Clos IP Fabrics with QFX5100 Switches

Building Flexible, Programmable Data Center Networks Using Layer 3 Protocols and Overlay Networking

Table of Contents

List of Figures

Executive Summary

This paper describes how to build large IP fabrics using the Juniper Networks® QFX5100 line of switches. This paper is written by network engineers for network engineers and covers why you would want to build a Clos IP fabric and how to build, configure, and verify Clos IP networks using the QFX5100 Switch. Example configurations are included, enabling you to architect and configure your Clos IP network based on the examples.

Introduction

Everywhere you look in the networking world you see something about IP fabrics or Clos networks. Something is brewing, but what is it? What's driving the need for IP fabrics? If an IP fabric is the answer, then what is the problem we're trying to solve?

Many over-the-top (OTT) companies have been building large IP fabrics for a long time, but they have rarely received any attention for doing so. Such companies generally have no need for compute virtualization and write their applications in such a way that high availability is built into the application. With intelligent applications and no compute virtualization, it makes a lot of sense to build an IP fabric using nothing but Layer 3 protocols. Layer 2 has traditionally been a point of weakness in data centers in terms of scale and high availability. It's a difficult problem to solve when you have to flood traffic across a large set of devices and prevent loops on Ethernet frames that don't natively have a time-to-live field.

If companies have been building large IP fabrics for a long time, then why have IP fabrics received so much attention lately? The answer is because of overlay networking in the data center. The problem being solved is twofold: 1) network agility and 2) simplifying the network. Overlay networking combined with IP fabrics in the data center is the answer.

Overlay Networking

One of the first design considerations in a next-generation data center is do you need to centrally orchestrate all resources in the data center so that applications can be deployed within seconds? The follow-up question is do you currently virtualize your data center compute and storage with hypervisors or cloud management platforms? If the answer is "yes" to these questions, you must consider an overlay architecture when it comes to the data center network.

Seeing how compute and storage have already been virtualized, the next step is to virtualize the data center network. Using an overlay architecture in the data center allows you to decouple physical hardware from the network, which is one of the key tenets of virtualization. Decoupling the network from the physical hardware allows the data center network to be programmatically provisioned within seconds.

The second benefit of overlay networking is that it supports both Layer 2 and Layer 3 transport between VMs and servers, which is very compelling to traditional IT data centers. The third benefit is that it has a much larger scale than traditional VLANs and supports up to 16.7 million tenants. Two great examples of products that support overlay architectures are Juniper Networks Contrail and VMware NSX.

Moving to an overlay architecture places a different "network tax" on the data center. Traditionally, when servers and virtual machines are connected to a network, they each consume a MAC address and host route entry in the network. However, in an overlay architecture, only the virtual tunnel endpoints (VTEPs) consume a MAC address and host route entry in the network. All virtual machine (VM) and server traffic is now encapsulated between VTEPs, and the MAC address and host route of each VM and server aren't visible to the underlying networking equipment. The MAC address and host route scale have been moved from the physical network hardware into the hypervisor.

Bare Metal Servers

It's rare to find a data center that has virtualized 100% of its compute resources. There's always a subset of servers that cannot be virtualized due to performance, compliance, or any other number of reasons. This raises an interesting question: If 80% of the servers in the data center are virtualized and take advantage of an overlay architecture, how do you provide connectivity to the other 20% of physical servers?

Overlay architectures support several mechanisms to provide connectivity to physical servers. The most common option is to embed a VTEP into the physical access switch as shown in Figure 1.

