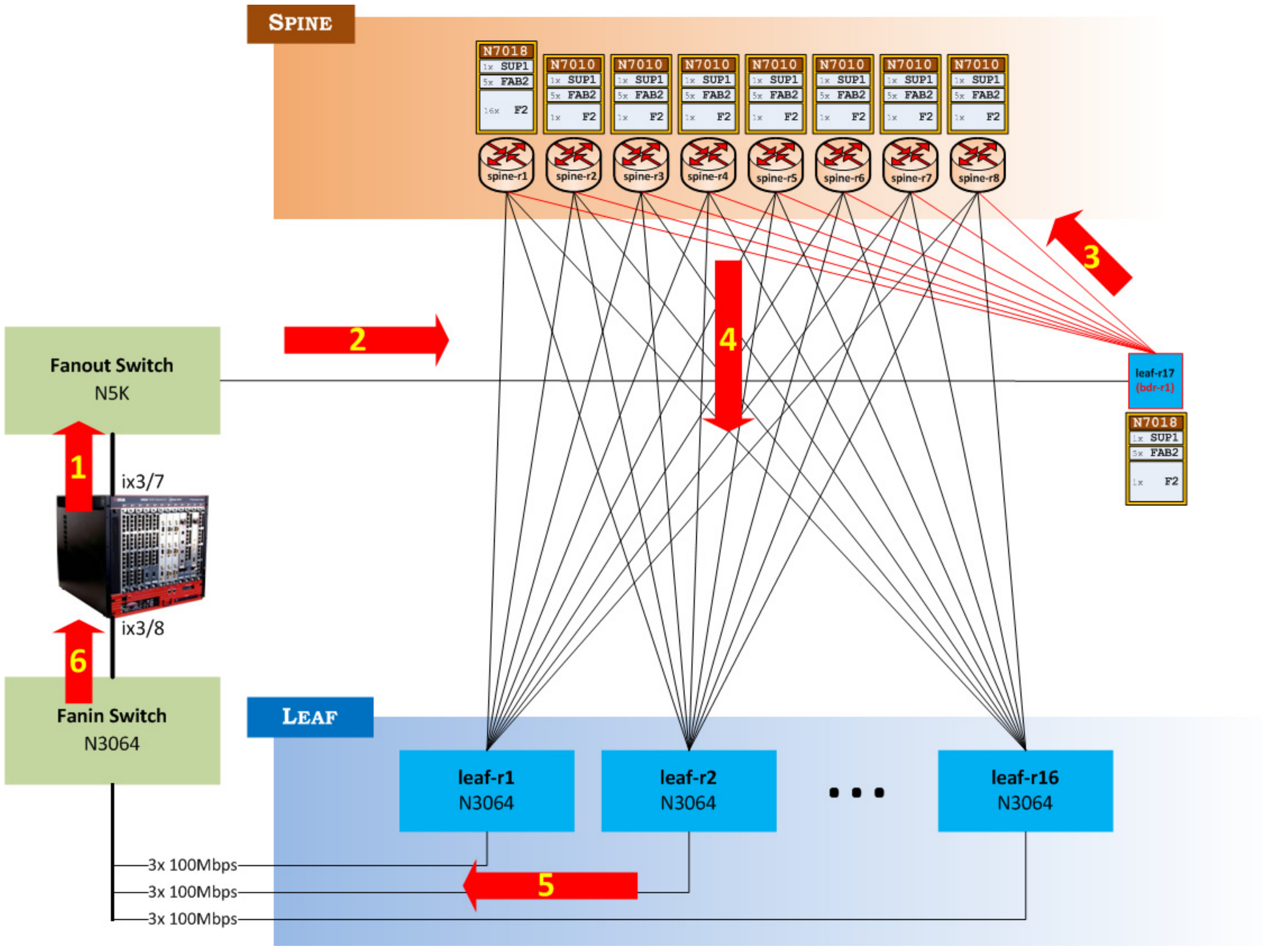
Initially, there was noise traffic sent to exhaust both “bandwidth” and “buffer utilization”, but it was determined exercising buffers was sufficient, along with Hadoop traffic, to create Incast events. For completeness, the “bandwidth utilization” noise floor traffic method is described in Bandwidth Utilization Noise Floor Traffic Generation, page F-1.

The border devices represent “external” networks and are injecting a default route, effectively acting as a sensor for spurious traffic.

Buffer Utilization

Figure 2-14 shows an IXIA shared buffer setup.

Figure 2-14 IXIA Shared Buffer Setup



The IXIA is connected to each Leaf indirectly, and using a series of oscillating traffic bursts, in conjunction with the bandwidth "noise" traffic above, both dedicated and shared buffers on the Leafs are consumed at will (oscillating traffic is needed because the IXIA wasn't able to consistently consume N3K buffers with steady-stream traffic). The purposes of sending traffic through the border leaf and to the Spines are two-fold:

1. IXIA didn't have enough 10G ports to connect to every Leaf.
2. Sending traffic via ECMP towards the Spine, and then the Spine downto the Leafs, simulates real traffic flow, albeit uni-directional (IXIA is both the source and sink).

In detail, this is how the IXIA is configured for shared buffer impairment traffic:

2x 10G interfaces, in total, are used to Send (Ix3/7) and Recv (Ix3/8) uni-directional UDP traffic. The source traffic comes into an N5K fanout switch (this switch held other experiments to the border, so it was left intact – technically, the IXIA could be connected directly to the border leaf, achieving the same result) to Border leaf-r1 (msdc-leaf-r17), which connects to Spines r1 – r8.

- Refer to the following example for Leaf dest IP 10.128.4.131:

```
msdc-leaf-r17# show ip route 10.128.4.131
IP Route Table for VRF "default"
'*' denotes best ucast next-hop
```

```
'**' denotes best mcast next-hop
'[x/y]' denotes [preference/metric]
'%<string>' in via output denotes VRF <string>

10.128.4.128/25, ubest/mbest: 8/0
    *via 10.1.1.32, [20/0], 8w0d, bgp-64617, external, tag 64512
    *via 10.2.1.32, [20/0], 8w0d, bgp-64617, external, tag 64512
    *via 10.3.1.32, [20/0], 8w0d, bgp-64617, external, tag 64512
    *via 10.4.1.32, [20/0], 8w0d, bgp-64617, external, tag 64512
    *via 10.5.1.32, [20/0], 8w0d, bgp-64617, external, tag 64512
    *via 10.6.1.32, [20/0], 8w0d, bgp-64617, external, tag 64512
    *via 10.7.1.32, [20/0], 8w0d, bgp-64617, external, tag 64512
    *via 10.8.1.32, [20/0], 8w0d, bgp-64617, external, tag 64512
```

- Traffic is sourced from the same IP (10.128.128.151), but there are 3 unique dest IP's for each leaf (msdc-leaf-r1-16), Vlans 11-13:

```
msdc-leaf-r1# show ip int brief
IP Interface Status for VRF "default"(1)
Interface            IP Address      Interface Status
Vlan11               10.128.4.129    protocol-up/link-up/admin-up
Vlan12               10.128.5.1      protocol-up/link-up/admin-up
Vlan13               10.128.6.1      protocol-up/link-up/admin-up

msdc-leaf-r2# show ip int brief
IP Interface Status for VRF "default"(1)
Interface            IP Address      Interface Status
Vlan11               10.128.8.129    protocol-up/link-up/admin-up
Vlan12               10.128.9.1      protocol-up/link-up/admin-up
Vlan13               10.128.10.1     protocol-up/link-up/admin-up
```

