CHAPTER 1

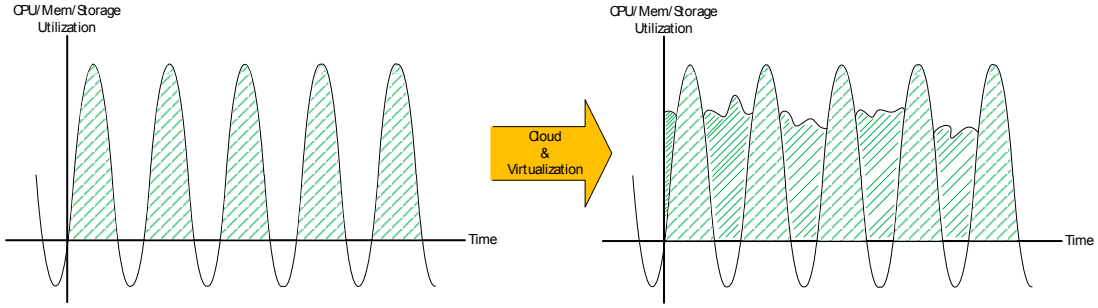
MSDC Scale Characteristics

The following scaling characteristics are defined as a prelude to the Massively Scalable Data Center design that drives the MSDC technology and differentiates it from VMDC/Enterprise.

Drivers

- **Commoditization**!
- **Cloud Networking**. Classical reasons to adopt cloud networking include:
  - Improving compute, memory, and storage utilization across large server fleets (Figure 1-1). Efficiencies improve when troughs of the utilization cycle are filled in with useful work.

Figure 1-1 Optimization Benefits of Clouds



  - Increased efficiencies enable customers to innovate by freeing up compute cycles for other work as well as providing a more flexible substrate to build upon.
- **Operations and Management (OaM)**.
- **Scalability**. Application demands are growing within MSDCs. This acceleration requires infrastructure to keep pace.
- **Predictability**. Latency variation needs to be kept within reasonable bounds across the entire MSDC fabric. If every element is nearly the same, growth is easier to conceptualize and the impact scaling has on the overall system is relatively easy to predict—**homogeneity**, discussed later in Design Tenets, page 1-5, is a natural outgrowth of predictability.

Differences Between VMDC/Enterprise and MSDC

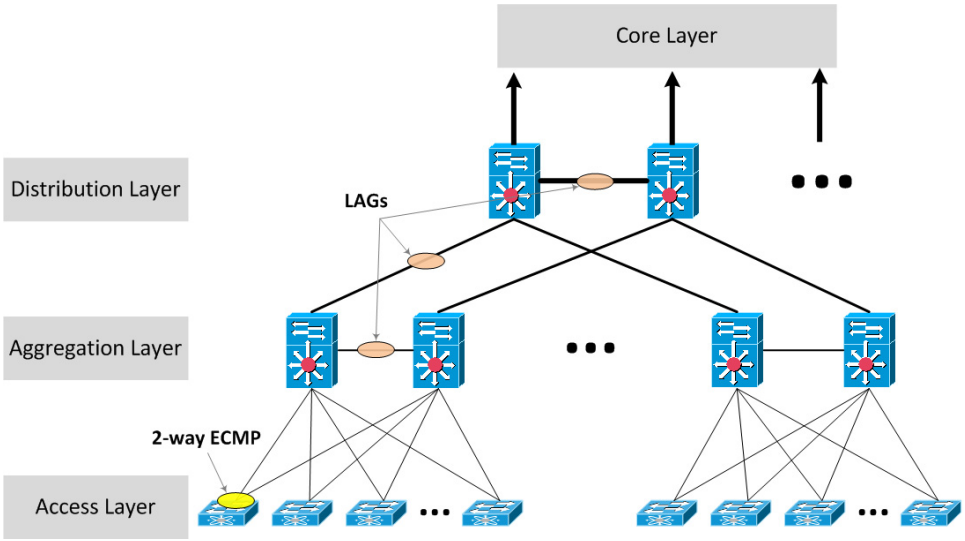
Here are some key concepts that differ between VMDC/Enterprise and MSDCs:

1. **Fault Tolerance vs. Fault Avoidance—**The concept of **Fault Tolerance** has traditionally been interchangeable with redundancy when describing networks. For purposes of this document the scope of its definition is narrowed to a specific type of redundancy; faults are handled by the entire system and impact is minimized by overall system design [high degree of Equal Cost Multi-Pathing (ECMP)]. ECMP can dramatically increase cross-sectional bandwidth available between any two layers of the network. Since ECMP provides many parallel paths it is tolerant of any number of path failures (assuming bandwidth needs do not exceed remaining links capacity). On the other hand, VMDCs are more concerned about **Fault Avoidance,** which is where faults are handled by individual components and impact is avoided by designing redundant components into nodes which comprise the system (dual SUPs).
2. **Scale—**MSDC data centers interconnect tens to nearly hundreds of thousands of compute, memory, and storage resources under a single roof (or in some cases, many roofs whose networks are joined via optical infrastructures), comprising a single, unified network. Typical MSDCs can have over 24,000 point to point links, 250,000+ servers, and over 800 network elements.
3. **Churn—**The steady state of a network designed for fault tolerance has a routing protocol which is in a constant state of flux. Such network flux is called **churn.** Churn can be caused by unplanned events such as link failures, linecard or chassis failures, routing loops, as well as planned events, such as Change Management procedures. Routing protocols provide insight into amounts of churn a network experiences—as portions of the network are brought offline or become unavailable due to unplanned failures, routing protocols notice such changes and propagate updates to the rest of the network. The more churn, the more routing updates are seen. Churn in MSDCs is the norm—always in a constant state of flux.

Traditional Data Center Design Overview

Figure 1-2 shows a traditional data center network topology.

Figure 1-2 Traditional Network Topology



These networks are characterized by a set of aggregation pairs (AGGs) which aggregate many access (aka Top of Rack, or ToR) switches. AGGs then connect to an upstream distribution (DIS) layer, which is followed by a core (COR) layer which aggregates the DIS layer and connects to other networks as needed. Another noticeable characteristic in these networks which differ from that of MSDCs is inter-AGG, inter-DIS, and inter-COR links between pairs; in MSDCs the amount of bandwidth needed, and the fact that today's platforms do not provide the necessary port density, make it unnecessary and even cost-prohibitive to provide inter-device links which meet requirements. In MSDCs, the routing decision to take a particular path from ToR to the rest of the network is made early on at the ToR layer.

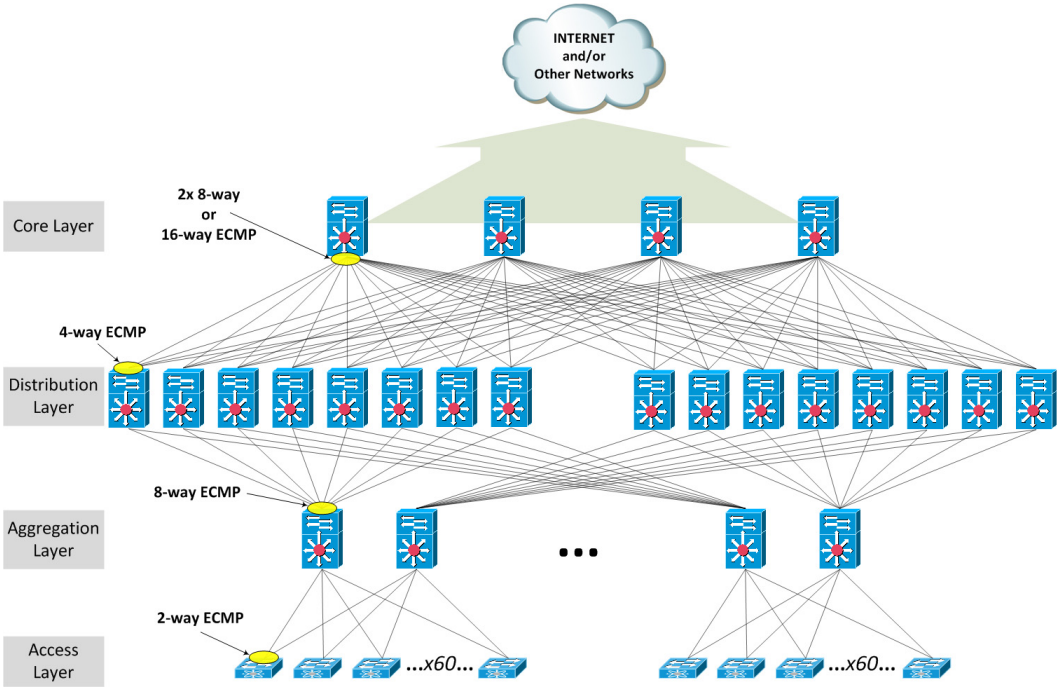
Traditional data center networks are designed on principles of fault avoidance. The strategy for implementing this principle is to take each switch[1] (and links) and build redundancy into it. For example, two or more links are connected between devices to provide redundancy in case of fiber or transceiver failures. These redundant links are bundled into port-channels that require additional configuration or protocols. Devices are typically deployed in pairs requiring additional configuration and protocols like VRRP and spanning-tree to facilitate inter-device redundancy. Devices also have intra-device redundancy such as redundant power supplies, fabric modules, clock modules, supervisors, and line cards. Additional features (SSO) and protocol extensions (graceful-restart) are required to facilitate supervisor redundancy. The steady state of a network designed with this principle is characterized by a stable routing protocol. But it comes at the expense of:

- Operational complexity.
- Configuration complexity.
- Cost of Redundant Hardware—this in turn increases capital costs per node and increases the risk of things to fail, longer development time, longer test plans.
- Inefficient use of bandwidth (single rooted).
- Not being optimized for small flows (required by MSDCs).

Whereas MSDCs are more interested in being able to fail a device and the overall system doesn't care—thus reducing liability each network element can introduce into the system upon failure—again, this is fault tolerance.

1. For the purposes of this document, the term "switch[es]" refers to basic networking elements of an MSDC network. These basic elements can be routers and/or switches, in the strict sense. However unless otherwise noted, a "switch" is a L3 device that can perform both traditional L2 switching and L3 routing functions, including speaking routing protocols such as OSPF and BGP.

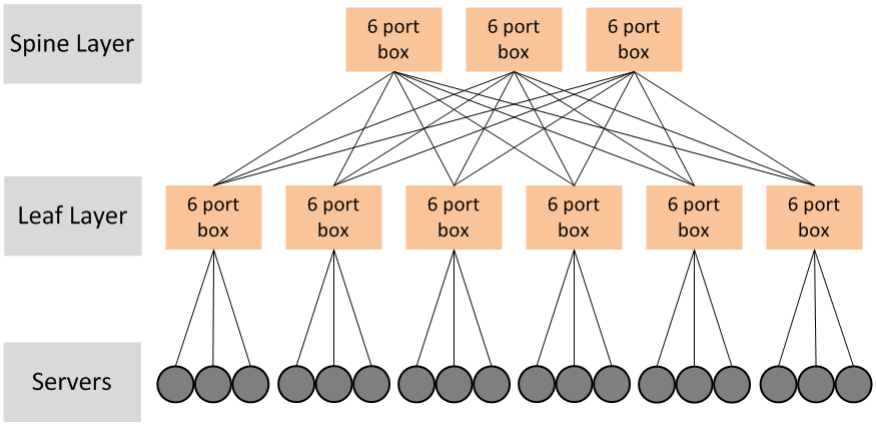
Figure 1-3 Evolution, From the Field



Evidence of this break from traditional data center design is already observed in the field, as seen in this sanitized version of a particular customer's network above. Here a higher degree of ECMP is seen than is present in earlier network architectures, however there are still weaknesses in the above design – most notably the sizeable reduction in bandwidth capacity if one AGG device fails. ECMP is allowing for higher cross-sectional bandwidth between layers, thus greater east-west bandwidth for applications, and reduces the fault domain as compared to traditional designs – that is failure of a single device only reduces available bandwidth by a fraction.

Finally, the logical conclusion to the trend towards more ECMP is a Clos[2] design with a "Spine" and a "Leaf" as shown in Figure 1-4. The Spine is responsible for interconnecting all Leafs, and provides a way for servers in one rack to talk to servers in another in a consistent way. Leafs are responsible for equally distributing server traffic across all Spine nodes.

Figure 1-4 3-Stage Folded Clos Topology



2. Refer to Interconnecting Building Blocks, page 1-9 for details on how MSDC's use Clos topology.

Design Goals

Data in this guide is based on thorough research into customer motivations, design tenets, and top-of-mind concerns, all coupled by the drivers discussed.

Design Tenets

All engineering requirements for the design of a MSDC are mapped, at varying degrees, to these fundamental concerns and governing tenets:

- **Cost**—Customers want to spend less on their network, or turn their network into a profit-center. For example, a 1W savings in power on a server NIC can translate to $1M saved overall.
- **Power**—Reducing power footprint, as well as improving PDU efficiencies, are major concerns to customers.
- **East-West BW**—AKA "crosstalk". Applications are demanding more bandwidth due to multi-layers and large fanout. In a MSDC context, applications typically generate huge fanout ratios, for example 1:100. For every byte inbound to the data center, this can translate to 100bytes inside the MSDC because a typical social website 2.0 takes well over 100 backend (east-west) transactions per single north-south transaction. Oversubscription is less tolerated in MSDC environments.
- **Transparency**—Customers use this term to help communicate the idea of building an intelligent network which fosters easier, predictable communication between East-West components.
- **Homogeneity**—Eliminating one-offs makes operating MSDC networks easier at scale.
- **Multipathing**—ECMP brings fault domain optimization. ECMP reduces liability of a single fault, or perhaps a small number of faults, to the overall system.
- **Control**—Programmability, automation, monitoring and bug/defect management, and innovation velocity. The more customers control (code), vendor's adoption of relevant technologies, the more they can integrate into their own software infrastructure which gives them a competitive advantage. Being able to influence quality assurance with Vendors are traits that give customers control they need to operate successful environments.