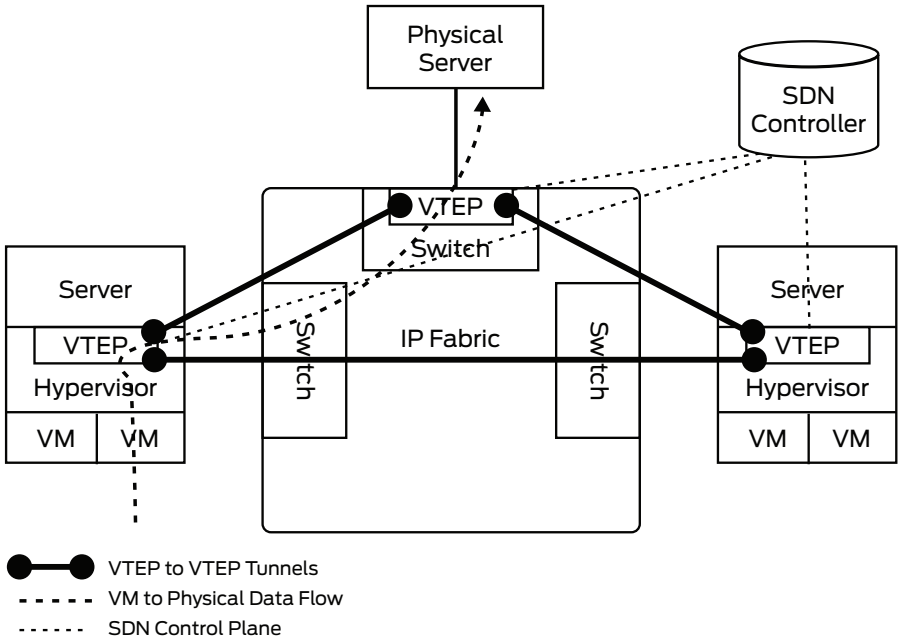

Figure 1: Virtual to physical data flow in an overlay architecture

Each server on the left and right of the IP fabric, shown in Figure 1, has been virtualized with a hypervisor. Each hypervisor has a VTEP inside that handles the encapsulation of data plane traffic between virtual machines. Each VTEP also handles MAC address learning, provisioning of new virtual networks, and other configuration changes. The server on top of the IP fabric is a simple physical server, but doesn't have any VTEP capabilities of its own. In order for the physical server to participate in the overlay architecture, it needs something to encapsulate the data plane traffic and perform MAC address learning. Being able to handle the VTEP role inside of an access switch simplifies the overlay architecture. Now each access switch that has physical servers connected to it can simply perform the overlay encapsulation and control plane on behalf of the physical server. From the point of view of the physical server, it simply just sends traffic into the network without having to do anything else.

IP Fabric

In summary, there are two primary drivers for an IP fabric: OTT companies with simple Layer 3 requirements, and the introduction of overlay networking that uses the IP fabric as a foundational underlay. Let's start off by taking a look at the requirements of overlay networking in the data center and how an IP fabric can meet and exceed the requirements.

All VM and server MAC addresses, traffic, and flooding can be encapsulated between VTEPs in an overlay architecture. The only network requirement of VTEPs is Layer 3 connectivity. Creating a network that's able to meet the networking requirements is very straightforward. The challenge is how to design a transport architecture that's able to scale in a linear fashion as the size increases. A very similar problem was solved back in 1953 in the telecommunications industry. Charles Clos invented a method to create a multistage network that is able to grow beyond the largest switch in the network, as shown in Figure 2.

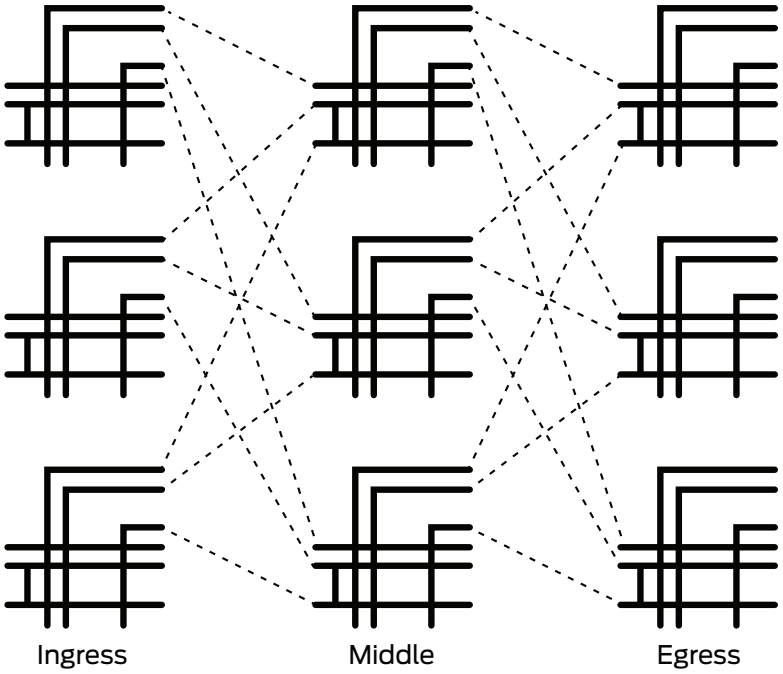

Figure 2: Charles Clos' multistage topology

The advantage to a Clos topology is that it's nonblocking and allows predictable performance and scaling characteristics. Figure 2 represents a three-stage Clos network: ingress, middle, and egress.