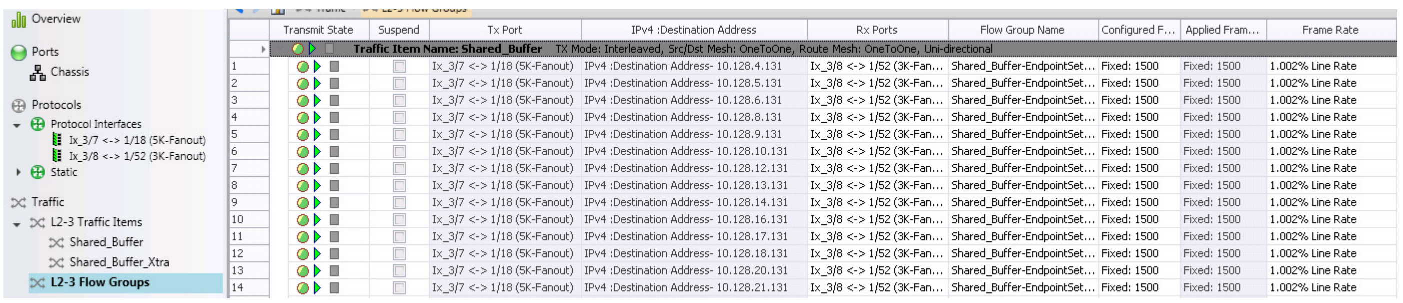
- All Leaf switches have 3x 100Mb links connected to an N3K fan-in switch, which connects to IXIA (Ix3/8):

```
msdc-leaf-r1# show cdp neighbors
Capability Codes: R - Router, T - Trans-Bridge, B - Source-Route-Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater,
                  V - VoIP-Phone, D - Remotely-Managed-Device,
                  s - Supports-STP-Dispute


Device-ID             Local Intrfce Hldtme Capability  Platform      Port ID
msdc-leaf-r42(FOC1550R05E)
                       Eth1/46       131    R S I s   N3K-C3048TP-1 Eth1/1
msdc-leaf-r42(FOC1550R05E)
                       Eth1/47       135    R S I s   N3K-C3048TP-1 Eth1/2
msdc-leaf-r42(FOC1550R05E)
                       Eth1/48       133    R S I s   N3K-C3048TP-1 Eth1/3
```

Two traffic items are configured:

1. Shared_Buffer
2. Shared_Buffer_Xtra

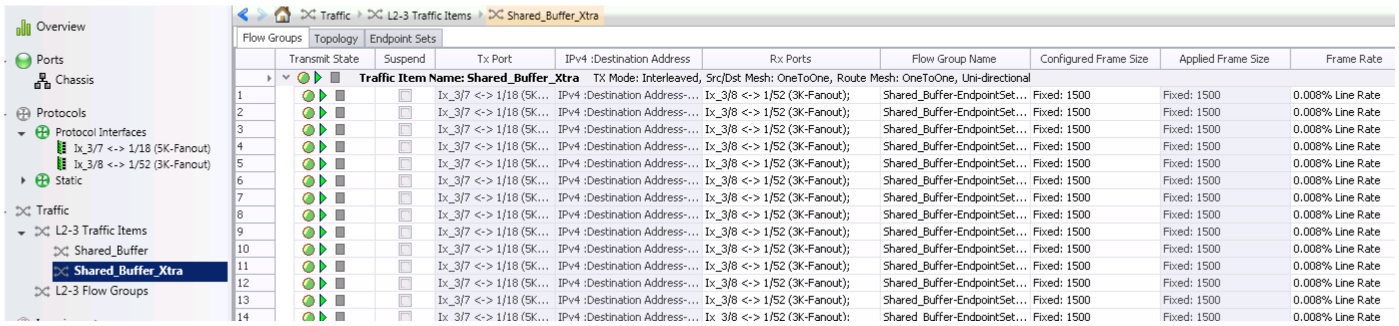
Shared_Buffer (Figure 2-15) has 48 endpoints that send UDP traffic unidirectional (3 streams to each leaf) at ~ 100Mb. This causes dedicated buffers to be consumed for that port, but does not dip into the system-wide shared buffer pool.



	Transmit State	Suspend	Tx Port	IPv4 :Destination Address	Rx Ports	Flow Group Name	Configured F…	Applied Fram…	Frame Rate
	Traffic Item Name: Shared_Buffer	TX Mode: Interleaved, Src/Dst Mesh: OneToOne, Route Mesh: OneToOne, Uni-directional							
1		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.4.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
2		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.5.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
3		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.6.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
4		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.8.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
5		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.9.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
6		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.10.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
7		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.12.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
8		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.13.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
9		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.14.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
10		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.16.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
11		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.17.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
12		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.18.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
13		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.20.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate
14		☐	Ix_3/7 <-> 1/18 (5K-Fanout)	IPv4 :Destination Address- 10.128.21.131	Ix_3/8 <-> 1/52 (3K-Fan…	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	1.002% Line Rate

Figure 2-15 IXIA Flows for Shared_Buffer

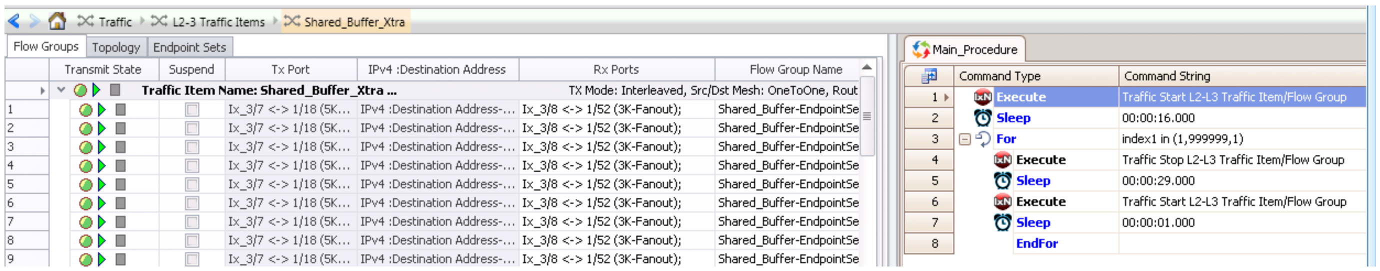
Shared_Buffer_Xtra (Figure 2-16) has the same 48 endpoints and traffic profile except that it sends traffic at ~ 800Kb.



	Transmit State	Suspend	Tx Port	IPv4 :Destination Address	Rx Ports	Flow Group Name	Configured Frame Size	Applied Frame Size	Frame Rate
	Traffic Item Name: Shared_Buffer_Xtra	TX Mode: Interleaved, Src/Dst Mesh: OneToOne, Route Mesh: OneToOne, Uni-directional							
1		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
2		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
3		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
4		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
5		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
6		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
7		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
8		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
9		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
10		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
11		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
12		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
13		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate
14		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSet…	Fixed: 1500	Fixed: 1500	0.008% Line Rate

Figure 2-16 IXIA Flows for Shared_Buffer_Xtra

This exceeds the interface throughput when combined with the first profile and starts to consume shared buffers. To achieve a shared buffer impairment without running out of buffers an IXIA script is used to stop and start the Xtra traffic stream, while the Shared_Buffer stream runs continuously (Figure 2-17).



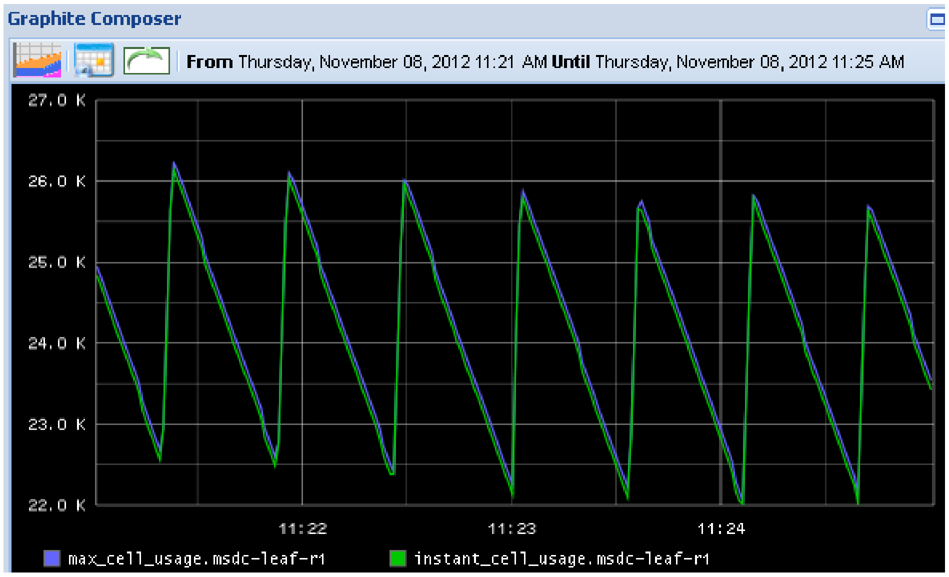
	Transmit State	Suspend	Tx Port	IPv4 :Destination Address	Rx Ports	Flow Group Name
	Traffic Item Name: Shared_Buffer_Xtra ...				TX Mode: Interleaved, Src/Dst Mesh: OneToOne, Rout	
1		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe
2		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe
3		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe
4		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe
5		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe
6		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe
7		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe
8		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe
9		☐	Ix_3/7 <-> 1/18 (5K…	IPv4 :Destination Address-…	Ix_3/8 <-> 1/52 (3K-Fanout);	Shared_Buffer-EndpointSe

	Command Type	Command String
1	Execute	Traffic Start L2-L3 Traffic Item/Flow Group
2	Sleep	00:00:16.000
3	For	index1 in (1,999999,1)
4	Execute	Traffic Stop L2-L3 Traffic Item/Flow Group
5	Sleep	00:00:29.000
6	Execute	Traffic Start L2-L3 Traffic Item/Flow Group
7	Sleep	00:00:01.000
8	EndFor	

Figure 2-17 IXIA Shared Buffer Impairment Timing

The timing of the script first loads up the shared buffers to ~8.5k for each of the 3 interfaces and then switches to a pattern where it alternates between bleeding off and increasing the buffer usage. This allows for a majority of the shared buffers to be used without exceeding the limit and dropping packets. The process forms a saw tooth pattern of usage shown in Figure 2-18.