Customer Architectural Motivations

From these tenets, MSDC networks are designed based on a principle of fault tolerance. Mechanisms aren't put into place to prevent failures from happening, but rather to minimize the impact a fault has on network operations. The strategy for implementing this principle is to design redundancy into the network architecture.

A traditional 2-wide AGG or DIS pair (Figure 1-2 and Figure 1-3) will lose 50% of its capacity if a single node fails. A scaled out MSDC architecture minimizes the impact of similar failures by creating more parallel nodes. The multiple parallel nodes of a Clos network[3] Spine are a perfect example of redundancy coming from the network architecture which also minimizes the impact a fault has on overall network operations. In an 8x Spine[4] Clos network, for example, loss of an entire Spine node would only

3. Refer to Interconnecting Building Blocks, page 1-9 for additional analysis of Clos topologies and their benefits.
4. 8x Spines means "8 devices that make a single Spine". This nomenclature will be used throughout this document. In cases where multiple Spines are used, it will be notated like: 8x8 Spine, that is, 8 overall Spines that are connected in parallel, each Spine also being composed of 8 devices. Also, refer to Interconnecting Building Blocks, page 1-9 for a more formal definition of Spines and Leafs.

reduce available bandwidth by one-eighth. Leaf devices could have two independent uplink failures and still operate at 75% capacity. From these two examples it is apparent that fault tolerant network design moves redundancy from individual network elements to the network as a system. Instead of each network element having a unique mechanism to handle its own failures, the routing protocol is responsible for handling failures at all levels. This drastically reduces configuration and operational complexity of each device. But simplification, as always, comes at a cost (flexibility) which must be balanced against the benefits of simplification.

As mentioned earlier, “M” in MSDC means “massive”. MSDC networks are massive, and require astounding amounts of fiber (24,576 links), transceivers (49,152 xfp/sfp+), power supplies (over 800 devices), line cards, supervisors, chassis, etc. Such data centers are home to tens or hundreds of thousands of physical servers, and the burden to interconnect those in intelligent ways is non-trivial. These network elements are put into the domain of a single routing protocol. Due to the sheer number of network elements in a single domain, failures are routine. Failures are the norm! Also, in MSDC networks, the “application” is tightly integrated with the network and often participates with routing protocols. For example, Virtual IPs (VIPs, these are IP addresses from services which load balancers are advertising) can be injected or withdrawn into the network at the discretion of the application. Routing protocols must keep the network stable despite near constant changes coming from both application updates and network element failures. Dealing with churn is a primary motivation for moving all redundancy and resiliency into the network.

Note Failures can be caused by various sources, whether intentional or not.

Top of Mind Concerns

MSDC customers face the following three major areas of concern on a daily basis:

- Operations and Management, page 1-6
- Scalability, page 1-6
- Predictibility, page 1-7

Any architectures or solutions customers may want to implement will be tempered by these main concerns discussed below.

Operations and Management

Operations and Management (OaM) of MSDCs is a major demand on those operating such networks. The operational implications of a MSDC cannot be overstated. Customers want tools to help them increase provisioning velocity, make change management procedures take less time, increase visibility of important telemetry, and minimize human-caused outages.

Scalability

The scalability limits of *individual* devices that make up MSDCs are well known. Each platform has route scale limits defined by TCAM partitioning and size. Base measurements like these can be used to quantify a network with a stable steady state. However, these limits do not properly define scalability of MSDC networks. Routing protocols are invoked in nearly every fault scenario, and as discussed in a previous section titled “Scale, Differences between VMDC/Enterprise and MSDC”, MSDCs are so large that faults are routine. Therefore true scalability limits of MSDC networks are in part defined by the capacity of its routing protocol to handle network churn.

Deriving a measurement to quantify network churn can be difficult. Frequency and amplitude of routing protocol updates depends on several factors; explicit network design, application integration, protocols used, failure rates, fault locations, etc. Measurements derived would be specific to the particular network, and network variations would bring statistical ambiguity. A more useful question is; "Depending on a particular network architecture, how does one know when churn limits have been reached?" MSDC customers are asking such a question today. For details, refer to Scalability, page 1-6.

Predictibility

Predictible latencies across the MSDC fabric are critical for effective application workload placement. A feature of the Clos topology is all endpoints are equidistant from one another, thus it doesn't matter where workloads are placed, at least in terms of topological placement.

Reference Topologies and Network Components

The SDU MSDC test topologies are intended to be a generic representation of what customers are building, or want to build, within the next 24 months. The building blocks used are what customers are ordering today, and have a solid roadmap going forward. MSDC customers care less about things such as ISSU (high availability for each individual component), and are more concerned with high-density, inexpensive, and programmable building blocks.

Building Blocks

The guide uses specific hardware and software building blocks, all of which fulfill MSDC design tenets. Building blocks must be cost-sensitive, consume lower power, simpler, programmable, and fascilitate sufficient multipathing width (both hardware and software are required for this).

Building blocks are broken down into three areas:

- Leaf Layer, page 1-7
- Spine Layer, page 1-8
- Fib, page 1-8

Leaf Layer

The Leaf Layer is responsible for advertising server subnets into the network fabric. In MSDCs this usually means Leaf devices sit in the Top-of-Rack (ToR), if the network is configured in a standard 3-stage folded Clos design[5].

Figure 1-5 shows the Nexus 3064, the foundation of the Leaf layer.

Figure 1-5 N3064

5. Refer to Interconnecting Building Blocks, page 1-9 for Clos details.

The Leaf layer is what determines oversubscription ratios, and thus size of the Spine. As such, this layer is of top priority to get right. The N3064 provides 64x 10G linerate ports, utilizes a shared memory buffer, is capable of 64-way ECMP, and features a solid enhanced-manageability roadmap.

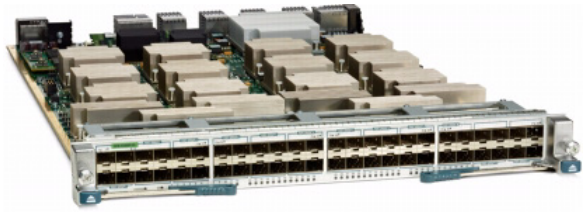
In exchange for Cisco's devices which employ more feature-rich ASICs (M-series linecards, 5500 switches, ISSU, triple redundancy) , this layer employs simpler designs that have fewer "moving parts" to effectively forward packets while learning the network graph accurately.

Spine Layer

The Spine layer is responsible for interconnecting all Leafs. Individual nodes within a Spine are not connected to one another nor form any routing protocol adjacencies among themselves. Rather, Spine devices are responsible for learning "infrastructure" routes, that is routes of point-to-point links and loopbacks, to be able to correctly forward from one Leaf to another. In most cases, the Spine is not used to directly connect to the outside world, or other MSDC networks, but will forward such traffic to specialized Leafs acting as a Border Leaf. Border Leafs may inject default routes to attract traffic intended for external destinations.

Figure 1-6 shows the F2 linecard providing 48x 10G linerate ports (with the appropriate Fabric Cards).

Figure 1-6 F2 Linecard



The Nexus 7K is the platform of choice which provides high-density needed for large bandwidth networks, has a modular operating system which allows for programmability. The N7004 consumes 7RU of space but only provides 2 I/O slots, and is side-to-side airflow (although not a first-order concern, MSDCs prefer front-to-back, hot-isle, cold-isle cooling when they can get it). The N{7009|7010|7018} are preferable since their port-to-RU ratio is much higher (real-estate is a concern in MSDCs). If front-to-back airflow is required, the N7010 provides this function. N7009 and N7018 utilize side-to-side airflow. The building blocks in SDU testing employs all three N{7009,7010,7018} platforms.

Customers have voiced concern about complexities and costs of the M-series linecards, and thus requested simpler linecards that do less, but do those fewer tasks very fast and highly reliable. F2 fits very well with those requirements. It provides low-power per 10G port, low-latency, and utilizes ingress buffering to support large fanout topologies.

Note The F2 linecard is based on Cisco's Clipper ASIC detailed in Appendix C, "F2/Clipper Linecard Architecture".

Fib

New FIB management schemes are needed to meet the demands of larger networks. The number of loopbacks, point-to-point interfaces, and edge subnets are significantly higher than in traditional networks. And as MSDCs are becoming more cloud-aware, more virtualization-aware, the burden on a FIB can skyrocket.

Obviously, dedicating hardware such as TCAM to ever-growing FIBs is not scalable; the number of entries can grow to hundreds of millions, as seen in some MSDC customer's analysis. This is cost and power prohibitive.

Regardless of size, managing FIB churn is a conern.

Strategies to address this concern:

1. One strategy to manage FIB is merely to reduce the size of FIB by separating infrastructure[6] routes from customer[7] routes. If the network system could relegate hardware to simply managing infrastructure ONLY, this could take the FIB from hundreds of thousands, even millions, down to 24,000 or less. Customer routes could be managed by a system that is orthogonal to the infrastructure itself, this could be the network, or it could be off-box route controller cluster(s).
2. The strategy used in Phase 1 was to manage the FIB by learning routes over stable links – links that are directly connected routes. In such situations, churn is only introduced as physical links go down and is less fragile than a topology which completely relies on dynamic insertion of prefixes. For example, MSDC networks based on a 3-stage Clos architecture may have 32 Spine devices (N7K+F2) and 768 Leaf devices (N3064). The FIB will be comprised of a stable set of 24,576 point-to-points, 800 loopbacks, and then server subnets being advertised by the Leafs.

Interconnecting Building Blocks

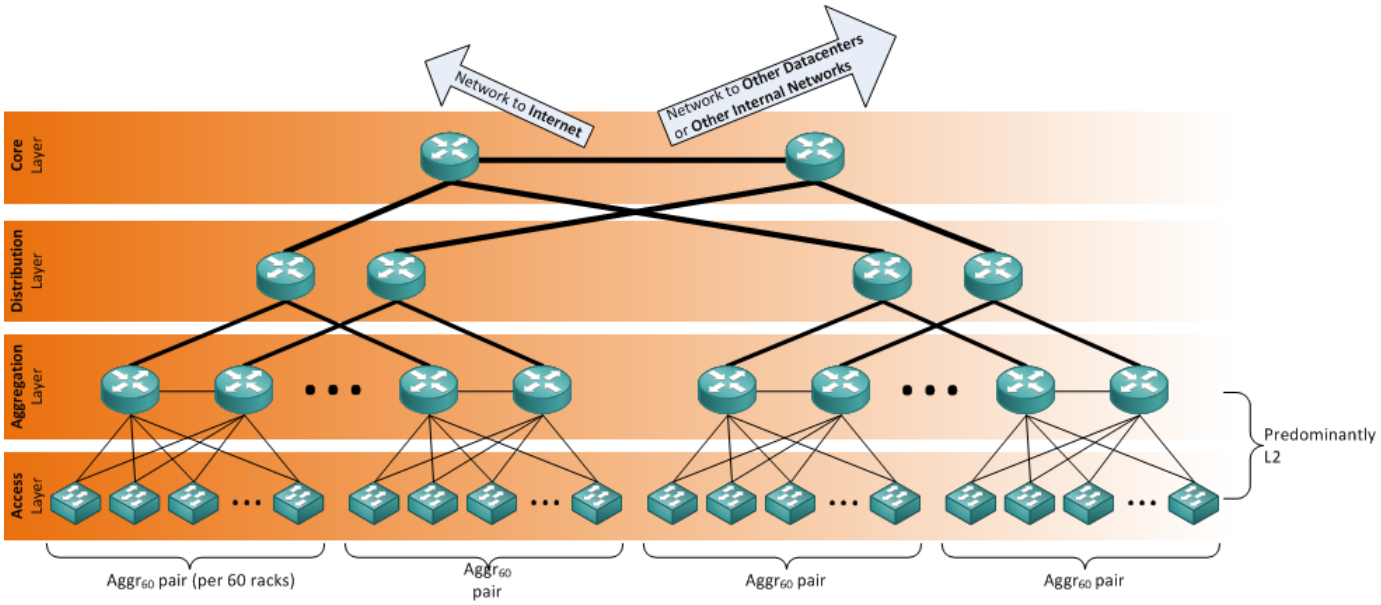
As stated in the "Scale" section on page 1-31, MSDC data centers have many compononets comprising a single whole. Getting such resources to talk to one another, as well as delivering packets to and from the Internet, is non-trivial. Therefore it is recommended to review designs that make MSDC fabrics more robust and incrase resiliency.

Traditional Tree Hierarchy

Refering back to the "Traditional Data Center Design Overview" section on page 1-2, a traditional tree hierarchy may look similar to Figure 1-7.

6. Infrastructure routes = loopbacks and point-to-point fabric links.
7. Customer routes = the subnets the end-hosts live in.

Figure 1-7 Traditional, Hierarchical Design

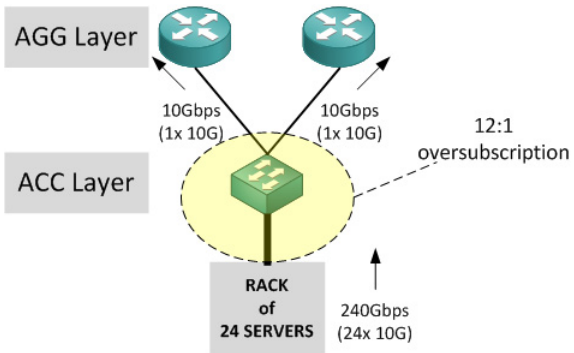


These networks are characterized by a set of aggregation pairs (AGGs) which aggregate many access (aka Top of Rack) switches.

The bandwidth increases significantly near the root of the tree, but non-blocking functionality is not supported, therefore introducing significant oversubscription. Examples of oversubscription and blocking, in traditional architectures, are displayed in Figure 1-8.