We can take the same principles of the Clos network and apply them to creating an IP fabric. Many networks are already designed like this and are often referred to as spine and leaf networks, as illustrated in Figure 3.

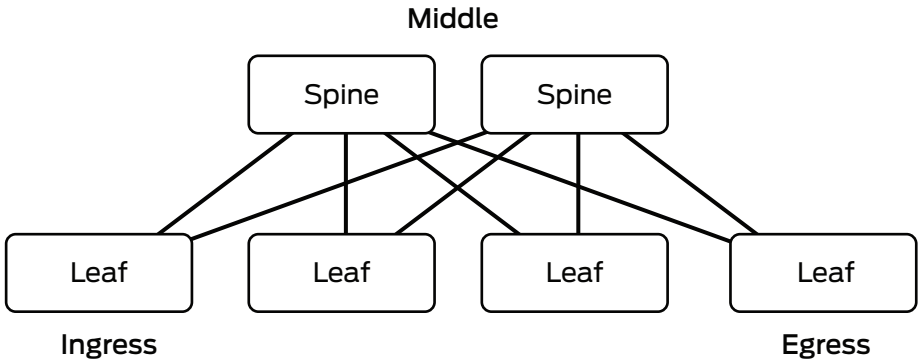

Figure 3: Spine and leaf topology

A spine and leaf network is actually identical to a three-stage Clos network—it is sometimes referred to as a folded three-stage Clos network because the ingress and egress points are folded back on top of each other, as illustrated in Figure 3. In this example, the spine switches are simple Layer 3 switches, and the leaves are top-of-rack switches that provide connectivity to the servers and VTEPs.

The secret to scaling up the number of ports in a Clos network is adjusting two values—the width of the spine and the oversubscription ratio. The wider the spine, the more leaves the IP fabric can support. The more oversubscription placed into the leaves, the larger the IP fabric as well. Let's review some example topologies in detail to understand how the pieces are put together and the end results.

768 x 10 Gigabit Ethernet Virtual Chassis Fabric

The first example is a new Juniper technology called Virtual Chassis Fabric, which enables you to create a three-stage IP fabric using a set of QFX5100 switches that's managed as a single device. As of Juniper Networks Junos® operating system Release 13.2, the maximum number of switches in a Virtual Chassis Fabric is 20 as shown in Figure 4.