Figure 2-18 IXIA Shared Buffer Impairment Traffic Oscillation

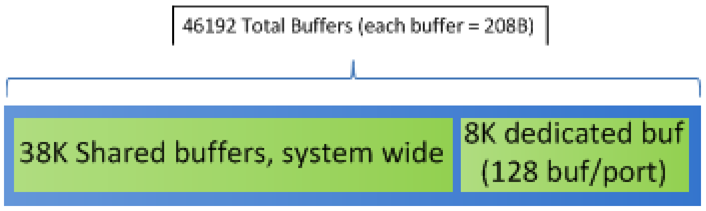


Buffer Allocation

Because the primary objective in these tests is to observe buffer behavior on the N3K Leaf layer, it must be ensured that dedicated buffers are consumed and shared buffer space is being exercised.

Figure 2-19 shows the overall schema of shared vs dedicated buffers on the N3K

Figure 2-19 N3K Buffers


46192 Total Buffers (each buffer = 208B)
38K Shared buffers, system wide
8K dedicated buf (128 buf/port)

This means the noise floor will consume all 128 dedicated buffers per port and has the capability of leeching into shared space, at will. With this control, Incast traffic can be pushed over the tipping point of consuming the remainder of available buffer space, i.e. – shared buffers, thus causing an Incast event. Table 2-2 shows how buffers are allocated system-wide.

Table 2-2 How Buffers are Carved Up on N3K

Reserved Memory	Physical Port	CPU Port	Loopback Port	Total MB
#	(For 3064) 64	1	1	
Queue/Port	15 (10+5)	48	5	
Total # of Qs	960	48	5	1013
Cells	7680	384	40	8104
Bytes	1597440	79872	8320	1685632
	Reserved	**Shared**	**Total**	
Cells	8104	37976	46080	
Bytes	1685632	7899088	9584640	(9.14 MB)

Note There is a defined admission control related to when shared buffer space is consumed by each port.

Admission control criteria are:

1. Queue Reserved space available
2. Queue dynamic limit not exceeded
3. Shared Buffer Space available

N3064-E imposes dynamic limits on a per queue basis for each port. The dynamic limit is controlled by the alpha parameter, which is set to 2. In dynamic mode, buffers allocated per interface cannot exceed the value based on this formula:

$$\frac{buffers}{interface\_threshold} \leq (\alpha)(num\_unused\_cells\_in\_buffer)$$

See N3K datasheets for a more detailed treatment of buffer admission control.

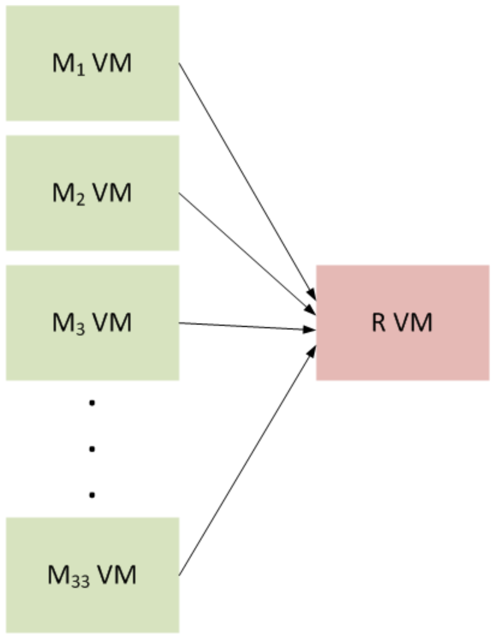
Monitoring

Standard Hadoop, Nagios, Graphite and Ganglia tools were used to monitor all VMs involved. Custom Python scripts, running on the native N3K Python interpreter, were created to monitor shared buffer usage.

Incast Event

Figure 2-20 shows a logical representation of the Incast event created.

Figure 2-20 Incast Event with M Mappers and R Reducers



Incast events were created by one of three methods:

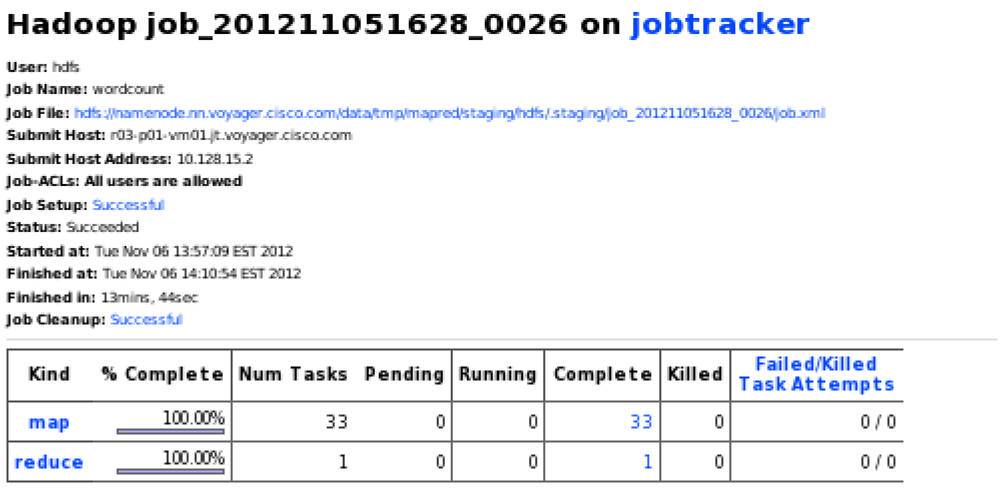
1. Fail x number of Mapper (M) VMs.
2. Fail y number of racks where M VMs live.
3. Fail z number of Leafs where M VMs live.

The test results in this section show two examples of a 33:1 Incast event created by inducing failures, as listed above, between the 33 M VMs to the 1 Reducer (R) VM: copying a 1GB file.

Note Actual locations of M or R VMs is determined by the Hadoop system when a job is created, thus monitoring scripts must first query for the locations before executing their code.

For the first example (Figure 2-21, Figure 2-22), two Hadoop jobs were executed: _0026 and _0027. Job 26 was tracked, and when the Map phase reached 96% of completion a script would kill 15% of the Map nodes only used in job 27. This would force failures on that particular job and cause block replication (data xfer) throughout the network. This was an attempt to introduce a cascading failure. However, it did not occur – Job 26 experienced the expected incast event, but no additional failure events were seen. Though numerous errors due to force-failed datanodes were observed in Job27, it too completed once it was able to recover after the Incast event.