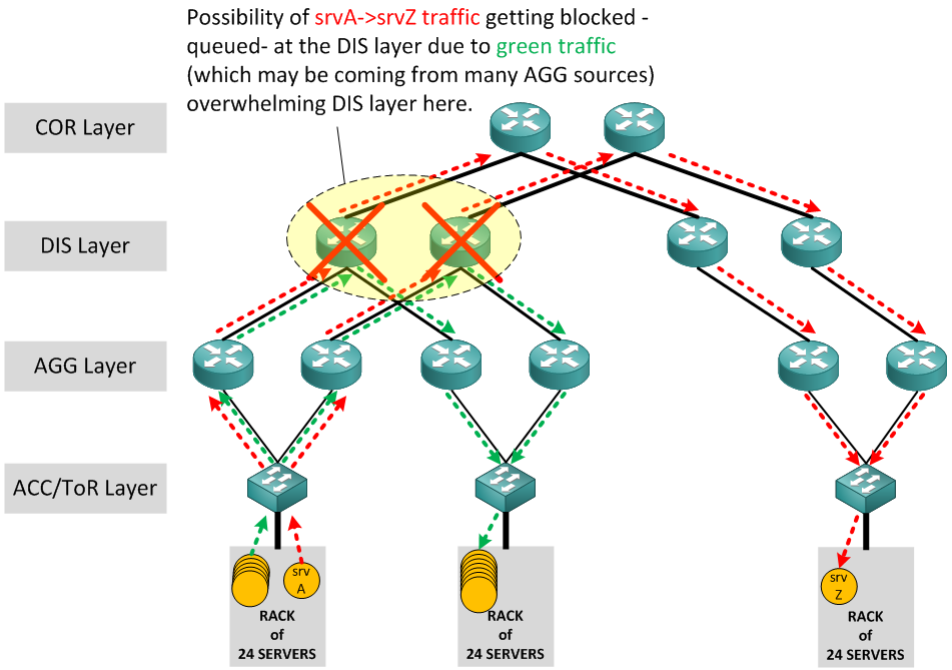
Oversubscription—means ingress capacity exceeds egress capacity. In Figure 8, if you have a rack of 24x 10G attached servers, the ACC device needs at least 240G of port capacity facing the upstream layer to be 1:1 oversubscribed (1:1 would actually mean there is NO oversubscription). If the ACC device has 24x 10G server ports and 2x 10G uplinks, you have 12:1 oversubscription. To allow the entire network to operate at linerate, 1:1 oversubscription is required. However, not all networks need to provide 1:1 performance; some applications will operate fine when oversubscription occurs. Therefore in some scenarios non-blocking designs aren't necessary. The architect should have a thorough understanding of application traffic patterns, bursting needs, and baseline states in order to accurately define the oversubscription limits a system can tolerate.

Figure 1-8 Oversubscription Scenario



Blocking—Oversubscription situations at device level, and even at logical layers, are causes of applications getting blocked which results in network queueing. For example in Figure 1-9 server A wants to talk to server Z, but the upstream DIS layer is busy handling other inter-rack traffic. Since the DIS layer is overwhelmed the network causes server A to "block". Depending on the queueing mechanisms and disciplines of the hardware, queueing may occur at ingress to the DIS layer.

Figure 1-9 Blocking Scenario



In Figure 1-8 and Figure 1-9 10G interfaces are being considered as the foundation. If the system wants to deliver packets at line-rate these characteristics should be considered:

- Each ACC/ToR device could only deliver 20G worth of server traffic to the AGG layer, if we assume there are only 2x 10G uplinks per ACC device. That represents only 8% of total possible server traffic capability! This scenario provides a 1:1 oversubscription.
- Each AGG device needs to deliver ten times the number of racks-worth of ACC traffic to the DIS layer.
- Each DIS device needs to deliver multiple-terrabytes of traffic to the COR layer.

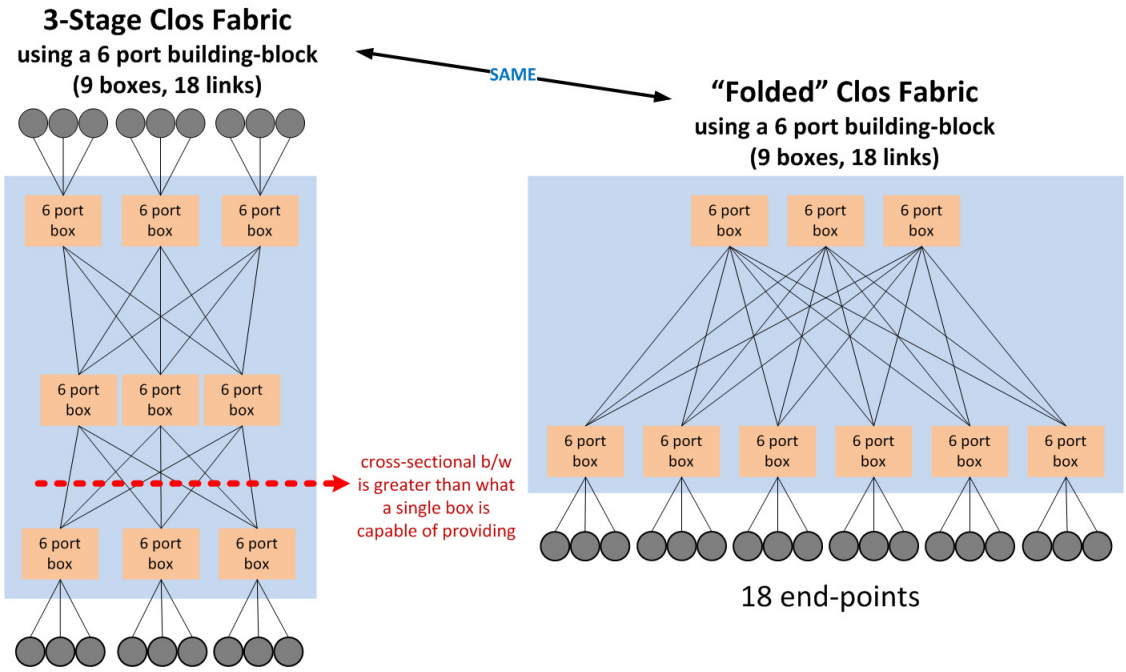
Scaling such a network becomes cost-prohibitive, and growth becomes more complex because additional branches of the tree need to be built to accomodate new pods . In addition to bandwidth constraints, there are also queueing concerns based on the amount of buffer available to each port within the fabric.

The problem with these traditional topologies, in the MSDC space, is they can't sustain the burst of east-west traffic patters and bandwidth needs common in MSDC environments.

Clos

Figure 1-10 shows an example of a Clos topology composed of a hypothetical 6-port building block.

Figure 1-10 3-Stage Clos



In 1953 Charles Clos created the mathematical theory of the topology which bears his name; a non-blocking, multi-stage topology which provides greater bandwidth than what a single node[8] is capable of supplying. The initial purpose of the Clos topology was to solve the n^2 interconnect problem in telephone switching systems: it interconnects n inputs to n outputs with ? n^2 nodes. The labeling of both inputs and outputs with the same variable, n, is by design – that is, we marry each output with an input; the number of outputs equals the number of inputs, or there is precisely one connection between nodes of one stage to those of another stage. Said another way, a Clos network connects a large number of inputs and outputs with "smaller-sized" nodes.[9]

In this example, using 6-port switches, we connect 18 endpoints (or "edge ports") in non-blocking fashion[10] using a 3-stage Clos topology. We use the phrase "folded Clos" to mean the same thing as a 3-stage Clos, but is more convenient for network engineers to visualize ports, servers, and topology in a folded Clos manner. For terminology, in a 3-stage Clos we have an ingress Leaf layer, a Spine center layer, and an egress Leaf layer. If we fold it, we simply have a Leaf layer and a Spine layer.

If we create a Clos network using building blocks of uniform size, we calculate the number of edge ports using a relationship derived from Charles Clos' original work:

$$num_{edgeports} = \frac{k^{h-1}}{(h-1)}$$

where k is the radix of each node (its total number of edges), and h is the number of stages (the "height" of the Clos). Some examples:

- k=6, h=3

8. The term "node" is synonymous with "switch".
9. Since the context of this document is about networks, not unidirectional phone interconnects, we will consider the term "ports" to mean bi-directional ports that contain both TX and RX lines.
10. Non-blocking for this document means a sender X can send to receiver Y and not be blocked by a simultaneous sender Q, hanging off the same switch as X, sending to receiver R, which lives on a different switch than Y.

$$num = \frac{6^{3-1}}{3-1} = \frac{36}{2} = 18$$

- k=64, h=3

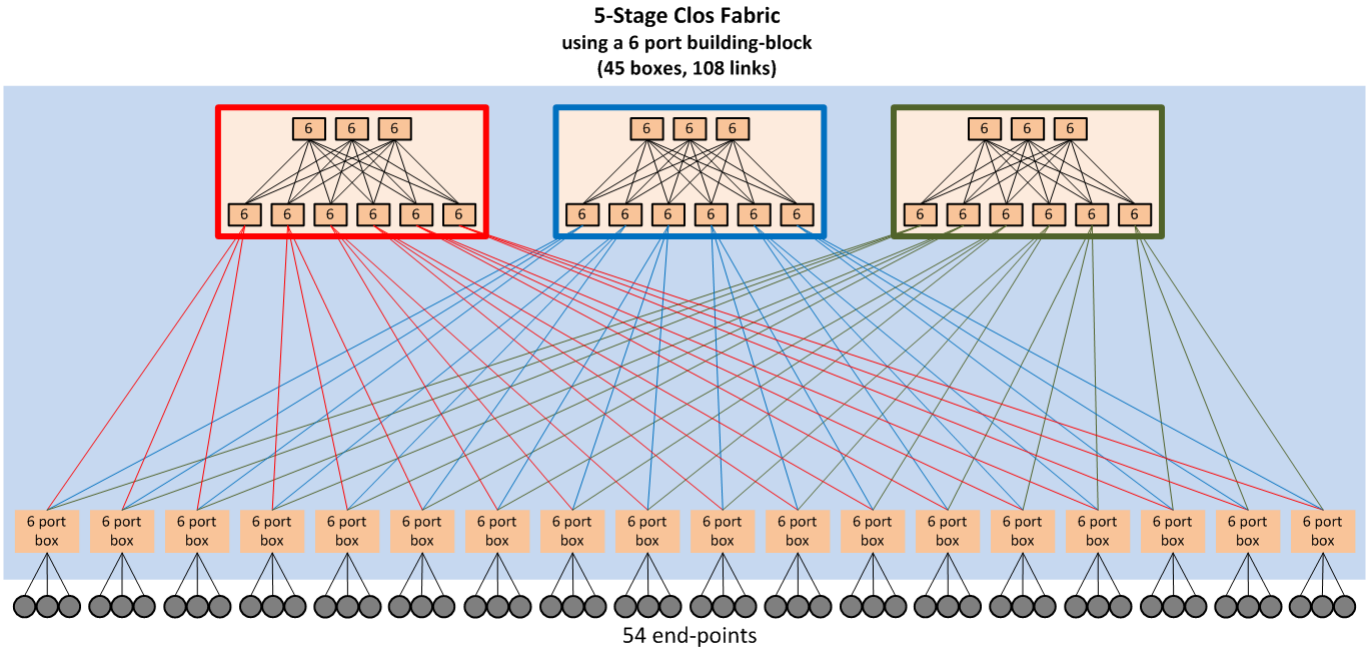
$$num = \frac{64^{3-1}}{3-1} = \frac{4096}{2} = 2048$$

- k=64, h=5

$$num = \frac{64^{5-1}}{5-1} = \frac{16{,}777{,}216}{4} = 4{,}194{,}304$$

Intuition, however, shows a 5-stage Clos built using, say, the Nexus 3064, doesn't actually give you more than 4 million edge ports, but rather 65,536 edge ports (2048 Leafs multiplied by 32 edge-facing ports). Figure 1-11 shows an example of a 5-stage folded Clos, using 6-port building blocks.

Figure 1-11 5-Stage Clos



Here we have 54 edge ports (not 214 ports as the formula predicts), up from 18 when using a 3-stage Clos. The primary reason to increase the number of stages is to increase the overall cross-sectional bandwidth between Leaf and Spine layers, thus allowing for an increased number of edge ports.

Note The discrepancy between the above formula and intuition can be explained by a "trunking" factor in Clos' derivation due to the middle stages – since the N3064 isn't a perfect single crossbar, but rather a multi-stage crossbar itself, the above formula does not work where h ? 5. And it should be noted that in the strict sense a Clos network is one in which each building-block is of uniform size and is a perfect, single crossbar.