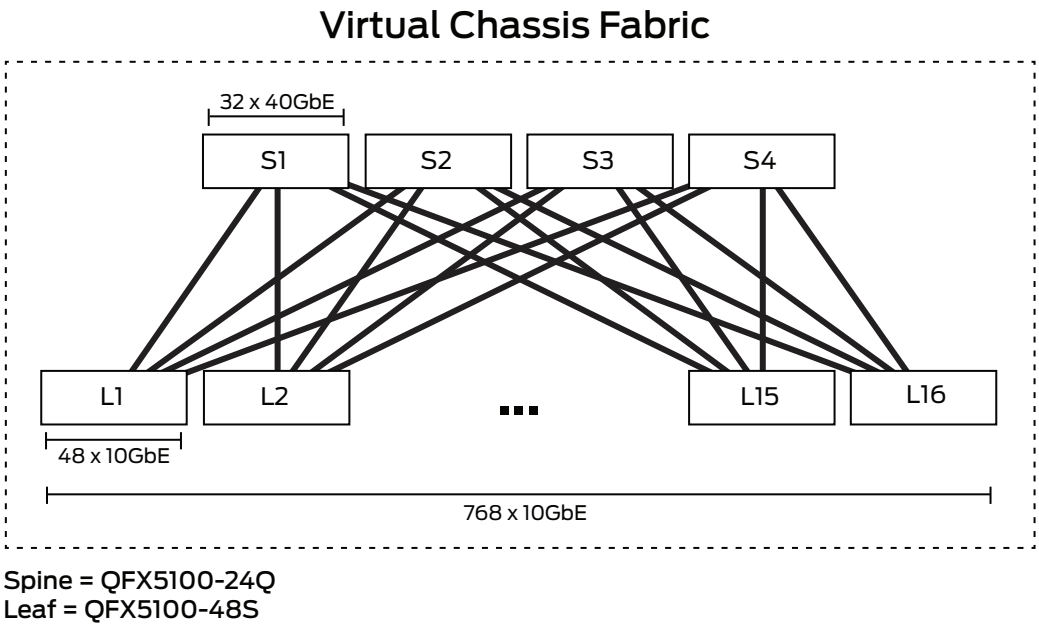

Figure 4: Virtual Chassis Fabric of 768 x 10GbE ports

In this example, the spine is constructed of four QFX5100-24Q switches. Each switch supports up to 32 x 40 Gigabit Ethernet (GbE) interfaces. The leaves are built using the QFX5100-48S, which supports 48 x 10GbE and 6 x 40GbE interfaces. Each leaf uses 4 x 40GbE interfaces as uplinks, with one link going to each spine, as illustrated in Figure 4. This creates an oversubscription of 480:160 or 3:1 per leaf. Because Virtual Chassis Fabric only supports 20 switches, we have a total of 4 spine switches and 16 leaf switches, for a total of 20 switches. Each leaf supports 48 x 10GbE interfaces. Because there are 16 leaves total, this brings the total port count up to 768 x 10GbE with 3:1 oversubscription.

If your scaling requirements exceed the capacity of Virtual Chassis Fabric, that's not a problem. The next option is to create a simple three-stage IP fabric that is able to scale to thousands of ports.

3072 x 10GbE IP Fabric

The next option is creating a simple three-stage IP fabric using the QFX5100-24Q and QFX5100-96S, but this time we won't use Virtual Chassis Fabric. The QFX5100-24S is a 32 x 40GbE switch, and the QFX5100-96S is a 96 x 10GbE and 8 x 40GbE switch. Combining the QFX5100-24Q and the QFX5100-96S creates an IP fabric of usable 3072 x 10GbE ports, as shown in Figure 5.

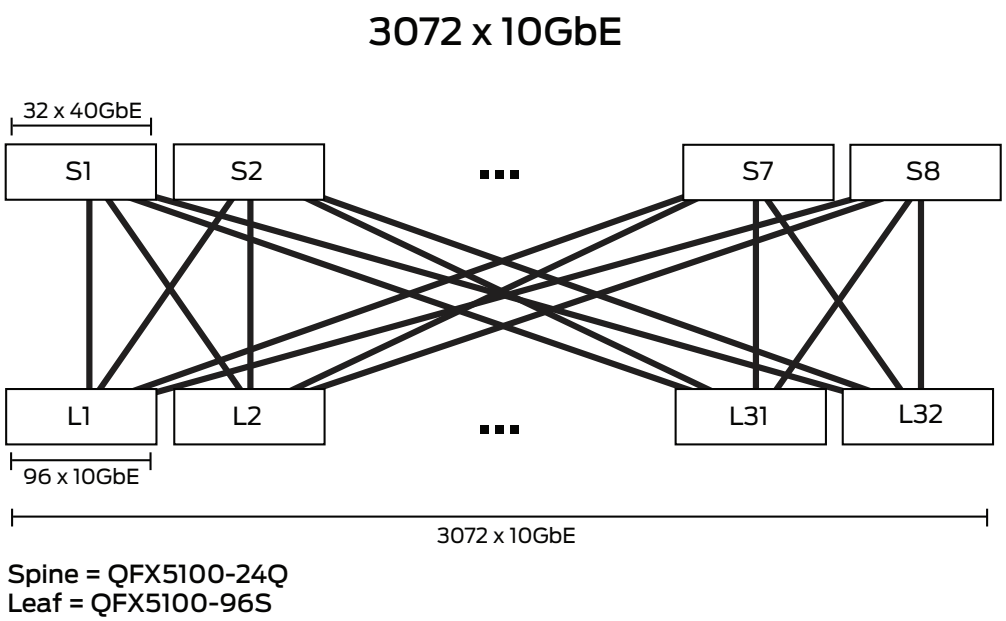

Figure 5: 3072 x 10GbE IP fabric topology

The leaves are constructed using the QFX5100-96S, and 8 x 40GbE interfaces are used as uplinks into the spine. Because each leaf has eight uplinks into the spine, the maximum width of the spine is eight. Each 40GbE interface per leaf connects to a separate spine—thus, each leaf consumes one 40GbE interface per spine. To calculate the maximum size of the IP fabric, you have to multiply the number of server interfaces on the leaf by the number of leaves supported by the spine. In this example, the spine can support 32 leaves and each leaf can support 96 ports of 10GbE. This is a total of 3072 usable 10GbE ports with a 3:1 oversubscription ratio.