Figure 2-21 33 Mappers to 1 Reducer

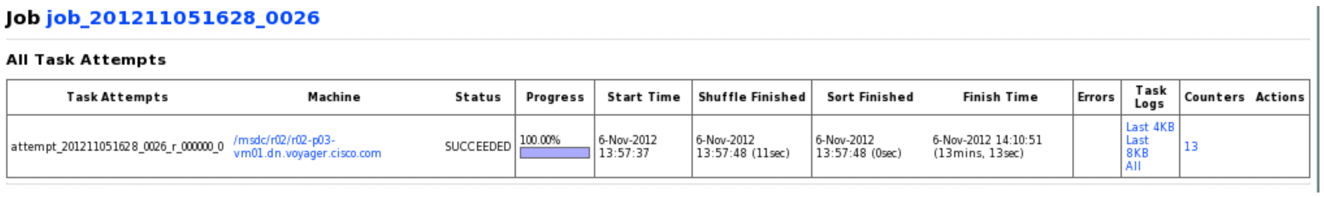


Hadoop job_201211051628_0026 on jobtracker

User: hdfs
Job Name: wordcount
Job File: hdfs://namenode.nn.voyager.cisco.com/data/tmp/mapred/staging/hdfs/.staging/job_201211051628_0026/job.xml
Submit Host: r03-p01-vm01.jt.voyager.cisco.com
Submit Host Address: 10.128.15.2
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Tue Nov 06 13:57:09 EST 2012
Finished at: Tue Nov 06 14:10:54 EST 2012
Finished in: 13mins, 44sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	33	0	0	33	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

Figure 2-22 Task Ran to Completion Once it Recovered



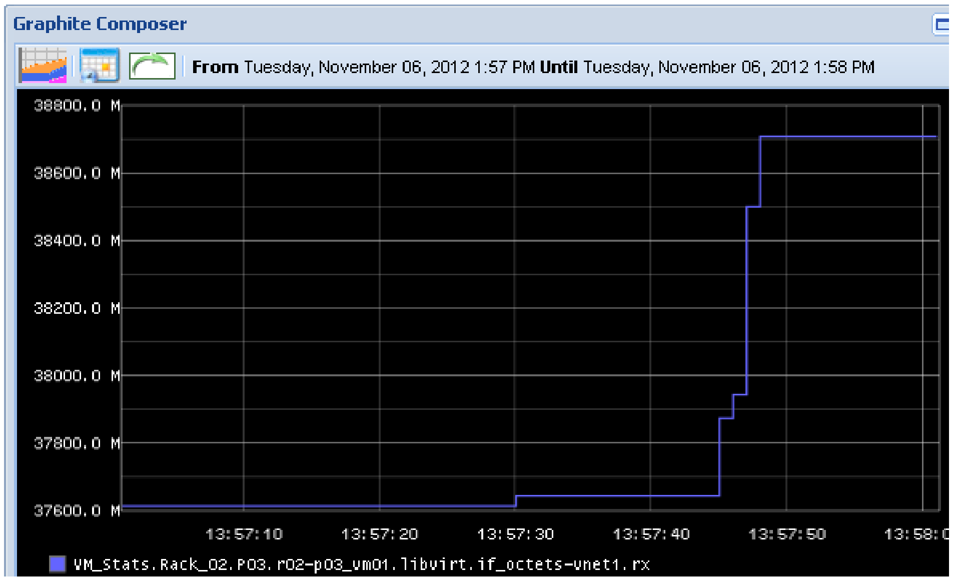
Job job_201211051628_0026

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Shuffle Finished	Sort Finished	Finish Time	Errors	Task Logs	Counters	Actions
attempt_201211051628_0026_r_000000_0	/msdc/r02/r02-p03-vm01.dn.voyager.cisco.com	SUCCEEDED	100.00%	6-Nov-2012 13:57:37	6-Nov-2012 13:57:48 (11sec)	6-Nov-2012 13:57:48 (0sec)	6-Nov-2012 14:10:51 (13mins, 13sec)		Last 4KB Last 8KB All	13	

The Reduce Copy phase is when the reducer requests all Map data in order to sort and merge the resulting data to be written to the output directory. The Incast burst occurs during this 'Copy' phase, which occurs between the Start time and Shuffle Finished time (Figure 2-23). Due to tuning parameters used to maximize network throughput bursting, the 1GB data transfer completed within a few seconds during the time window of 11s.

Figure 2-23 Traffic Received from Perspective of Reducer



Interfaces on the Leaf switch which connects to servers are 1-33 – 37, map to r02-p0(1-5)_vm01, respectively, thus Leaf interfaces which connect to the Reducer is 1-35. Figure 2-24 shows packet loss seen by the switch interface during event. Because data points for packets dropped are plotted every 10s by Graphite, and reported every 1s by the switch, the time period is slightly skewed.

Figure 2-24 Packet Loss, as Seen by Leaf Device

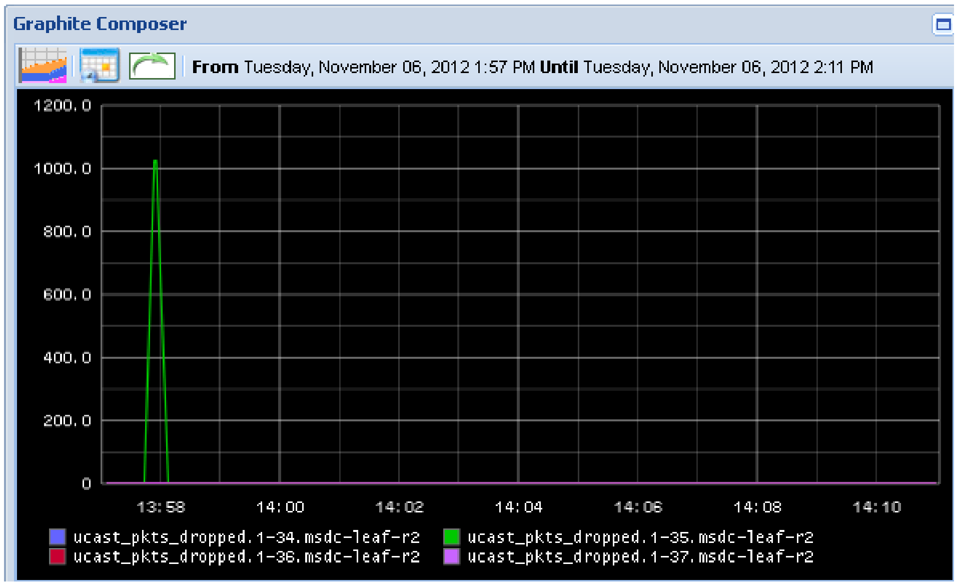


Figure 2-25 shows global instant cell usage and max cell usage, observed as the sharp burst in traffic, for the Reducer (Leaf-R2). The instant cell data point doesn't show up for this interface because the event occurs quickly then clears before the data point can be captured. However, max cell usage is persistent and reflects the traffic event.

Figure 2-25 Instant and Max Cell (Buffer) Usage, as Seen on the N3K

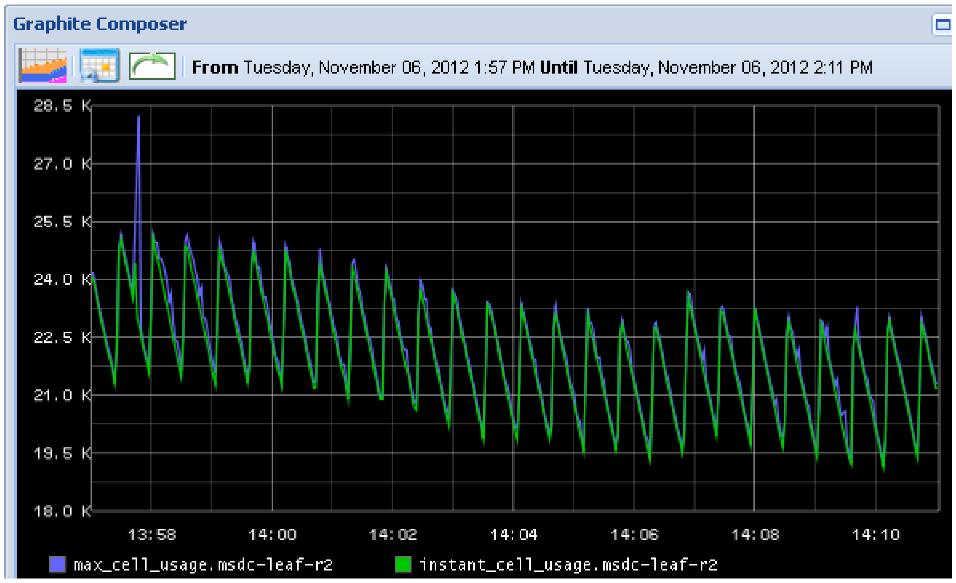
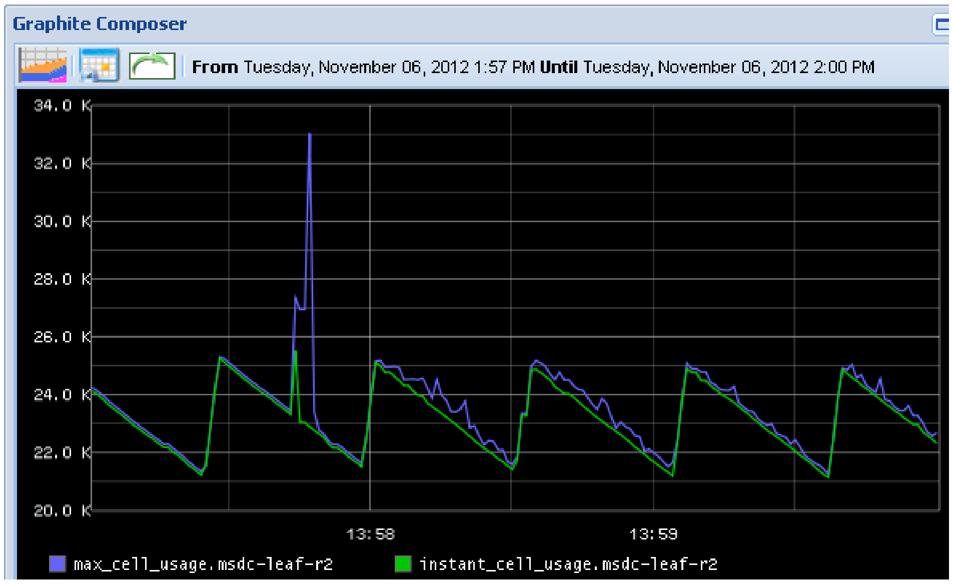


Figure 2-26 is a zoomed-in view of the spike. The additional spiking after the event is due to block replication that occurs from the force-failed VMs.