As such, because the nodes of today (Nexus 3064) are multi-stage Clos'es themselves, a more appropriate formula for MSDC purposes is one in which *h* is always 3 (in part because cabling of a strict Clos network where *h* ? 5 is presently cost-prohibitive, and the discussion of more stages is beyond the scope of this document), and the formula is simplified to:

$$num = \frac{Nk}{2}$$

Where *N* is the radix of Spine nodes and *k* is the radix of each Leaf; we divide by two because only half the ports on the Leaf (*k*) are available for edge ports at 1:1 oversubscription. Therefore a 3-stage Clos using only N3064s would provide 2048 edge ports:

$$num = \frac{64 * 64}{2} = 2048$$

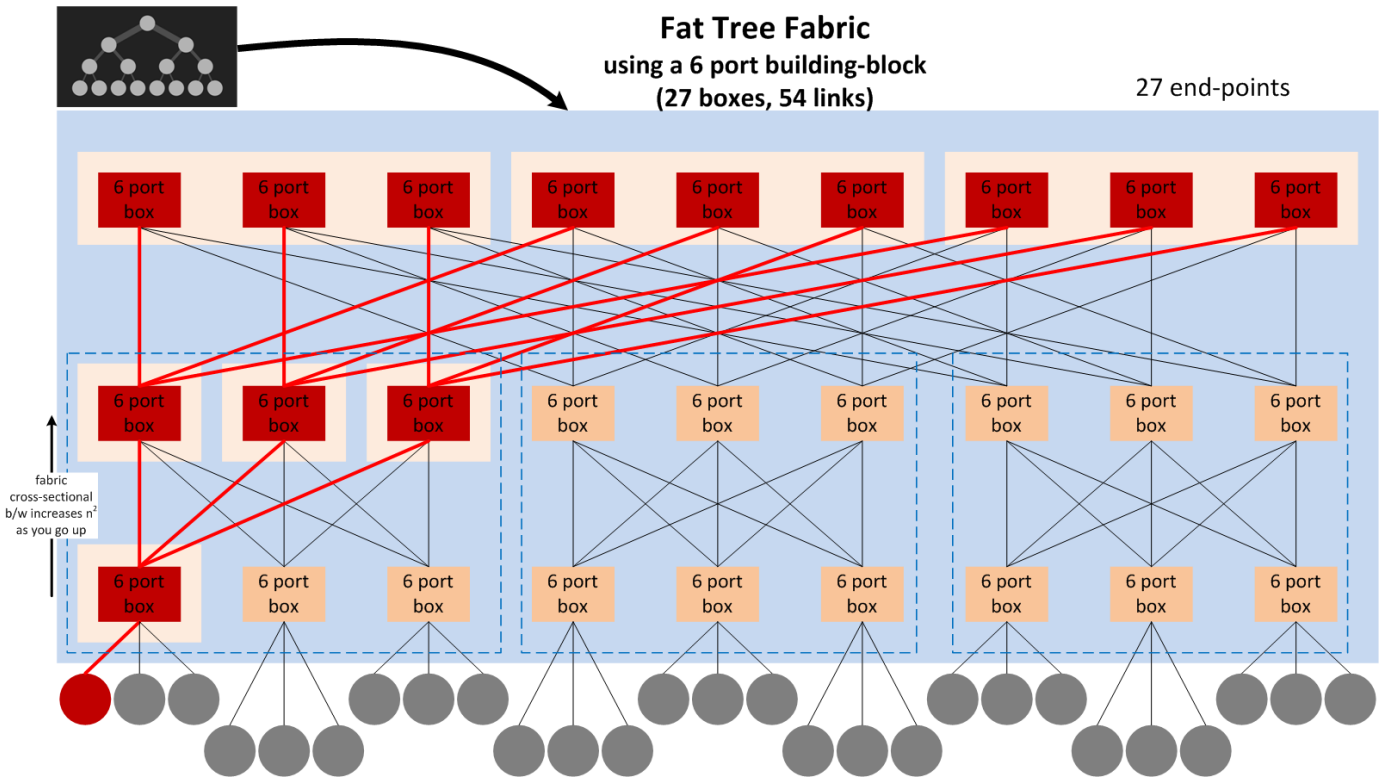
Or, a 3-stage Clos using fully-loaded N7018s+F2 linecards as Spine nodes and N3064s as Leafs, you get 24,576 edge ports:

$$num = \frac{768 * 64}{2} = 24{,}576$$

Fat Tree

A Fat Tree is a tree-like arrangement of a Clos network (Figure 1-12), where the bandwidth between each layer increases by x^2, hence the tree gets thicker (fatter) the closer to the trunk you get.

Figure 1-12 Fat Tree Topology



Note the boxes outlined with dotted-blue; these are the "Leafs" of the Clos; the topmost grouping of nodes, 3x3, are each the "Spines". In other words, you essentially have a 3-stage folded Clos of 6 "nodes", comprised of 3x 6-port Spines nodes and 3x 6-port Leafs. This creates 27 edge-ports.

Compared to a standard Clos, while it's true you get more edge ports with a Fat Tree arrangement, you also potentially have more devices and more links. The cost of doing such must considered when deciding on a particular topology.

Table 1-1 compares relative costs of Clos and Fat-trees using hypothetical 6-port building blocks.

Table 1-1 Clos and Fat-Trees Relative Cost Comp Using Hypothetical 6-Port Building Blocks

Topology	Building Block, Ports/Box	Fabric Boxes	Fabric Links (don't Include server links)	Total End-hosts
Fat Tree	6	27	54	27
Clos-3	6	9	18	18
Clos-5	6	45	108	54

Figure 1-13 Topologies, Comparison of Relative Costs[11]

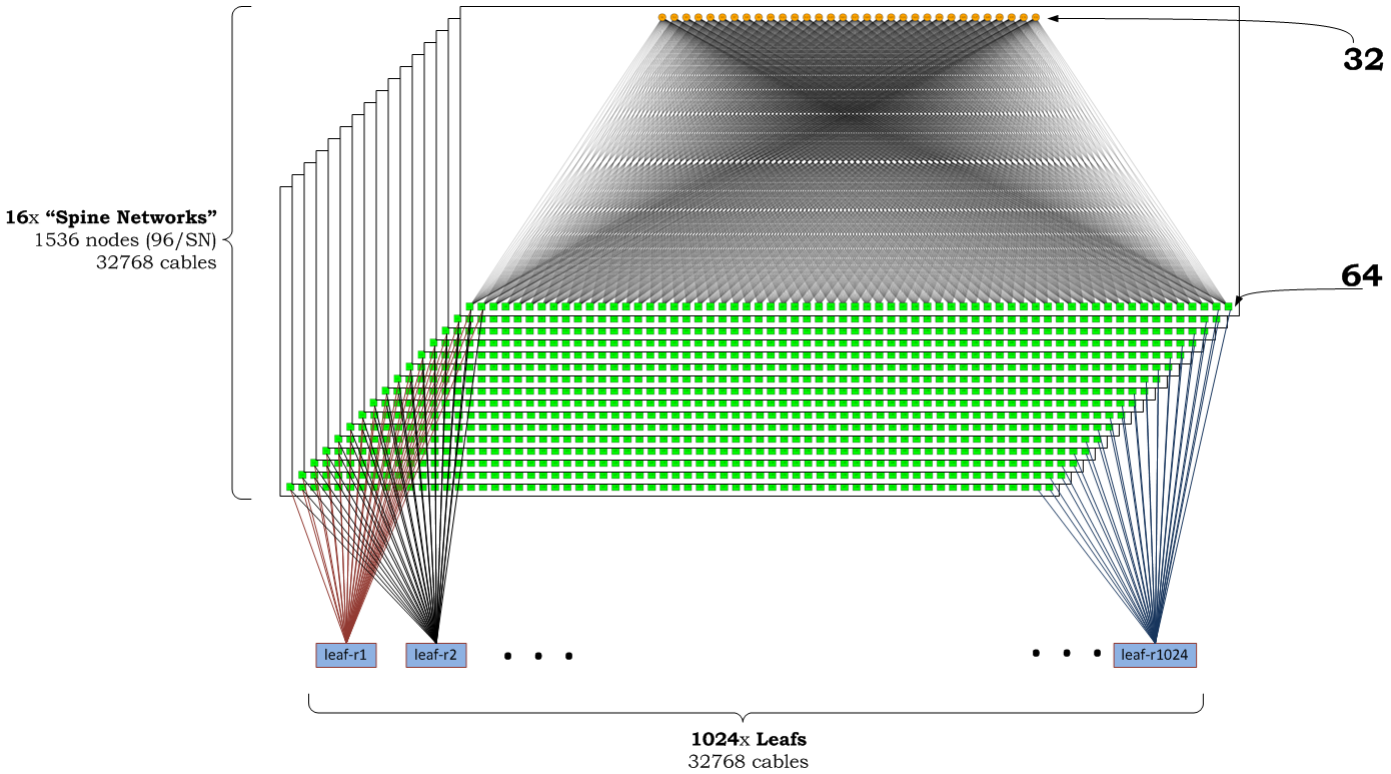


Table 1-2 shows an N3K as the building block (user to calculate x, y, and z for an N3K-based Fat Tree).

11. Costs, in this case, refers to the number of Fabric Boxes, Links, and Optics to achieve a particular end-host capacity.

Table 1-2

Topology	Building Block, Ports/Box	Fabric Boxes	Fabric Links	Total End-hosts
Fat Tree	64	x	y	z
Clos-3	64	96 (32_{spines}+64_{leafs})	2048	2048
Clos-5	64	5120 ($(96*32)_{spines}$+$(32*64)_{leafs}$)	131072 (2048*32+2048*32)	65536

Table 1-3 shows a modification for the CLOS-5 case that might employ a 16-wide "Spine" rather than a 32-wide "Spine" (Spine, in the CLOS-5 sense, means that each Spine "node" is comprised of a 3-stage Clos), thus each Leaf has 2 connections/Spine. In other words, you cut the number of devices and end-hosts in half.

Table 1-3

Topology	Building Block, Ports/Box	Fabric Boxes	Fabric Links	Total End-hosts
Clos-5'	64	2560 ($(96*16)_{spines}$+$(32*32)_{leafs}$)	131072 (2048*32+2048*32)	32768

Figure 1-14 represents such a topology.

Figure 1-14 Modified 5-Stage Clos Topology

It cannot be overstated the importance of also considering the amount of cabling, the cost of cabling, and the quantity and cost of optics in large Clos topologies!

The width of a Spine is determined by uplink capacity of the platform chosen for the Leaf layer. In the Reference Archicture, the N3064 is used to build the Leaf layer. To be 1:1 oversubscribed the N3K needs to have 32x 10G upstream and 32x 10G edge-facing. In other words, each Leaf layer device can support racks of 32x 10G attached servers or less. This also means that the Spine needs to be 32 nodes wide.

Note In Figure 1-14, there is latitude in how one defines a Spine "node". For example, there might be 16 nodes, but each Leaf uses 2x 10G ports to connect to each Spine node. For simplicity, a Spine in the strict sense, meaning that for each Leaf uplink there must be a discrete node, is what is used. The size of the Spine node will determine the number of Leafs the Clos network can support.

With real-world gear we construct the Clos with N3Ks (32 ports for servers each) and N7Ks+F2 (768 ports for Leafs, which means there are a total of 768 Leafs) as Leafs and Spines respectively. This means a total of 24,456 10G ports are available to interconnect servers at the cost of 800 devices, 24,456 cables, and 48912 10G optical transceivers.

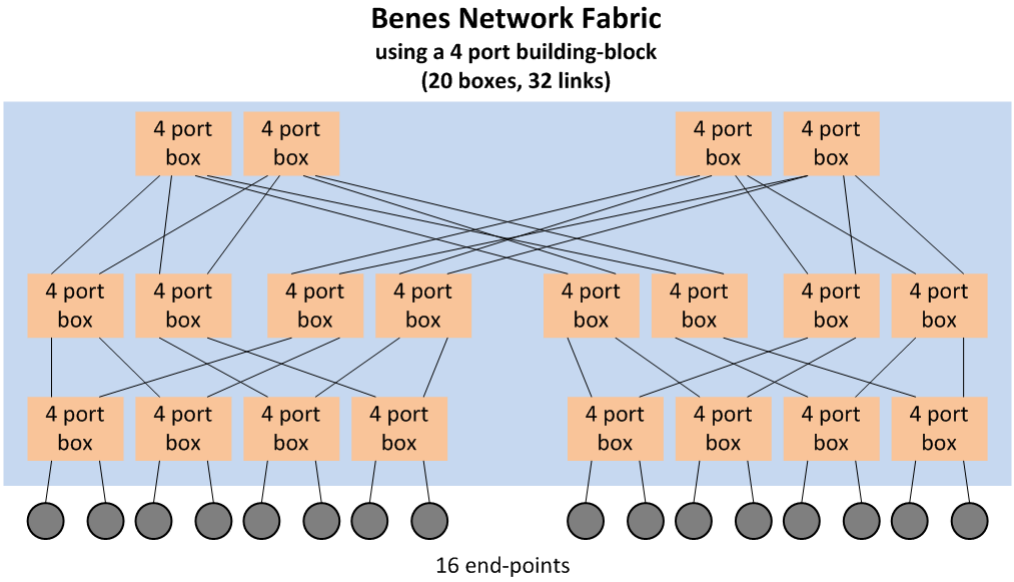
Because of limited rack real-estate, power, and hardware availablility, the test topology employs a 16-wide Spine with 20 Leafs, each Leaf supporting at most 16 servers/rack (this leaves a total of 32 ports on the N3Ks unused/non-existent for the purposes of our testing).

The lab topology had 16 N7Ks as a spine, so it is a 16-wide Spine Clos architecture.

Other Topologies

Clos'es and Fat Trees are not the only topologies being researched by customers. While Clos'es are the most popular, other topologies under consideration include the Benes[12] network (Figure 1-15).