Control Plane Options

One of the big benefits to using Virtual Chassis Fabric is that you don't have to worry about the underlying control plane protocols of the IP fabric. It just works. However, if you need to create a network that exceeds the scale of Virtual Chassis Fabric, you need to take a look at what the control plane options are.

One of the fundamental requirements for creating an IP fabric is the distribution of prefixes. Each leaf needs to send and receive IP routing information with all of the other leaves in the IP fabric. The question now becomes what are the options for an IP fabric control plane and which is the best? We can start by reviewing the fundamental requirements of an IP fabric and mapping the results to the control plane options as shown in Table 1.

Table 1: IP Fabric Requirements and Control Plane Options

Requirement	OSPF	IS-IS	BGP
Advertise Prefixes	Yes	Yes	Yes
Scale	Limited	Limited	Extensive
Traffic Engineering	Limited	Limited	Extensive
Traffic Tagging	Limited	Limited	Extensive
Multivendor Stability	Yes	Yes	Extensive

The most common options for the control plane of an IP fabric are OSPF, IS-IS, and BGP. Each protocol can fundamentally advertise prefixes, but the protocols vary in terms of scale and features. OSPF and IS-IS use a flooding technique to send updates and other routing information. Creating areas can help limit the amount of flooding, but then you start to lose the benefits of an SPF routing protocol. On the other hand, BGP was created from the ground up to support a large number of prefixes and peering points. The best use case in the world to prove this point is the Internet.

The ability to shift traffic around in an IP fabric could be useful.. For example, you could steer traffic around a specific spine switch while it's in maintenance. OSPF and IS-IS have limited traffic engineering and traffic tagging capabilities. Again, BGP was designed from the ground up to support extensive traffic engineering and tagging with features such as local preference, MEDs, and extended communities.

One of the interesting side effects of building a large IP fabric is that it's generally done iteratively and over time. It is common to see multiple vendors creating a single IP fabric. Although OSPF and IS-IS work well across multiple vendors, the real winner here is BGP. Again, the best use case in the world is the Internet. It consists of a huge number of vendors, equipment, and other variables, but they all use BGP as the control plane protocol to advertise prefixes, perform traffic engineering, and tag traffic.

Because of the scale, traffic tagging, and multivendor stability, BGP is the best choice when selecting a control plane protocol for an IP fabric. The next question is how do you design BGP in an IP fabric?

BGP Design

One of the first decisions to make is whether to use IBGP or EBGP. The very nature of an IP fabric is based off equal cost multipath (ECMP). One of the design considerations is how does each option handle ECMP? By default, EBGP supports ECMP without a problem. However, IBGP requires a BGP route reflector and the AddPath feature to fully support ECMP.

Let's take a closer look at the EBGP design in an IP fabric. Each switch represents a different autonomous system number, and each leaf has to peer with every other spine in the IP fabric, as shown in Figure 6.

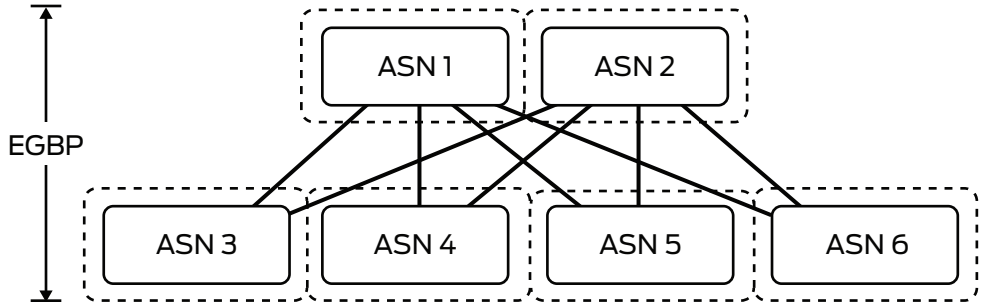

Figure 6: Using EBGP in an IP fabric

Using EBGP in an IP fabric is very simple and straightforward. It also lends itself well to traffic engineering by using local preference and autonomous system padding techniques.

Designing IBGP in an IP fabric is a bit different since IBGP requires that all switches peer with every other device within the IP fabric. To mitigate the burden of having to peer with every other device in the IP fabric, we can use inline BGP route reflectors in the spine of the network, as illustrated in Figure 7. The problem with standard BGP route reflection is that it only reflects the best prefix and doesn't lend itself well to ECMP. In order to enable full ECMP, we have to use the BGP AddPath feature, which provides additional ECMP paths into the BGP advertisements between the route reflector and clients.