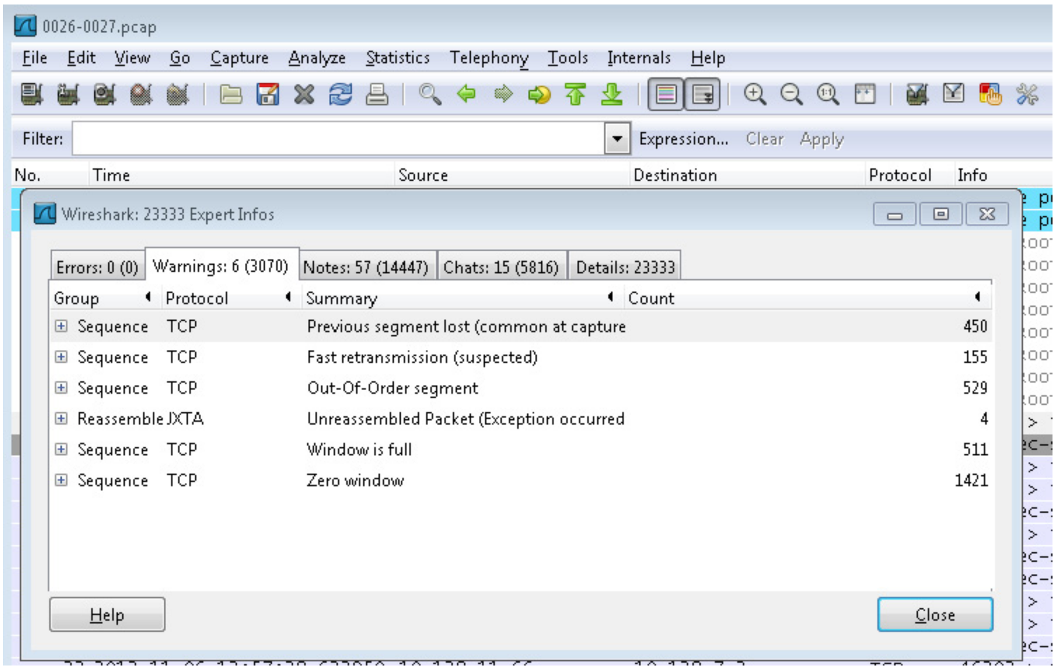
Figure 2-26 Max Cell Usage Zoom on the Spike



The reason why the spike didn't use all 37976 shared buffers available on the N3K system is because of buffer admission control – cannot exceed 2x available buffer per interface.

Lastly, for Job26, Figure 2-27 shows a Wireshark Expert Analysis of this job from a trace taken on the Reducer. Throughput collapse is evidenced by “Zero window” parameter (this means the TCP connection has a window-size of 0 and no payload can be transmitted/acknowledged); after which TCP slow-start mechanism kicks in.

Figure 2-27 *TCP Statistics*

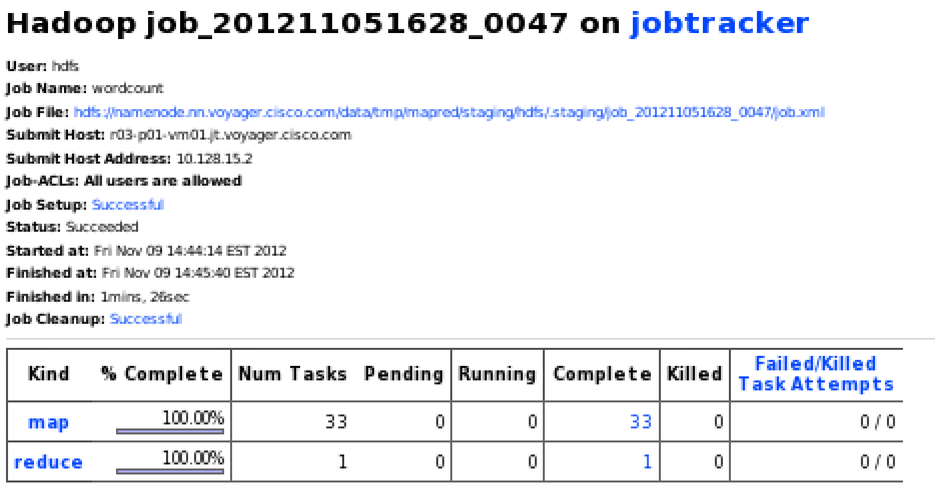

0026-0027.pcap
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Info
Wireshark: 23333 Expert Infos
Errors: 0 (0) Warnings: 6 (3070) Notes: 57 (14447) Chats: 15 (5816) Details: 23333
Help
Close

Group	Protocol	Summary	Count
Sequence	TCP	Previous segment lost (common at capture	450
Sequence	TCP	Fast retransmission (suspected)	155
Sequence	TCP	Out-Of-Order segment	529
Reassemble	JXTA	Unreassembled Packet (Exception occurred	4
Sequence	TCP	Window is full	511
Sequence	TCP	Zero window	1421

The second example is Job47 (Figure 2-28, Figure 2-29), which looks similar to Job26, but there is an additional comparison to the Control at the end. As before, there are 33 Mappers and 1 Reducer. One Hadoop job was launched with the IXIA shared buffer impairment running without any force failures. The Reduce copy phase produced a spike causing drops and degradation.

Due to the tuning parameters used to maximize network throughput bursting the 1GB data transfer was complete within a few seconds during the time window of 12s.

Figure 2-28 *Job47: 33 Mappers and 1 Reducer*

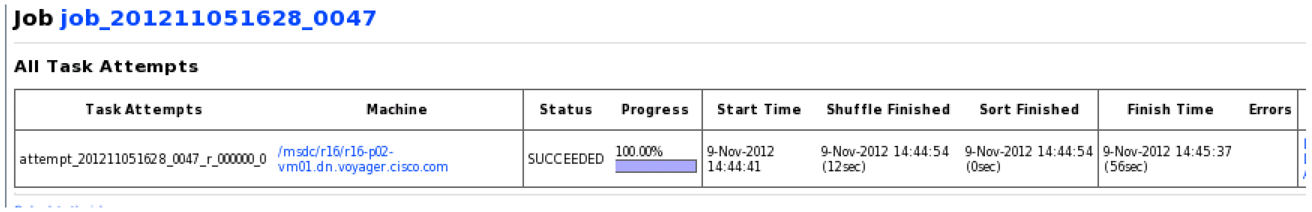


Hadoop job_201211051628_0047 on jobtracker

User: hdfs
Job Name: wordcount
Job File: hdfs://namenode.nn.voyager.cisco.com/data/tmp/mapred/staging/hdfs/.staging/job_201211051628_0047/job.xml
Submit Host: r03-p01-vm01.jt.voyager.cisco.com
Submit Host Address: 10.128.15.2
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Fri Nov 09 14:44:14 EST 2012
Finished at: Fri Nov 09 14:45:40 EST 2012
Finished in: 1mins, 26sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	33	0	0	33	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

Figure 2-29 Completed Successfully After it Recovered From the Incast Event



Job job_201211051628_0047

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Shuffle Finished	Sort Finished	Finish Time	Errors
attempt_201211051628_0047_r_000000_0	/msdc/r16/r16-p02-vm01.dn.voyager.cisco.com	SUCCEEDED	100.00%	9-Nov-2012 14:44:41	9-Nov-2012 14:44:54 (12sec)	9-Nov-2012 14:44:54 (0sec)	9-Nov-2012 14:45:37 (56sec)	