Figure 1-15 Benes Topology



Or 1-dimensional (ring) (Figure 1-16), 2 and 3-dimensional Toroid[13], (Figure 1-17 & Figure 1-18)and hypercubes (Figure 1-19)

Figure 1-16 1D Toroid (Ring)

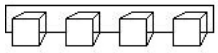
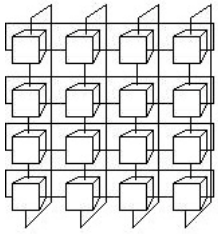


Figure 1-17 2D Toroid



12. http://en.wikipedia.org/wiki/Benes_network#Clos_networks_with_more_than_three_stages
13. http://en.wikipedia.org/wiki/Grid_network

Figure 1-18 3D Toroid

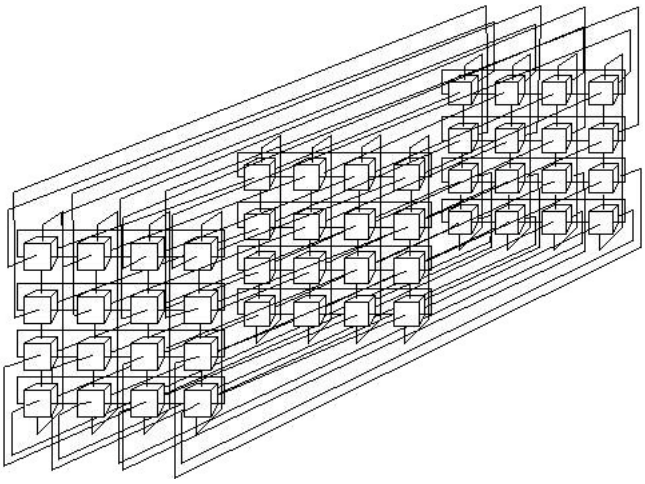
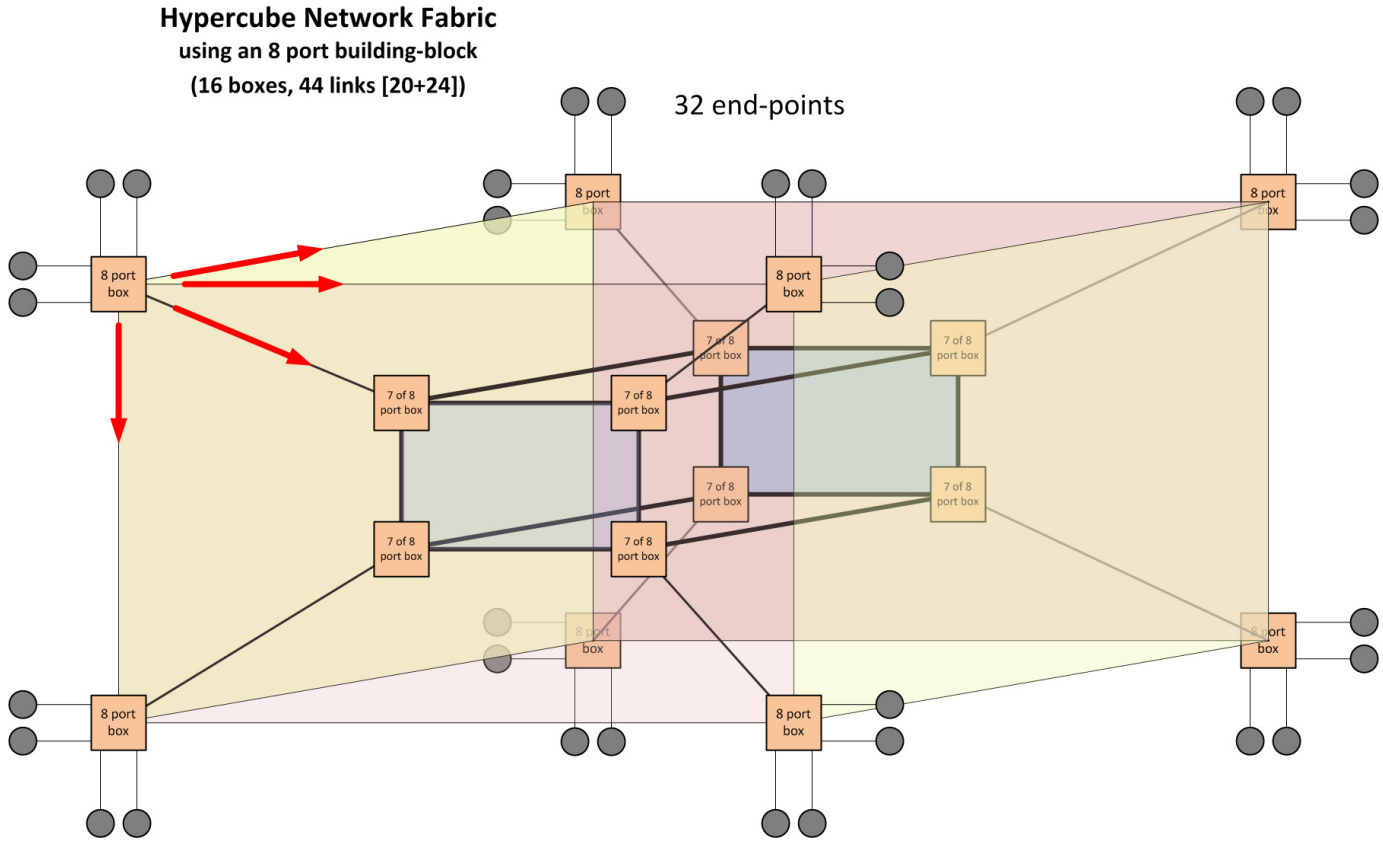


Figure 1-19 Hypercube



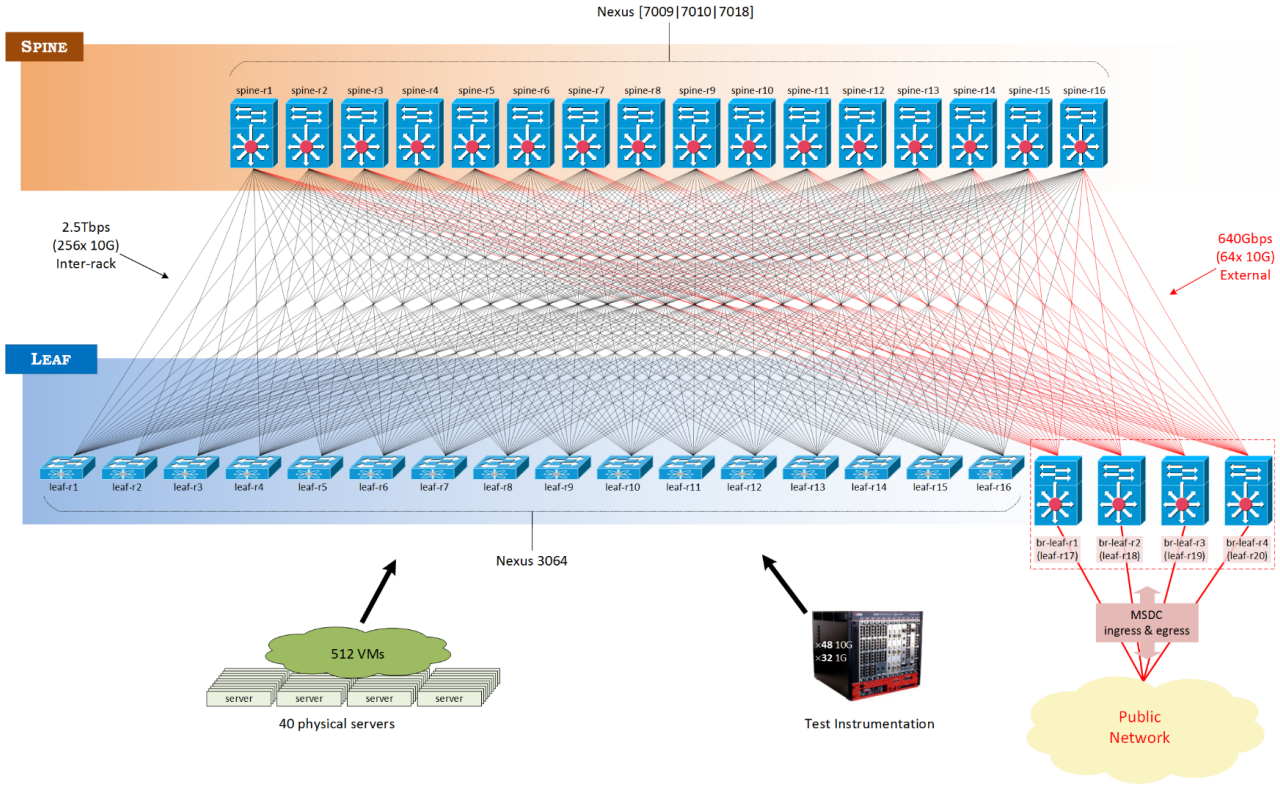
Each topology has pros and cons, and may be better suited to a particular application. For example, toroid and hypercube topologies favor intra-node, east-west traffic, but are terrible for north-south traffic. Toroid networks are in fact being utilized by some vendors, especially in the High Performance Computing (HPC) space. While a fuller discussion of alternative topologies is no doubt interesting to network engineers and architects trying to optimize the network to their applications, it is beyond the scope of the present document.

As of this writing, customers by and large gravitate to 3-stage Clos'es since they represent an acceptable balance of east-west vs north-south capability, implementation cost and complexity, number of cables and optics needed, and ease of operating. It goes without saying that Clos topologies are very well understood since they've been in use, such as in ASICs, for around 60 years and the theory is well developed as compared to more exotic topologies.

How a Clos Architecture Addresses Customer Concerns

The lab topology is based on a standard 3-stage folded Clos arrangement (Figure 1-20) . Leafs are N3064s and Spine devices are N7ks. Inter-Leaf bandwidth is 2.5Tbps, and Leaf to the outside world (Border) is 640Gbps. 4 Leafs, leaf-{r17-r20} are dedicated for Border functionality, br-leaf-{r1-r4}. For this phase of testing Border Leafs are merely injecting default route. Servers are attached directly to the Leaf layer, along with test instrumentation hardware such as IXIA. In order to create additional end-host nodes for the purpose of TCP testing, as well as to enable greater flexibility in subnetting arrangements, KVM VMs are configured across the server fleet.

Figure 1-20 SDU MSDC Test Topology



Refer to the bullet points in the "Design Tenets" section on page 1-5, to see how this type of topology can be use to meet those needs.

COST

Low-cost platforms for the Leaf, such as N3064, are based on commodity switching chipsets. For the Spine, F2 linecards on N7K are used. F2 is a higher density, high performance linecard with a smaller feature set than that of the M-series.

POWER

The testing did not focus as much on power as in the other areas of concern.

EAST-WEST BW

This area was of greatest concern when considering a MSDC topology for the lab. Utilizing the 3-stage folded Clos design afforded 2.5Tbps of east-west bandwidth, with each Leaf equally getting 160Gbps of the total.

TRANSPARENCY

An area of concern that is not discussed in this guide. It is expected that overlays may play an important part in achieving sufficient transparency between logical and physical networks, and between customer applications and the network.

HOMOGENEITY

There are only 2 platforms used in our MSDC topology, N3K and N7K. With only a small number of platform types it is expected that software provisioning, operations, and performance predictability will be achievable with present-day tools.

MULTIPATHING

The use of 16-way ECMP between the Leaf and Spine layers is key.[14] For a long time, IOS, as well as other network operating systems throughout the industry, were limited to 8 path descriptors for each prefix in the FIB. Modern platforms such as those based on NX-OS double the historical number to 16 as well as provide a roadmap to significantly greater ECMP parallelization. 64-way is currently available.[15] 128-way is not far off.[16]

CONTROL

This aspect of MSDC design tenets is met by programmability, both in initial provisioning (PoAP) and monitoring. This guide addresses both of these areas later in the document. However, it is acknowledged that “control” isn’t just about programmability and monitoring, but also may include the customer’s ability to influence a Vendor’s design, or even for large portions of the network operating system, for example, to be open to customer modifications and innovation. These last 2 aspects of control are not addressed in this guide.

14. As of this writing, NX-OS on Nexus 3064 is capable of 32-way ECMP.

15. http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps11541/data_sheet_c78-651097.html

16. This view is solely the author’s view and does not necessarily represent any official commitments by relevant BUs or Cisco at large.

Cisco Efforts in the MSDC Space

Cisco recognizes the significance of the MSDC market, and as such has created various internal initiatives to better position Cisco's expertise and innovation in this market-space. SDU's MSDC Engineering team represents one such initiative. As Cisco better understands customer requirements, it is able to become a more valued innovation partner with customers across the MSDC spectrum.

Applications

When discussing applications in MSDC environments, it is important to recognize not all MSDC operators actually control the applications running on their infrastructure. In the case of MSDC-scale public cloud service providers, for example, MSDC operators have little control over what applications tenants place into the cloud, when they run workloads, or how they tune their software.

Conversely in situations where public tenancy is not a constraint, operators tend to have very fine-grained control over the applications workloads. In many cases, the applications are written in-house rather than being purchased "off the shelf". Many use open source components such as databases, frameworks, message queues, or programming libraries. In such scenarios, applications vary widely between MSDC customers, but many share common threads:[17]

- Workloads are distributed across many nodes.
- Because of their distributed nature, many-to-one conversations among participating nodes are common.
- Applications that the data center owner controls are generally designed to tolerate (rather than avoid) failures in the infrastructure and middleware layers.
- Workloads can be sensitive to race conditions; but customers have made great efforts to minimize this with increased intelligence in the application space (independent of the network).

Exceptions certainly exist to the above application characteristics, but by-and-large represent the trends seen in present-day MSDCs.

Distribution

Distribution in MSDC environments may vary. Common distribution schemes include:

- In most cases, workloads are distributed among multiple racks (for scale and resiliency).
- In many cases, workloads are distributed among multiple independent clusters or pods (for manageability, resiliency, or availability reasons).
- In some cases, workloads are distributed among multiple data centers (for resiliency or proximity).

While the exact schemas for distribution may vary, some common rationales drive the design. A few common key characteristics which determine how workloads are distributed include:

- Performance
- Manageability
- Resiliency to failures (redundancy, fault isolation zones, etc)
- Proximity (to audience or other interacting components/data stores/applications)
- Scalability and elasticity

17. This information is based on extensive work with Account Teams as well as customer surveys.

- Cost

Workload Characterizations

Workloads vary between MSDC's based on applications being supported and how much control the customer has over the workload. For example, large cloud service providers hosting thousands of tenants have little control over workloads tenants deploy on top of the provider's IaaS or PaaS. Traffic in the network, disk I/O on end hosts, and other resource usage may be very inconsistent and hard to predict. Even in these cases, however, MSDC customers may have some distributed applications running atop the hardware that it has direct control over, such as orchestration systems, cloud operating systems, monitoring agents, and log analysis tools. Most such applications are designed to have as light a footprint as possible in order to preserve the maximum resources possible for sale to tenants.