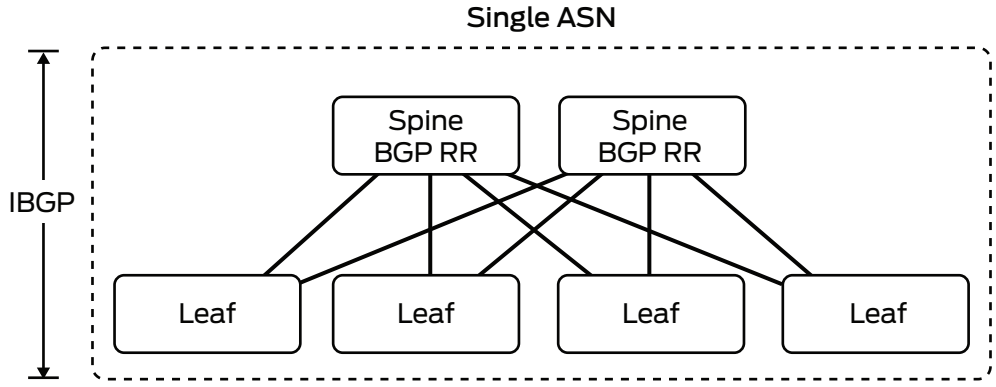

Figure 7: Using IBGP in an IP fabric

The QFX5100 Switch supports both BGP design options of IBGP and EBGP. Both options work equally well. However, EBGP's design and implementation are simpler. It is always more stable to go back to designing a machine with fewer moving parts—it's our recommendation to use EBGP when creating simple three-stage IP fabrics. There's no need to worry about BGP route reflection and AddPath.

Implementation Requirements

There is a set of requirements that has to be worked out in order to create a blueprint for an IP fabric. At a high level it revolves around IP address management (IPAM) and BGP assignments. Let's break the requirements out into the next level of detail.

- **Base IP prefix**—All of the IP address assignments made within the IP fabric must originate from a common base IP prefix. It's critical that the base IP prefix has enough address space to hold all of the point-to-point addressing, as well as loopback addressing of each switch in the IP fabric.
- **Point-to-point network mask**—Each leaf is connected to every spine in the IP fabric. These connections are referred to as the point-to-point links. The network mask used in the point-to-point links determines how much of the base IP prefix is used. For example, using a 30-bit network mask uses twice as much space as using a 31-bit network mask.
- **Point-to-point IP addresses**—For every point-to-point connection, each switch must have an IP address assignment. You need to decide if the spine receives the lower-numbered or the higher-numbered IP address assignment. This is more of a cosmetic decision and doesn't impact the functionality of the IP fabric.
- **Server-facing IP prefix**—In order to provide Layer 3 gateway services to VTEPs, the leaves must have a consistent IP prefix that's used for server-facing traffic. This is separate from the base IP prefix used to construct the IP fabric. The server-facing IP prefix must be large enough to support the address requirements of each leaf in the IP fabric. For example, if each leaf required a 24-bit subnet and there were 512 leaves, the minimum server-facing IP prefix would have to be at least 15 bits, such as 192.168.0.0/15, which would allow you to have 512 24-bit subnets. Each leaf would have a 24-bit subnet, such as 192.168.0.0/24, and each one could use the first IP address for Layer 2 gateway services, such as 192.168.0.1/24.
- **Loopback addressing**—Each switch in the IP fabric needs a single loopback address using a 32-bit mask. The loopback address can be used for troubleshooting and to verify connectivity between switches.
- **BGP autonomous system numbers**—Each switch in the IP fabric requires its own autonomous system number. Each spine and each leaf have a unique BGP autonomous system number. This allows EBGP to be used between the leaves and spines.
- **BGP export policy**—Each of the leaves needs to advertise its local server-facing IP prefix into the IP fabric so that all other servers can reach it. Each leaf would also need to export its loopback address into the IP fabric as well.
- **BGP import policy**—Because each leaf only focuses on server-facing IP prefixes and loopback addressing, all other addressing of point-to-point links can be filtered out.
- **Equal cost multipath routing**—Each spine and leaf should have the ability to load-balance flows across a set of equal next hops. For example, if there are four spine switches, each leaf has a connection to each spine. For every flow exiting a leaf switch, there should exist four equal next hops—one for each spine. In order to do this, equal cost multipath (ECMP) routing should be enabled.