As with Job26, the burst received by Reducer (r16-p02_vm01) is seen in Figure 2-30:

Figure 2-30 Traffic Burst to the Reducer

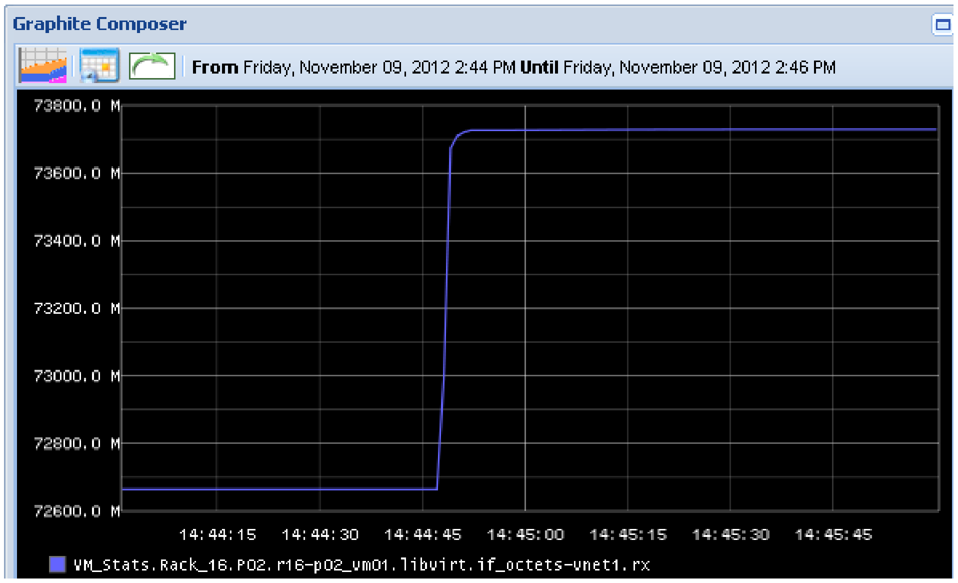


Figure 2-31 shows packet loss for the Incast event.

Figure 2-31 Packet Loss for Job47 During Incast Event

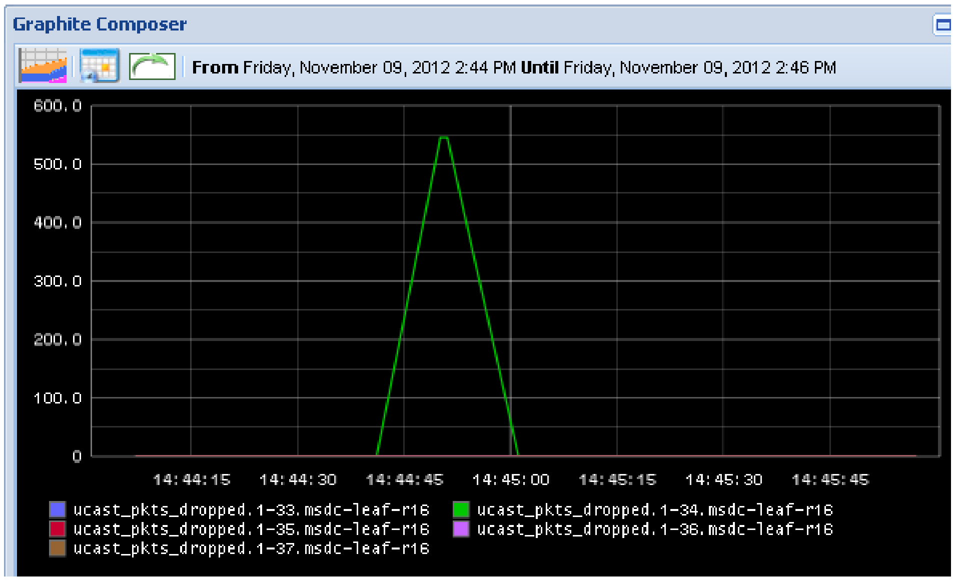
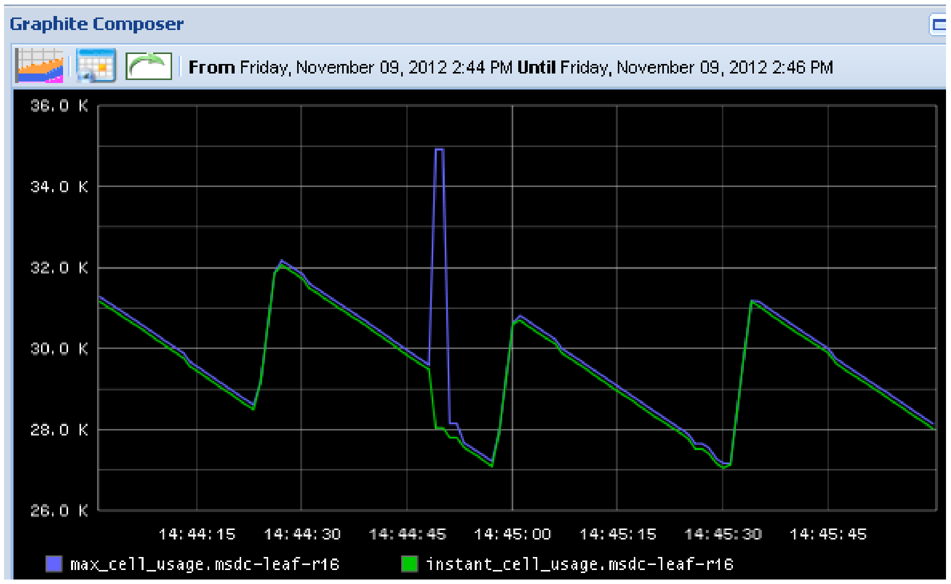


Figure 2-32 shows instant and max cell (buffer) usage.

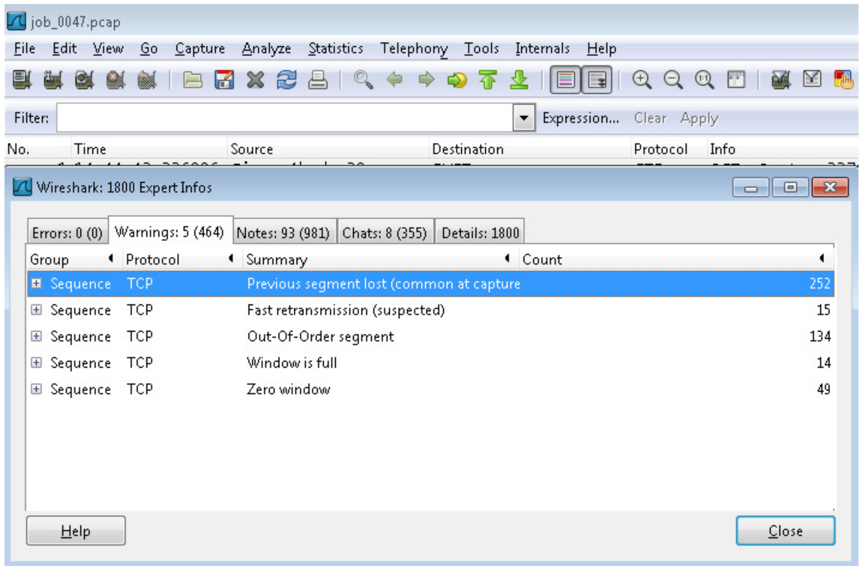
Figure 2-32 Zoom In on Spike in Max Cell Usage



Note Detailed analysis that follows is based on TCP sessions which contribute to the overall whole of the Hadoop job.

Figure 2-33 shows TCP connection stats throughput collapse.