By contrast, web portal or e-commerce providers may run applications designed in-house and therefore have flexibility in how to tune workloads that best suit underlying infrastructure. In such networks, tenants tend to be entities within the same corporation which actively collaborate on how best to use available resources. Workloads can be tuned for maximum efficiency, and elasticity may follow predictable trends (e-commerce sites might expect more load during holiday shopping season). Workloads in such customer environments can be loosely characterized as a series of interacting applications that together create a singular end-user SaaS experience. Characteristics of these systems reflect the purpose of the application. For example, distributed applications participating in the presentation of a website generate small packets (128-512 bytes) and short-lived conversations. Big data analysis workloads by contrast may have longer sustained flows as chunks of data are passed around and results of analysis returned.

Because of workload variability found in MSDC environments, it is strongly recommended that architects make careful study of the applications to be deployed before making infrastructure design decisions.

Provisioning

It doesn’t matter if a network is the highest performing network engineers may build for their applications if the network cannot get provisioned quickly and accurately. Timing is essential because MSDC network change often. Popular reasons for frequent network changes include Change Management (CM) proceedures or rapid scale growth to meet seasonal traffic bursts. It is not uncommon for customers to require entire datacenters be built within weeks of an initial request. Also, provisioning systems that easily integrate into customer’s own software mechanisms are the ones that get deployed.

Power On Auto Provisioning (PoAP)

PoAP, or Power on Auto Provisioning, is bootstrapping and kickstarting for switches. PoAP is an important fascilliatator for effective, timely, and programmable provisioning. It uses a DHCP client during initial boot-up to assign the device an IP address and then load software images and/or configuration. This is currently supported on the N3K, N5K, N7K lines.

Pieces of the Puzzle

In our lab topology, these are the pieces that make up the PoAP process:

- Nexus 3064s as Device Under Test (DUT).[18]

- Nexus 7Ks as DHCP relay.
- Cisco UCS server, configured with stock CentOS running isc-dhcpd, as DHCP server.
- Cisco UCS server, CentOS, as TFTP/FTP/SFTP/HTTP server.[19]
- Configuration scripts written in Python (although TCL could be used as well; however this guide completely focuses on modern Python).

PoAP in a MSDC

PoAP in MSDCs is important for the following reasons:

- MSDC's have a lot of devices to provision, especially Leafs.
- Configuring devices manually doesn't scale.
- MSDCs already use the network to bootstrap servers and would like to be able to treat network infrastructure in a similar manner.
- Speed of deployment.

PoAP Step-by-Step

Step 1 Device fully boots up.

Step 2 Empty config or 'boot poap enable' configured.

Step 3 All ports (including mgmt0) put into L3/routed mode, and DHCP Discover sent.

a. DHCP Discover has client-ID set to serial number found in 'show sprom backplane'

b. DHCP Discover has broadcast flag set

c. DHCP Discover requests TFTP server name/address and bootfile name options

Step 4 DHCP Offer randomly selected. DHCP Request sent, DHCP Ack received.

Step 5 Download PoAP script using TFTP/HTTP server and bootfile options from DHCP Offer.

Step 6 MD5sum verified and script executed.

Step 7 If script errors out, POAP restarts.

Step 8 If script completes successfully, Device is rebooted.[20]

PoAP Scripts

- Can be written in Python or TCL. Python is considered more modern.
- First line of script is md5sum over rest of script text.
  - #md5sum="0b96a4f2b9f876b4af97d4e1b212fabf"
  - Update with every script change!

18. http://www.cisco.com/en/US/docs/switches/datacenter/nexus3000/sw/fundamentals/503_U3_1/b_Nexus_3000_Fundamentals_Guide_Release_503_U3_1_chapter_0111.html

19. If the Python script does not require image or configuration download, then FTP/HTTP servers aren't required.

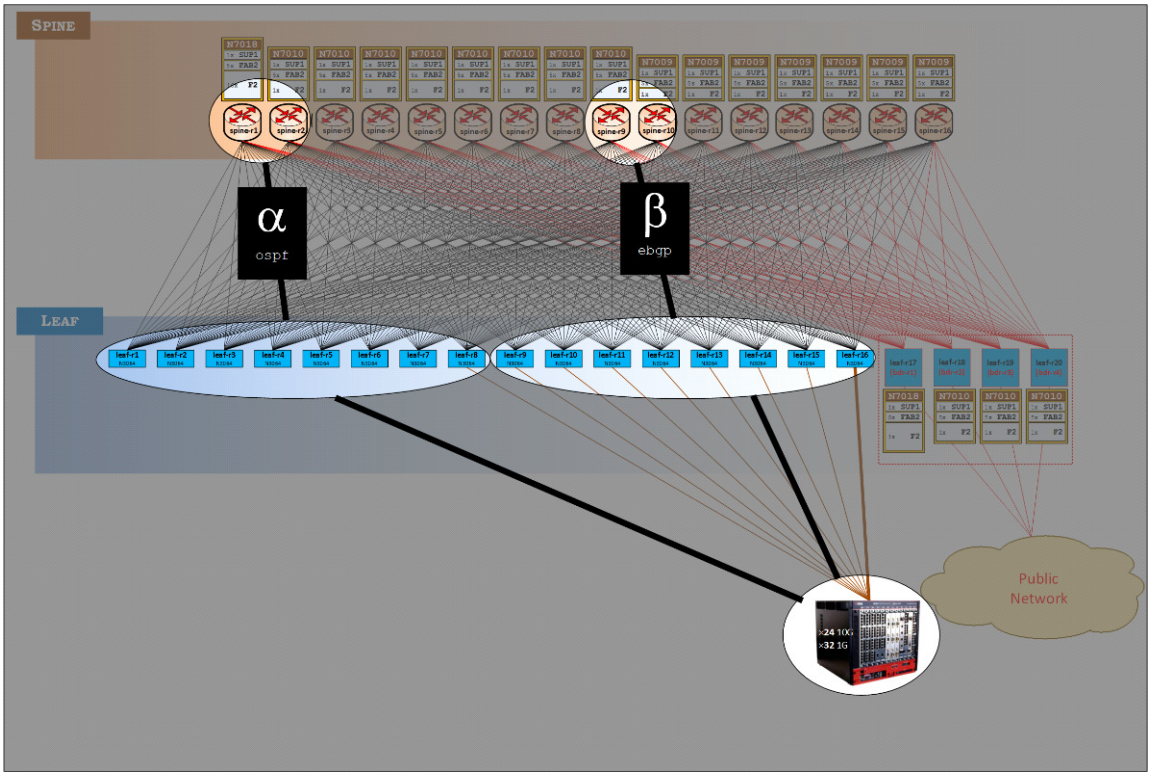
20. Configurations applied after the first reboot may be things like hardware profile portmode, hardware profile unicast, and system urpf.

- Sample scripts available on CCO download page (it's with kickstart images)
  - Upcoming scripting "community" for code sharing with/among customers to be available.[21]
- Full system initialization and libraries available
  - Script can be customized to do almost anything![22]
- Script troubleshooting is time consuming, therefore keep the script simple!
- PoAP process on switch is very basic, script does all the magic.

Topology and Infrastructure Setup

Throughout this phase of testing many logical topology changes were made, one change for each of 4 cycles; a, b, c, and d. PoAP was solely used to perform the configuration modifications automatically. Figure 1-21, Figure 1-22, Figure 1-23, and Figure 1-24 show progessive changes the topology underwent for the different cycles within this phase of testing. Each separate and logical topology is represented by a Greek letter to conveniently distinguish them. The minute details of each diagram aren't important, but rather the fact that logically separate topologies, using the same physical topology, were configured without human hands throughout the testing.

Figure 1-21 Two Parallel and Independent Topologies, one testing OSPF, the other BGP



21. https://github.com/datacenter

22. http://www.cisco.com/en/US/docs/switches/datacenter/nexus3000/sw/python/api/python_api.html

Figure 1-22 Two Topologies, Integrating Monitoring/Provisioning Servers

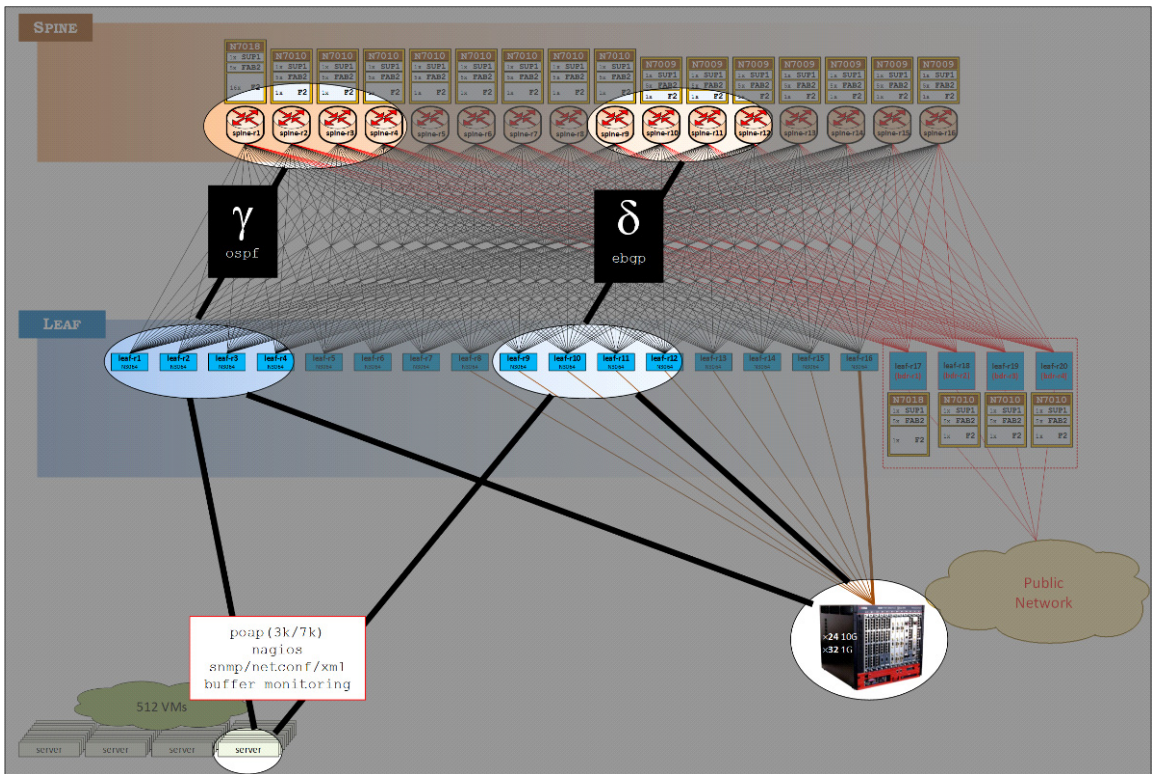


Figure 1-23 Two Topologies, Integrating End-Host Servers

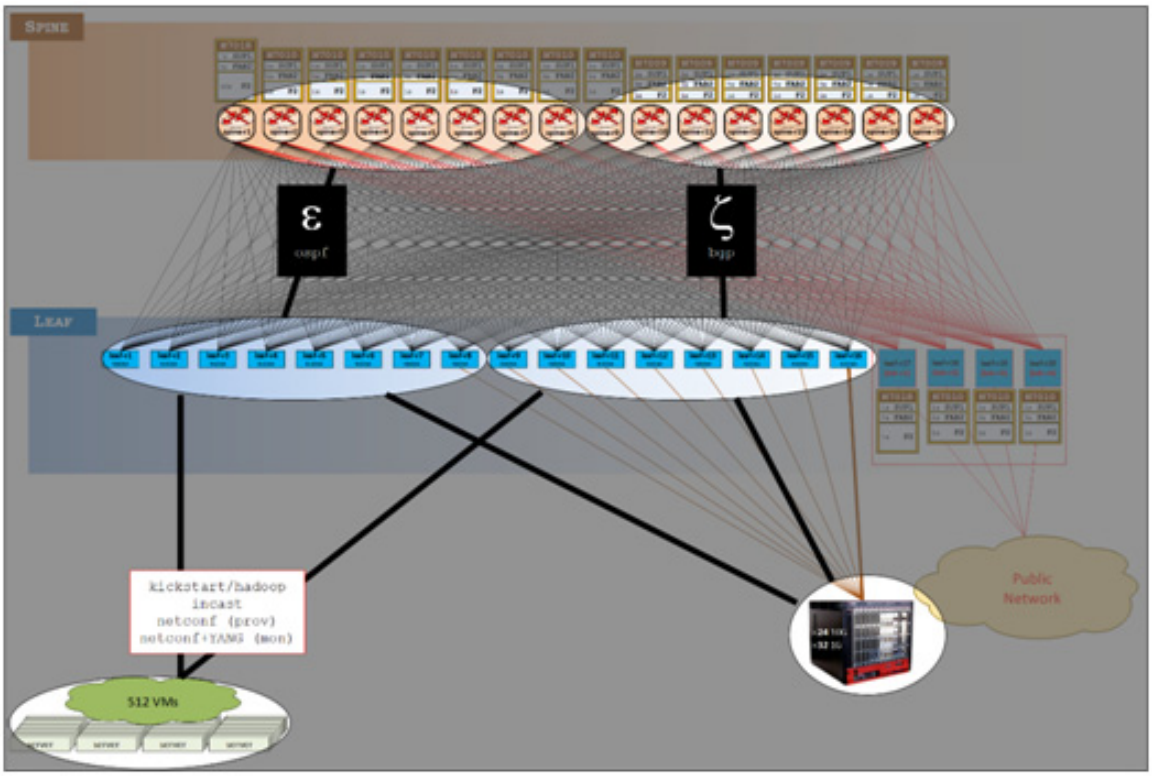
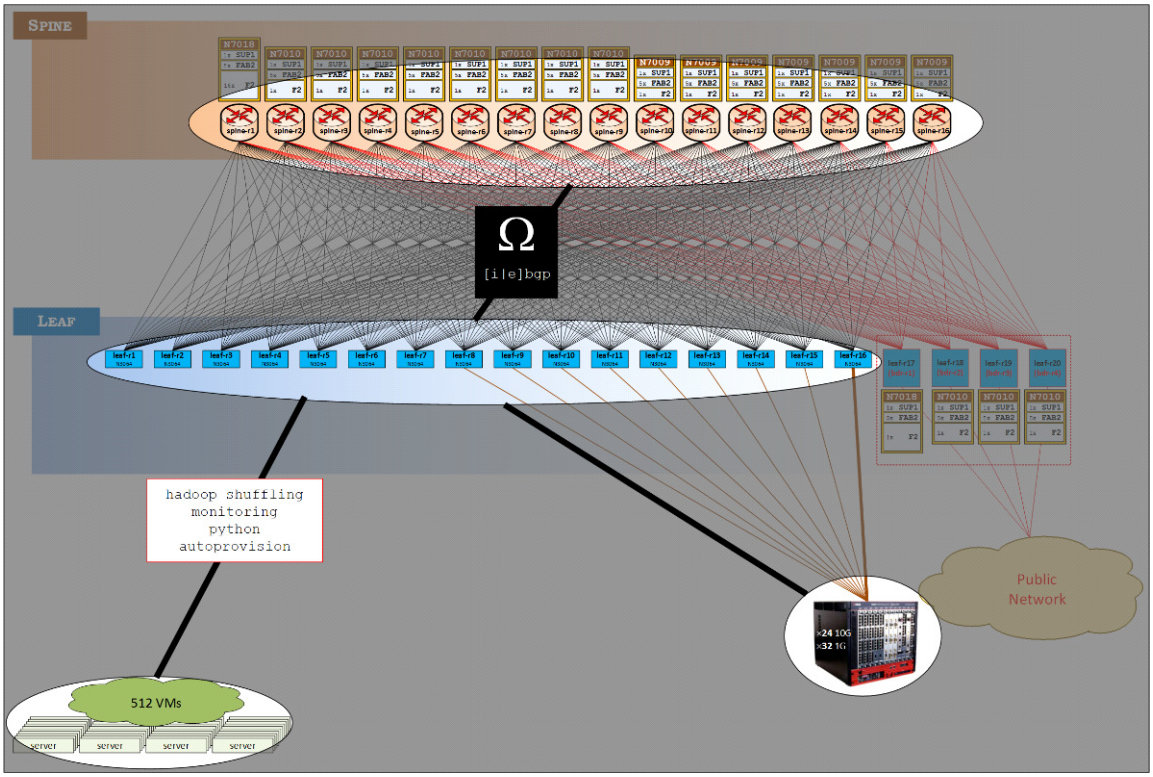