These requirements can easily build a blueprint for the IP fabric. Although the network might not be fully built out from day one, it's good to have a scale-out blueprint of the IP fabric so that there's no question about how to scale out the network in the future.

Decision Points

There are a couple of important decision points to consider when you design an IP fabric. The first is whether to use IBGP or EBGP. At first, this might seem like a simple choice, but there are some other variables that make the decision a bit more complicated. The second decision point is actually a fallout of the first—should you use 16-bit or 32-bit ASNs? Let's walk through the decision points one at a time and take a closer look.

The first decision point should take you back to your JNCIE or CCIE days. What are the requirements of IBGP versus EBGP? We all know that IBGP requires a full mesh in order to propagate prefixes throughout the topology. However, EBGP doesn't require a full mesh and is more flexible. Obviously, the reason behind this is loop prevention. In order to prevent loops, IBGP does not propagate prefixes detected from one IBGP peer to another. Each IBGP switch must have a BGP session to the other in order to fully propagate routes. On the other hand, EBGP simply propagates all BGP prefixes to all BGP neighbors. The exception is that any prefixes that contain the switch's own autonomous system number are dropped.

IBGP Design

Let's take a look at an IBGP design that meets all of the implementation requirements of building an IP fabric. The first challenge is how do you get around the full mesh requirement of IBGP? The answer is BGP confederations or route reflection. Given that an IP fabric is a fixed topology, route reflection lends itself nicely. Each spine switch can act as a BGP route reflector while each leaf is a BGP route reflector client.

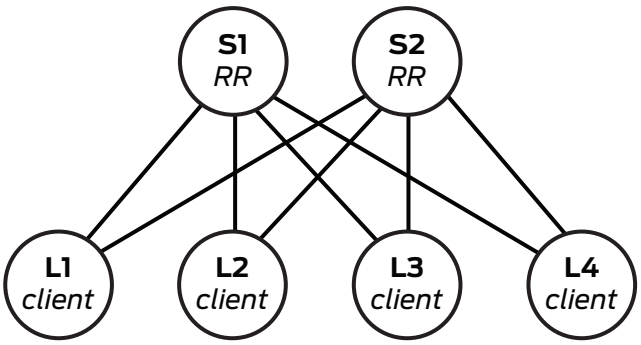

Figure 8: IBGP design with route reflectors

Each spine, as shown in Figure 8, serves as a BGP route reflector. Each leaf, L1 through L4, is a route reflector client.

> *It's important to ensure that the spine switches support BGP route reflection if you want to use an IBGP design. Fortunately, the QFX5100 line of switches supports BGP route reflection.*

The other critical implementation requirement that must be met in an IBGP design is ECMP routing. By default, BGP route reflectors only reflect the best route. This means that if four ECMP routes exist, only the single, best prefix is reflected to the clients. Obviously, this breaks the ECMP requirement and something must be done.

The answer to this problem is to enable the BGP route reflector to send multiple paths instead of the best. There is currently a draft in the IETF that enables this behavior and is called `draft-ietf-idr-add-paths`. The feature is called BGP AddPath. Now the route reflector can offer all ECMP routes to each client.

> *Make sure that the spine switches in your IP fabric support BGP Add Path if you want to design the network using IBGP. The QFX5100 line of switches supports BGP Add Path as well as BGP route reflection.*

In summary, the spine switches must support BGP route reflection as well as BGP Add Path to meet all of the IP fabric requirements. IBGP requires a couple of more requirements, but it allows you to manage the entire IP fabric as a single autonomous system number.

EBGP Design

The other alternative is to use EBGP to design the IP fabric. By default, EBGP meets all of the implementation requirements when you build the IP fabric. There's no need for BGP route reflection or BGP Add Path with EBGP, as shown in Figure 9.

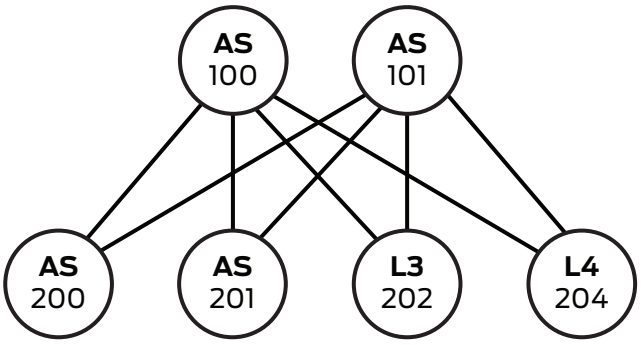

Figure 9: EBGP requires a BGP autonomous system number per switch

The only issue you really have to worry about is how many BGP autonomous system numbers you consume with the IP fabric. Each switch has its own BGP autonomous system number. Technically, the BGP private range is 64,512 to 65,535, which leaves you with 1023 BGP autonomous system numbers. If the IP fabric is larger than 1023 switches, you need to consider moving into the public BGP autonomous system number range or move to 32-bit autonomous system numbers.