Figure 2-33 TCP Stats as Reported by Wireshark of Packet Capture File


job_0047.pcap
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Info
Wireshark: 1800 Expert Infos
Errors: 0 (0) Warnings: 5 (464) Notes: 93 (981) Chats: 8 (355) Details: 1800
Help
Close

Group	Protocol	Summary	Count
Sequence	TCP	Previous segment lost (common at capture	252
Sequence	TCP	Fast retransmission (suspected)	15
Sequence	TCP	Out-Of-Order segment	134
Sequence	TCP	Window is full	14
Sequence	TCP	Zero window	49

The following configuration is a parsed tcptrace CLI output on VMs, with important metrics highlighted:

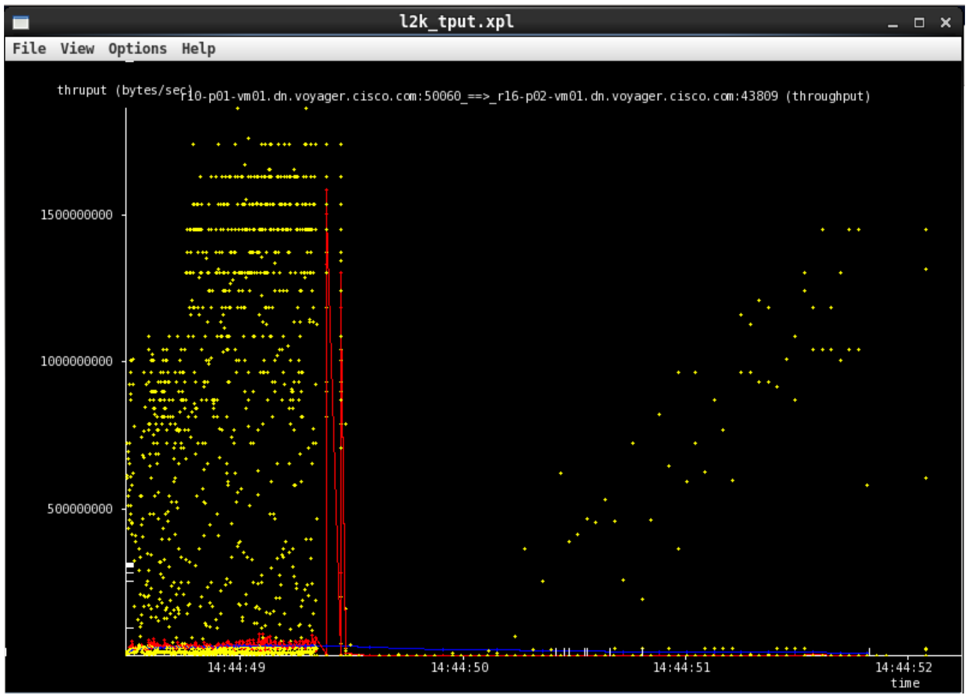
```
TCP connection 6:
        host k:        r16-p02-vm01.dn.voyager.cisco.com:43809
        host l:        r10-p01-vm01.dn.voyager.cisco.com:50060
        complete conn: yes
        first packet:  Fri Nov  9 14:44:48.479320 2012
        last packet:   Fri Nov  9 14:45:02.922288 2012
        elapsed time:  0:00:14.442968
        total packets: 3107
        filename:      job_0047.pcap
   k->l:                              l->k:
     total packets:         1476        total packets:         1631
     ack pkts sent:         1475        ack pkts sent:         1631
     pure acks sent:        1473        pure acks sent:           1
     sack pkts sent:          40        sack pkts sent:           0
     dsack pkts sent:          0        dsack pkts sent:          0
     max sack blks/ack:        1        max sack blks/ack:        0
     unique bytes sent:      302        unique bytes sent:  33119860
     actual data pkts:         1        actual data pkts:      1628
     actual data bytes:      302        actual data bytes:  33158956
     rexmt data pkts:          0        rexmt data pkts:          5
     rexmt data bytes:         0        rexmt data bytes:     39096
     zwnd probe pkts:          0        zwnd probe pkts:          0
     zwnd probe bytes:         0        zwnd probe bytes:         0
     outoforder pkts:          0        outoforder pkts:          0
     pushed data pkts:         1        pushed data pkts:        60
     SYN/FIN pkts sent:      1/1        SYN/FIN pkts sent:      1/1
     req 1323 ws/ts:         Y/Y        req 1323 ws/ts:         Y/Y
     adv wind scale:          10        adv wind scale:          10
     req sack:                 Y        req sack:                 Y
     sacks sent:              40        sacks sent:               0
     urgent data pkts:         0 pkts   urgent data pkts:         0 pkts
     urgent data bytes:        0 bytes  urgent data bytes:        0 bytes
     mss requested:         1460 bytes  mss requested:         1460 bytes
     max segm size:          302 bytes  max segm size:        26064 bytes
     min segm size:          302 bytes  min segm size:         1448 bytes
     avg segm size:          301 bytes  avg segm size:        20367 bytes
     max win adv:        3950592 bytes  max win adv:          16384 bytes
     min win adv:           1024 bytes  min win adv:          16384 bytes
```

```
zero win adv:              0 times     zero win adv:              0 times
avg win adv:         1953866 bytes     avg win adv:           16384 bytes
max owin:                303 bytes     max owin:             983193 bytes
min non-zero owin:         1 bytes     min non-zero owin:         1 bytes
avg owin:                  1 bytes     avg owin:             414083 bytes
wavg owin:                 0 bytes     wavg owin:             59842 bytes
initial window:          302 bytes     initial window:        14480 bytes
initial window:            1 pkts      initial window:            1 pkts
ttl stream length:       302 bytes     ttl stream length:  33119860 bytes
missed data:               0 bytes     missed data:               0 bytes
truncated data:          246 bytes     truncated data:     33067788 bytes
truncated packets:         1 pkts      truncated packets:      1628 pkts
data xmit time:        0.000 secs      data xmit time:        3.594 secs
idletime max:        10728.5 ms        idletime max:        10842.2 ms
throughput:               21 Bps       throughput:          2293148 Bps

RTT samples:               3           RTT samples:            1426
RTT min:                 0.6 ms        RTT min:                 0.1 ms
RTT max:                 1.0 ms        RTT max:                64.5 ms
RTT avg:                 0.8 ms        RTT avg:                13.5 ms
RTT stdev:               0.2 ms        RTT stdev:               9.3 ms

RTT from 3WHS:           0.6 ms        RTT from 3WHS:           0.3 ms

RTT full_sz smpls:         2           RTT full_sz smpls:         2
RTT full_sz min:         0.6 ms        RTT full_sz min:         0.1 ms
RTT full_sz max:         0.9 ms        RTT full_sz max:         0.3 ms
RTT full_sz avg:         0.8 ms        RTT full_sz avg:         0.2 ms
RTT full_sz stdev:       0.0 ms        RTT full_sz stdev:       0.0 ms

post-loss acks:            0           post-loss acks:            0
segs cum acked:            0           segs cum acked:          199
duplicate acks:            0           duplicate acks:           36
triple dupacks:            0           triple dupacks:            1
max # retrans:             0           max # retrans:             4
min retr time:           0.0 ms        min retr time:           0.0 ms
max retr time:           0.0 ms        max retr time:          89.2 ms
avg retr time:           0.0 ms        avg retr time:          35.5 ms
sdv retr time:           0.0 ms        sdv retr time:          47.7 ms
```

Note the RTT was quite large, especially considering all VMs for these tests are in the same datacenter.

Figure 2-34 shows a scatterplot taken from raw tcptrace data as sampled on the Reducer – thoughput collapse and ensuring TCP slow-start are easily visible. Yellow dots are raw, instantaneous, throughput samples. Red line is the average throughput based on the past 10 samples. Blue line (difficult to see) is the average throughput up to that point in the lifetime of the TCP connection.

Figure 2-34 Scatterplot of TCP Throughput (y-axis) vs Time (x-axis)