Figure 1-24 One Full Topology, Running Hadoop for Incast Testing



Monitoring

As networks grow, extracting useful telemetry in a timely manner is critical. Without relevant monitoring data it is impossible to manage MSDC-sized networks and stay profitable. MSDC customers want to do more with less, thus monitoring (with the requisite automation) is the glue which holds the infrastructure together. In addition to daily operations, monitoring provides essential information that allows for effective forward-planning and scaling, with a minimum number of network engineers.

Buffers

Statistics, gleaned from probes which monitor buffers, reveal important real-time MSDC characteristics. These characteristics show how traffic is distributed, how the infrastructure is performing, and are key indicators of where applications may suffer from blocking and queueing.

Why Congestion Matters

MSDCs are often home to highly distributed applications, and the network architecture is built with the applications in mind. It is the nature of these applications which leads to increased East-West bandwidth consumption across the data center. These distributed applications also have a high potential for creating bursty, many-to-one conversations such as what is observed in MapReduce jobs; these many-to-one conversations inevitably lead to congestion. Such systems are difficult to monitor since the visibility into performance and state is limited by the large size of the system.

Congestion leads to many possible problems, which includes:

- Session drops
- Job re-execution
- Job redistribution
- Data storage re-replication
- Reduced system capacity do to blacklisting
- Increased system performance churn due to nodes with fluctuating availability and performance.

Applications, or the infrastructure upon which they sit, can be optimized if operators have sufficient insight into congestion patterns. Such optimizations may include:

- Spin up or reduces nodes
- Rebalancing of storage
- Tweaking application traffic transmission patterns
- Operating System adjustments
- Alternate hardware can be chosen

One way to measure congestion is to observe buffering in relevant switches, that is, when buffering happens due to more traffic being sent than a port can handle, packets get dropped as those buffers fill. There are far fewer switches than compute nodes in MSDC distributed systems, thus making the switch a good vantage point to get an idea of what's happening in the system. This is especially true as the congestion data collected from switches can be correlated to events in the application, such as different phases of workload processing, or in the infrastructure, such as hardware failures. If congestion data can be gathered, then operators can model and optimize the system. If the supporting data can be analyzed quickly, system behavior changes might even be triggered on the fly.

Buffer Monitoring Challenges

Nexus 3064 switches use a 9MB shared buffer system. There are 8 unicast and 4 multicast queues per port. Deficit round-robin scheduling mechanism with multilevel schedule scheduling per-port and per-queue is used. 20% of the total buffer is dedicated to egress per-queue and per-port, the remaining 80% is dynamically shared. Instant and maximum cell utilization provided via CLI or XML (MSDC environments prefer XML), and EEM events can be triggered based on when thresholds are crossed.

When Watching Buffer Utilization Isn't Enough

In many cases, the worst problems happen when packets are dropped (not necessarily when they are buffered). Traffic elasticity in the application minimizes drops, but even if buffer space is available on the box doesn't mean it will be allocated to a congested port.

The admission control algorithm determines if a buffer cell is available for a port. Some buffer cells are allocated statically to each port even if not actually used. The CPU also may use some buffer capacity. If your app wants to react intelligently you want to know when a given port is about to run out of buffer space and cannot allocate more.

To be useful, buffer utilization must be correlated with other data from the system. Other data might come from the network, server, and/or applications, which means it's very useful to have metrics from all sources reporting into a single system. Correlation typically requires fine-grained data in order to see exactly how events in the software correspond to events in the network. Real-time graphing is useful when conditions are being actively observed, though this can be tricky at scale.

Traditionally, monitoring systems poll each node for data periodically. A classical Free Open Source Software (FOSS) example is Nagios. Usually the monitoring polling is done in serial, but can be parallelized to some degree (mod_gearman). Polling systems can only interact with switches via the mechanisms such as SNMP, SSH to CLI, Netconf, etc. In cases where CLI or Netconf are used, all command output must be sent to the monitoring system to be parsed and analyzed. Generally, these polling nodes don't tax CPU and Memory on the switch much (same as a user shell).

Pull vs Push Models

An example of the pull method is polling with SNMP at pre-set intervals. Polling based methods rarely provide sufficient granularity. For example, off-box polling via Nagios can generally fetch stats every 5-10s in previous testing. The ability of the polling server to open sessions and requests stats often actually limits the scale of such monitoring systems. Many of the readily available polling systems are harder to loadbalance (Nagios + mod_gearman incurs queue overhead and latency). Netconf over SSH isn't preferable, modern developers want REST or similar interfaces. There's a limit of 8 concurrent SSH sessions. General-purpose shell access requires crypt+auth. Not all NX-OS commands are available via XML. There are limited choices for open programming libraries. Inflexible transport method and verbose formatting (must transmit lots of useless XML over TCP). Lastly, XML is poorly formatted ("CLI wrapped in tags"), but is still preferable to unformatted CLI output. Poor granularity causes operators to miss bursts and limits the ability to correlate events and optimize the system.

On-switch Buffer Monitoring for Fine-grained Stats

To achieve real-time and for granular statistics gathering, we used a push model as follows.

- Created an on-switch monitoring daemon to provide buffer utilization stats to a sink at 1s or less intervals. SDU's python script is capable of publishing stats in pickle-protocol format to Graphite at approximately 0.17-0.20s intervals (though CPU utilization is excessive 40-50%). To avoid CPU hit, we published stats a 1s intervals.
- Explored other relevant stats via on-box monitoring to help correlate events. Adding interface counters and queuing drop counter was achieved, but quickly pushed reporting intervals about 1s. This can be mitigated somewhat with parallelization, though management and CPU/mem footprint is higher. SDU recommends choosing stats and commands carefully.
- Used on-switch monitoring and server stats to show correlation of Incast events and distributed system events in a 500+ node Hadoop cluster. SDU deployed 'collectd' to server nodes and had collectd sink data to the same Graphite server as the Leaf devices. This combination provided near real-time graphs allowing events to be seen as they happened with crafted MapReduce jobs.

The basic idea with on-switch monitoring is that a daemon running on each N3K periodically sends data to a central sink. **This changes the model from pull to push.** It also grants greater flexibility to Network Operations teams in how stats are gathered – doesn't force them to use SSH, but rather, TCP/UDP based protocols can be used. There's also a choice of encryption/authentication (or not) methods. SDU found that transporting of stats is more efficient than with polling mechanisms. The daemon opens a single TCP/UCP connection and sends data over it periodically… no need to tear down and setup repeatedly as in the Nagios model, for example. Only the actual stats are transported, not the surrounding XML/CLI overhead. The scalability of the central monitoring point increases due to such on-box monitoring. Central sink only needs to worry about receiving and storing data, not parsing or invoking filtering logic and device interaction. Parsing and filtering workloads are distributed to each switch rather than at a centralized point like in the Nagios model.

Doing on-switch monitoring allows operators to take advantage of Cisco features, such as PoAP (PoAP, page 2-1), or the ability to run native Python on the device, thus granting greater flexibility to developers to issue commands via an API. The NX-OS scheduler can also be used to keep the daemons running.

Deployment in Testing

Refer to Figure 1-20 on page 1-19 for details.

The daemon was written in Python with approximately 600 lines of code, and it used only modules provided by NX-OS – it wasn't necessary to load 3rd party libraries from bootflash, for example. The program sets up a TCP socket to a Graphite[23] receiver once then sends data via the Pickle[24] protocol at configurable intervals. Several CLI options are available to alter the frequency of stats collection, which stats are collected, where data is sent, and so forth.

SDU was able to demonstrate these capabilities with the on-switch system:

- Gathers data from both XML (when available) and raw CLI commands (when XML output not supported).
- Uses fast, built-in, modules like cPickly and Expat, to gather some stats, such as buffer cell utilization, and calculates other info not provided by NX-OS, such as % of buffer threshold used per port. As expected, there is a tradeoff between CPU impact and stat collection frequency moved to runtime via CLI arguments.

Graphite Setup

- Single server (8 cores/16 threads, 12GB RAM, 4-disk SATA3 RAID5 array).
- 8 carbon-cache instances fed by 1 carbon-relay daemon.
- Server receives stats from collectd [25]on each of 40 physical servers as well as on-switch monitoring daemons on each Leaf.
- Each collectd instance also provides stats for 14 VMs/server acting as Hadoop nodes.
- Incoming rate of over 36,000 metrics/sec possible, with 17,000-21,000 metrics/sec more the average.

Companion Script for NX-OS Scheduler

- NX-OS scheduler runs companion script every 1ms.
- Checks to see if daemon is running, starts it if not.
- Allows script to start at boot time, restart if crashed or killed.

Performance

- Buffer utilization stats every 0.18-0.20s possible, but uses 40-50% CPU!
- Buffer stats at approximately 1s intervals used negligible CPU, ranging from 2-5%.
- About 10.5MB footprint in memory.

Why Graphite and collectd? Both are high performance, open source components popular in cloud environments and elsewhere.

Graphite

- Apache2 license, originally created by Orbitz.
- Scales horizontally, graphs in near realtime even under load.
- Written in Python.

Accepts data in multiple easy-to-create formats.

23. http://graphite.wikidot.com/
24. https://graphite.readthedocs.org/en/latest/feeding-carbon.html#the-pickle-protocol
25. http://collectd.org/

Collectd

- GPLv2 license.
- Written in C.
- Low overhead on server nodes.
- Extensible via plugins.[26]
- Can send stats to Graphite via the Write Graphite plugin.[27]

Issues and Notes

Defects and caveats uncovered, and addressed, in SDU's work:

- **Issue 1**
  - No Python API for per-interface buffer stats…only switch-level stats.
  - Must therefore fall back to CLI for per-interface buffer usage data.
- **Issue 2**
  - No per-port maximum buffer utilization since cleared counter.
  - Will miss burst events lasting less than the frequency with which stats are collected.
- **Issue 3**
  - Scheduler config errors out during PoAP.
  - Means daemon can't be automatically started at bootup via a PoAP'd config.

Recommendations

The following recommendations are provided.

- When Possible, stick with Python modules already included on the N3K. Loading 3rd party pure-python modules from bootflash is possible, but provisioning and maintenance becomes more painful. This could be mitigated, however, by config management tools like Puppet, if it has support for Cisco devices.
- Balance granularity and CPU/memory footprint to specific needs. Adding more commands and stats to the daemon quickly lengthens the amount of time required to collect data and therefore the interval at which metrics can be published. The bulk of the overhead is in issuing commands and receiving output (not usually parsing or calculating). Parallelization can help by running multiple daemons or multiple instances of a daemon, each configured to gather only certain stats. This will certainly increase memory footprint, and may even increase CPU burden. But, doing parallelization make collecting different stats at different intervals easier.
- Use XML output from commands when possible. There is more reliable parsing available, as well as fast, especially with C-based Expat parser)
- Carefully select data sink, as it can become a choke point. SDU used Graphite, which scales relatively well, horizontally on multiple hosts and/or behind loadbalancers. Many MSDC customers have the resources and experience to design their own data sink systems.
- Avoid using per-interface command when possible, especially if you have a lot of interfaces to check. Parsing 'show queuing interface', once, is faster than issuing and parsing 64 individual 'show queuing interface x/y' commands.

26. https://collectd.org/wiki/index.php/Table_of_Plugins

27. https://collectd.org/wiki/index.php/Plugin:Write_Graphite

Caveats

The following caveats are provided.

- The on-switch approach still has some of the same pain points as other approaches. It still has to deal with issuing commands and parsing output. Full support for getting data via any one method other than CLI is lacking. Some commands have XML output, some don't. Some command have a Python API, most don't.
- The bottleneck for metric frequency is usually the CLI. Most of the bottlenecks SDU found were in how long it took to issue commands and get back output. For example:
  - **show queuing interface** has no XML output, takes ~1.1s
  - **show interface x/y | xml** on 29 interfaces took ~1.9s

Role of Virtualization

In this guide, virtualization plays only a supporting role and is not the focus of Phase 1. Virtualization was configured on the servers to provide infrastructure services, such as DHCP, TFTP, FTP/SFTP, and HTTP daemons. Virtualization was also used to create additional "nodes" in the Hadoop framework for the purpose of having finer-grained control over where workloads were placed.

Scale

Here we discuss best practice designs to raise the limits of the MSDC network. Refer to Fabric Protocol Scaling, page 2-8 for details on what the top churn elements are. BFD and routing protocols are discussed. Also, TCP Incast is introduced.

Fast Failure Detection (FFD)

The goal of FFD is to achieve sub-second detection of communication failures between two adjacent devices, on both the Control and Forwarding Plane. Below are common questions customers have when evaluating FFD methodologies in their environment

- What happens when there are intermediate L2 hops over L3 links?
- What happens when the protocol software fails?
- How fast will BFD detect unidirectional failures on ptp physical L3 links?

Operational simplicity and scalability is also another concern:

- Single set of timers that can apply to all routing protocols.
- Need to be lightweight and work with large number of peers without introducing instability (aggressive BGP timers increase CPU utilization).
- Media agnostic.

Issue with Tuning Routing Protocol Timers:

- Sub second convergence difficult to achieve.
- Different protocols require different set of timers—Not scalable.

- Indirect validation of forwarding plane failure. Not helpful in link down scenarios between p2p links.[28]
- May impact SSO and ISSU in redundant systems.
- High CPU overhead caused by the additional information carried in Routing protocol messages not needed for failure detection. Link utilization also increases as a result of frequent updates.
- Aggressive Routing protocol timers can lead to false positives under load/stress.

Quick BFD Overview

BFD is a lightweight hello protocol that provides fast detection of failure and supports routing protocols for IPv4, IPv6 and MPLS (RFC 5880). Salient features include:

- Detection in milliseconds
- Facilitates close alignment with hardware
- Three way handshake to bring up and teardown a session.
- Supports Authentication
- Active and Passive roles
- Demand and Asynchronous modes of operation
- Client notification upon state change of sessions. BFD Clients independently decide on action
- Protocol Version 0 and Version 1

BFD vs Protocol Timers in MSDC

Table 1-4 lists differences between BFD and protocol timers in MSDC.

Table 1-4 BFD vs Protocol Timers in MSDC

BFD	Timers
Single set of timer for all protocols.	Hello/dead timers different for each protocol and load.
Lightweight and can work with large number of peers without introducing instability (scalable).	Routing protocol messages carry superfluous information not needed for failure detection. Higher CPU load, link utilization and false positives can occur.
Distributed implementation (hellos sent from I/O module[1]).	Centralized implementation (hellos sent from SUP).
Failure notification to other protocols.	No failure notification to other protocols.
Interacts well under system HA events.	May impact SSO and ISSU in redundant systems (not as relevant in MSDCs).
Single L3 hop only[2].	Capable of single and multi L3 hop.
Sub-second failure detection.	Failure detection not sub-second.

1. For N7K implantation; not true for N3K.
2. The standard includes multi-hop, but Cisco implementation is only single-hop. Multi on roadmap.

28. On NXOS and many other products, link-down notification to protocols will always be faster than default dead-timer expiration.

General BFD Support

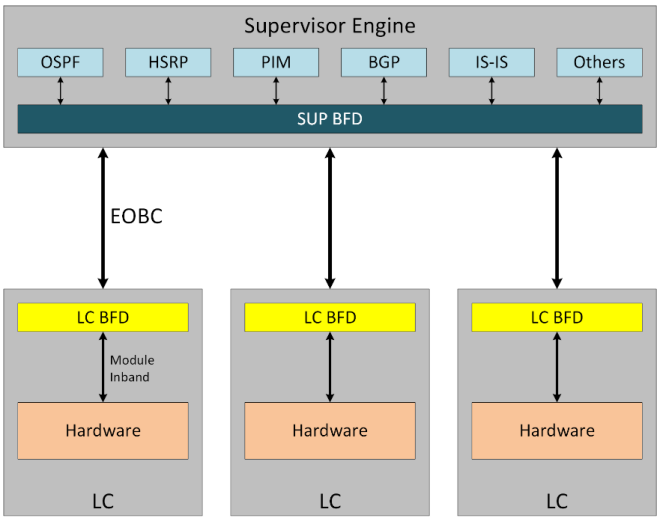
BFD was jointly developed by Cisco and Juniper. Many major vendors now support BFD, such as TLAB, Huawei, ALU. BFD at Cisco is implemented in IOS, IOS XR, and NX-OS with support for both BFD v0 and v1 packet formats. NX-OS implementation has been tested to interoperate with Cat6k, CRS, and various JUNOS platforms. Table 1-5 comparing the difference implementations across network Cisco's network OSes.

Table 1-5 BFD Support

Function	IOS	IOS-XR	NX-OS
Version Support	v0/v1	v0/v1	v1 only
Async and Echo Mode	Yes	Yes	Yes (except F1 LC)
Distributed Mode Implementation	Yes (GSR only)	Yes	Yes
BFD over Link Bundle[1]	Depends on Platform	Yes	Yes
Separate BFD Parameters per Protocol/Peer ID	Per interface	Yes	Global and Per interface. Applied to all sessions for all protocols.
Sessions Per LC[2]	NA	CRS—1k/7000pps 12k—100/1300pps	200 (50ms timer) 1k/system
Minimum Detection Timers	50ms	CRS—30ms 12K—150ms	50 ms 250 ms link bundle
IPv4 and IPv6	Yes	Yes	Yes (IPv6 on roadmap)
Single/Multi-hop Support	Single only	Yes	Single (multi on roadmap)
Active/Demand Mode	Active Mode only	Yes	Yes

1. As of this writing, BFD over bundles aren't standardized. Expect interoperability limitations.
2. F2 linecard (LC).

Figure 1-25 BFD, Systems on N7K



NX-OS N7K BFD Implementation

BFD functions as follows:

Step 1 Session request is received from the application (example OSPF, BGP).

Step 2 SUP-BFD process on the SUP determines type of port and ports operational parameters and IP address.

Step 3 A session discriminator is assigned and session context is created. A response is sent to the application.

Step 4 Finite State Machine (FSM) selects linecard where the session will be installed. ACLMGR programs required ACL (ACL's are required to redirect incoming BFD packets to appropriate line card CPU).

Step 5 Session control is passed to the linecard from the SUP.

Step 6 The LC-BFD process on LC sends notification to registered applications indicating session UP or DOWN status.

Step 7 If session state changes during session, BFD process on the LC will notify all registered applications.

BFD Recommendation

Based on the testing , here is a list of recommendations when using BFD:

- Use BFD if FFD tuning is needed.[29]
- Implement default timers for required protocol.
- Use Echo Mode to detect forwarding plane failure.
- Implement QoS to avoid false positive.
- BFD timers should be tuned to accommodate other distributed tasks such as netflow, Sflow.
- Incorporate validated limits as part of design consideration.[30] [31]

Graceful Restart

It is recommended to turn this feature off in an MSDC network. Graceful Restart allows the data-plane to continue forwarding packets should a control-plane-only failure occur (the routing protocol needs to restart but no links have changed). In a network with a stable control plane during steady-state, this is a very useful as it allows for hitless control-plane recovery. However, in a network with unstable control-plane during steady state, this feature can cause additional packet loss because the data-plane cannot handle addition updates during the restart interval.

Hiding Fabric Routes

The number of links between spine and leaf in an MSDC can be enormous. These routes alone can overwhelm a platforms FIB, without even adding the host facing subnets. There are several methods to work around this issue, each with their own pros and cons.

- BGP suppress connected routes

29. Cisco is constantly working to improve convergence with the N7K BU on OSPF. TCAM grooming can cause the forwarding plane to converge slowly. This is not specific to BFD.
30. http://www.cisco.com/en/US/docs/switches/datacenter/sw/6_x/nx-os/unicast/configuration/guide/l3_limits.html#wp1014867—N7K
31. http://www.cisco.com/en/US/docs/switches/datacenter/nexus3000/sw/configuration_limits/503_u2_2/b_Nexus3K_Configuration_Limits_for_Cisco_NXOS_Release_503_u2_2.html - N3k

- IPv6 Link Local Peering
- IP Unnumbered

TCP Incast

TCP Incast, also known as "TCP Throughput Collapse", a form of congestive collapse, is an extreme response in TCP implementations that results in gross under-utilization of link capacity in certain N:1 communication configurations.[32]

Packet loss, usually occurring at the last-hop network device, is a result of the N senders exceeding the capacity of the switch's internal buffering. Such packet-loss across a large fleet of senders may lead to TCP Global Synchronization (TGS), an undesirable condition where senders respond to packet losses by taking TCP timeouts in "lock-step". In stable networks, buffer queues are either usually empty or full; in bursty environments these limited queues are quickly overrun. A popular method for dealing with overrun queues is to enforce "tail drop". However when there are large numbers of [near] simultaneous senders, N, and the senders are sending to a single requestor, the resultant tail-drop packet-losses occur at roughly the same time. This in turn causes the sender's TCP automatic recovery mechanisms of congestion avoidance to kick in ("slow-start" and its variant and augmentations) at the same time. The net effect wasted bandwidth consumed that isn't doing much real work.[33]

Why A Concern in MSDC

Distributed systems, such as Hadoop clusters or large storage clusters, are canonical examples of systems affected by Incast. There are three primary drivers for SDU's research and testing in this area:

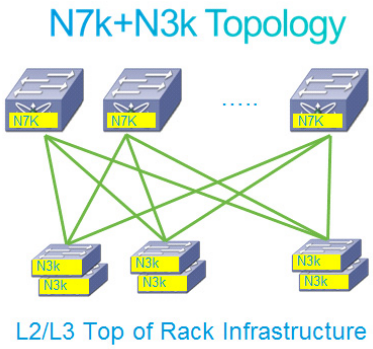
- Our Tier-1 MSDC Customers have expressed concern in this area.[34]
- There isn't much guidance and literature in the field, especially from Cisco, on how to deal with the Incast problem in MSDC contexts.
- We care about buffers and buffer-bloat since a 1993 proof, the key is appropriately sized buffers have predictable end-to-end delay from point A to Z. Predictability is important to MSDC Customers where infrastructure is such a dominating cost center.

Current Work at Cisco

Another team at Cisco has conducted BigData analysis using Hadoop (HDFS, MapReduce) across two topologies: N7K Spine/N5K Leaf, and N7K Spine/N3K Leaf. Since MSDCs do not use FEX or N5Ks, the N3K topology is most relevant (Figure 1-26).

32. Adapted from "http://radlab.cs.berkeley.edu/wiki/Incast".
33. Refer to "http://en.wikipedia.org/wiki/TCP_global_synchronization" for more details on TGS.
34. Account Teams from Facebook, Amazon, Yahoo!, Rackspace, and Microsoft have expressed interest and concern. This isn't necessarily a comprehensive list.

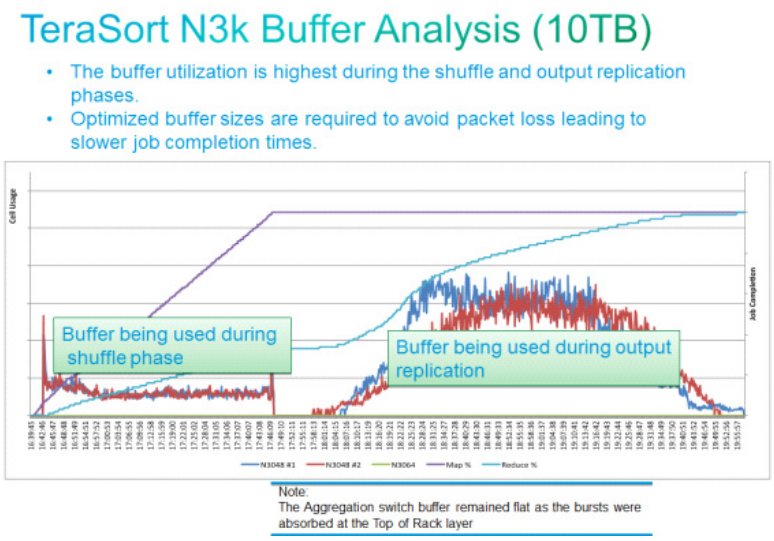
Figure 1-26 Topology from Prior Buffer Analysis Work



The team showed that with 10G attached servers there are fewer burdens on network buffering because servers will consume network data faster. Thus CPU, memory, and storage I/O becomes the bottleneck as opposed to network buffers (Figure 1-27). Work done in support of this guide differs from what has previously been done in two ways:

1. Testing used Hadoop as a way to generalize Incast conditions rather than analyzing Hadoop itself.
2. Testing builds upon work that has already been done by introducing a broader class of failure and churn scenarios and observe how the network behaves, for example, what happens when you fail larger groups of servers, or have gross-level rack failures?

Figure 1-27 Hadoop and Buffer Analysis with 10G



Industry Research Gaps This Testing Addresses

TCP Incast testing builds upon work that has already been done and focuses on these issues:

- More generalized—not focused on Hadoop itself, but rather provides a tool to generalize Incast and complete system impact.
- Failure injection and job concurrency (multiple jobs at various stages when failure occurs).
- Single rack and Multi-rack failures (focus on multi-rack) – induce “cascading failure(s)”.
- Detection of an Incast event from the perspectives of both network and servers.