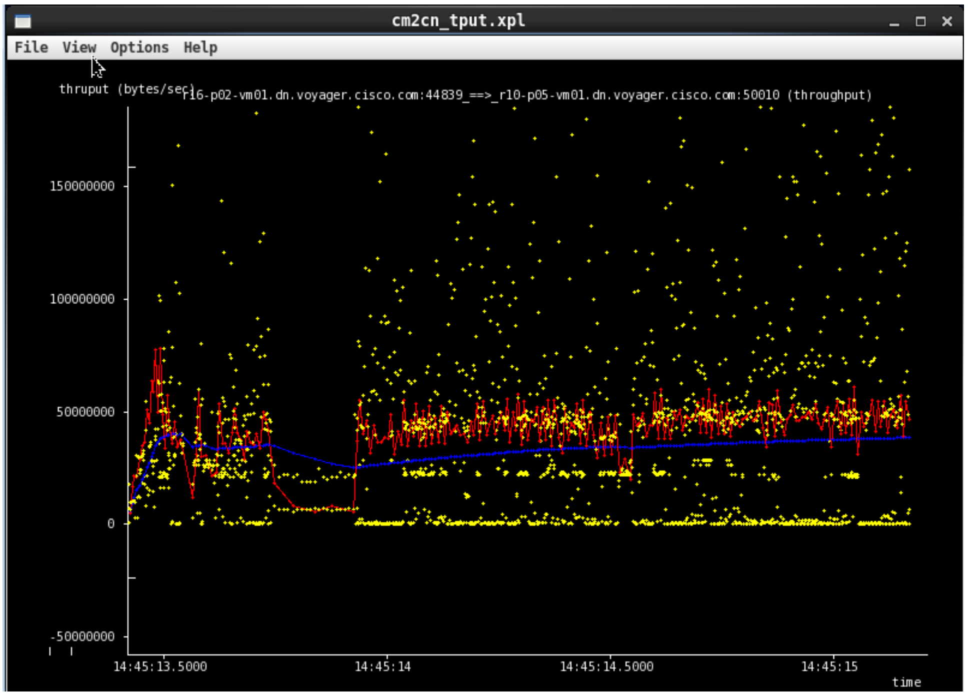
By way of comparison, here is the Control for the test: a copy of the same 1GB job between the Reducer to the output directory, as assigned by HDFS, and no Incast event was present (it's a one to many, not many to one, communication).

```
TCP connection 46:
        host cm:       r16-p02-vm01.dn.voyager.cisco.com:44839
        host cn:       r10-p05-vm01.dn.voyager.cisco.com:50010
        complete conn: yes
        first packet:  Fri Nov  9 14:45:13.413420 2012
        last packet:   Fri Nov  9 14:45:15.188133 2012
        elapsed time:  0:00:01.774713
        total packets: 4542
        filename:      job_0047.pcap
   cm->cn:                             cn->cm:
     total packets:          2146            total packets:          2396
     ack pkts sent:          2145            ack pkts sent:          2396
     pure acks sent:          100            pure acks sent:         1360
     sack pkts sent:            0            sack pkts sent:            0
     dsack pkts sent:           0            dsack pkts sent:           0
     max sack blks/ack:         0            max sack blks/ack:         0
     unique bytes sent:  67659222            unique bytes sent:     12399
     actual data pkts:       2044            actual data pkts:       1034
     actual data bytes:  67659222            actual data bytes:     12399
     rexmt data pkts:           0            rexmt data pkts:           0
     rexmt data bytes:          0            rexmt data bytes:          0
     zwnd probe pkts:           0            zwnd probe pkts:           0
     zwnd probe bytes:          0            zwnd probe bytes:          0
     outoforder pkts:           0            outoforder pkts:           0
     pushed data pkts:        928            pushed data pkts:       1034
     SYN/FIN pkts sent:       1/1            SYN/FIN pkts sent:       1/1
     req 1323 ws/ts:          Y/Y            req 1323 ws/ts:          Y/Y
     adv wind scale:           10            adv wind scale:           10
     req sack:                  Y            req sack:                  Y
     sacks sent:                0            sacks sent:                0
     urgent data pkts:          0 pkts       urgent data pkts:          0 pkts
     urgent data bytes:          0 bytes     urgent data bytes:          0 bytes
     mss requested:           1460 bytes     mss requested:           1460 bytes
     max segm size:          65160 bytes     max segm size:             12 bytes
     min segm size:            210 bytes     min segm size:              3 bytes
     avg segm size:          33101 bytes     avg segm size:             11 bytes
     max win adv:            15360 bytes     max win adv:           195584 bytes
     min win adv:            15360 bytes     min win adv:            16384 bytes
     zero win adv:               0 times     zero win adv:               0 times
     avg win adv:            15360 bytes     avg win adv:           183357 bytes
     max owin:              174158 bytes     max owin:                  37 bytes
     min non-zero owin:          1 bytes     min non-zero owin:          1 bytes
     avg owin:               65325 bytes     avg owin:                  10 bytes
     wavg owin:              66250 bytes     wavg owin:                  0 bytes
     initial window:           241 bytes     initial window:             3 bytes
     initial window:             1 pkts      initial window:             1 pkts
     ttl stream length:   67659222 bytes     ttl stream length:      12399 bytes
     missed data:                0 bytes     missed data:                0 bytes
     truncated data:      67544758 bytes     truncated data:             0 bytes
     truncated packets:       2044 pkts      truncated packets:          0 pkts
     data xmit time:         1.755 secs      data xmit time:         1.766 secs
     idletime max:            18.7 ms        idletime max:            19.2 ms
     throughput:          38124036 Bps       throughput:              6986 Bps

     RTT samples:             1086           RTT samples:              891
     RTT min:                  0.2 ms        RTT min:                  0.1 ms
     RTT max:                  3.0 ms        RTT max:                  8.0 ms
     RTT avg:                  1.3 ms        RTT avg:                  0.7 ms
     RTT stdev:                0.5 ms        RTT stdev:                1.0 ms

     RTT from 3WHS:            0.3 ms        RTT from 3WHS:            0.2 ms

     RTT full_sz smpls:          2           RTT full_sz smpls:          2
     RTT full_sz min:          0.3 ms        RTT full_sz min:          0.2 ms
     RTT full_sz max:          0.8 ms        RTT full_sz max:          0.3 ms
     RTT full_sz avg:          0.5 ms        RTT full_sz avg:          0.2 ms
     RTT full_sz stdev:        0.0 ms        RTT full_sz stdev:        0.0 ms

     post-loss acks:             0           post-loss acks:             0
     segs cum acked:           960           segs cum acked:           145
     duplicate acks:             1           duplicate acks:             0
     triple dupacks:             0           triple dupacks:             0
     max # retrans:              0           max # retrans:              0
     min retr time:            0.0 ms        min retr time:            0.0 ms
     max retr time:            0.0 ms        max retr time:            0.0 ms
     avg retr time:            0.0 ms        avg retr time:            0.0 ms
     sdv retr time:            0.0 ms        sdv retr time:            0.0 ms
================================
```

It comes as no surprise that RTT is significantly less than when there was Incast: 3ms down from ~60ms, what one would expect for a 1:1 interaction.

Finally, Figure 2-35 shows the scatterplot of the TCP connection while the file was being copied.

Figure 2-35 Example of Good TCP Throughput for 1:1 Control Test



The reason for the dip ¼ the way through is inconclusive, but the important point is that it doesn't go to zero, nor is slow-start seen after the dip (as one would expect if collapse had occurred), and the file copy for the Control test completed in 1.7 seconds (with reasonable RTT), as opposed to 14 seconds for Job47.

Incast Testing Summary

The objectives of performing Incast testing for Phase 1 were achieved, that is:

1. Hadoop was successfully used as a generic Incast traffic generator.
2. The Incast event was correctly identified and tracked using open tools, including Graphite, Wireshark, tcpdump, tcptrace, and SNMP stats from the N3K. Also, custom Python scripts for shared buffer monitoring were successfully executed directly on the N3K platform (refer to Incast Utility Scripts, IXIA Config, page E-1).
3. The N3K was shown to be able to deal with Incast insofar that it could allocate shared buffer enough to ensure the transaction completed.

Future Phases of MSDC testing may include additional Incast research. Such research would potentially explore additional tuning on both Linux and NX-OS platforms to better signal when Incast events occur, and perhaps even deal with Incast more proactively using technologies like ECN and buffer usage trending.

MSDC Conclusion

The purpose of this document was to:

1. Examine the characteristics of a traditional data center and a MSDC and highlight differences in design philosophy and characteristics.
2. Discuss scalability challenges unique to a MSDC and provide examples showing when an MSDC is approaching upper limits. Design considerations which improve scalability are also reviewed.
3. Present summaries and conclusions to SDU's routing protocol, provisioning and monitoring, and TCP Incast testing.
4. Provide tools for a network engineer to understand scaling considerations in MSDCs.

It achieved that purpose.

- Customers' top-of-mind concerns were brought into consideration and effective use of Clos topologies, particularly the 3-stage folded Clos, we examined and demonstrated how they enable designers to meet east-west bandwidth needs and predictable traffic variations.
- The Fabric Protocol Scaling section outlined considerations with Churn, OSPF, BGP, and BFD with regard to scaling.
- OSPF was tested and shown were current system-wide limits contrast with BGP today. For BGP, it was demonstrated how the customer's peering, reliability, and resiliency requirements could be met with BGP + BFD.
- Along with (3), the N3K was shown to have effective tools for buffer monitoring and signaling when and where thresholds are crossed.

Using underlying theory, coupled with hands-on examples and use-cases, knowledge and tools are given to help network architects be prepared to build and operate MSDC networks.