> *As you can see, EBGP has the simplest design that meets all of the implementation requirements. This works well when you create a multivendor IP fabric.*

Edge Connectivity

The other critical decision point is how do you connect your data center to the rest of the world and to other data centers? There are multiple decision points due to the number of endpoints and options. Let's review a simple example of two data centers as shown in Figure 10.

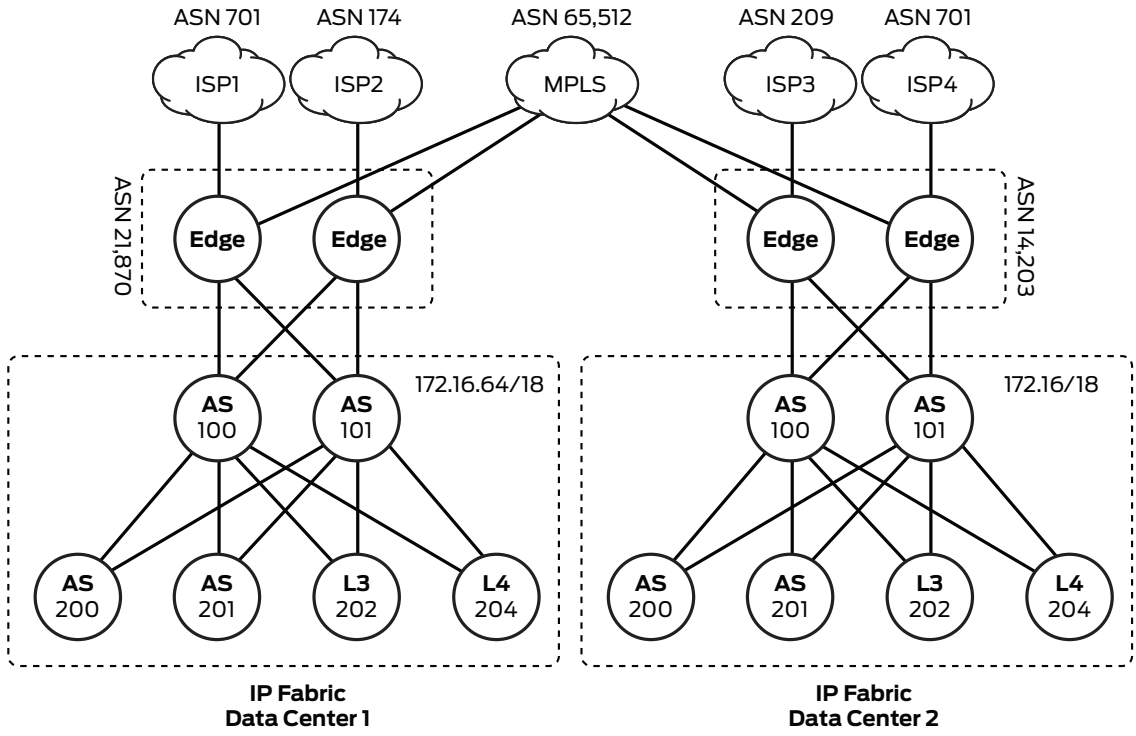

Figure 10: Edge connectivity in two data centers with IP fabrics

Each data center must:

- Have an IP fabric using the same BGP autonomous system number and scheme
- Include two edge routers with a unique BGP autonomous system number
- Be connected to two ISPs
- Be connected to a private MPLS network

Reusing the same EBGP design in each data center reduces the operational burden of bringing up new data centers. It also creates a consistent operational experience regardless of which data center you're in. The drawback is that using the same autonomous system numbers throughout the entire design makes things confusing in the MPLS core. For example, what BGP prefixes does AS 200 own? The answer is that it depends on which data center you're in.

One simple solution is to use the BGP autonomous system override feature. This allows the PE routers in the MPLS network to change the autonomous system used by the edge routers in each data center. Now we can simply say that ASN 21,870 owns the aggregate 172.16.64/18, and autonomous system number 14,203 owns the aggregate 172.16/18. For example, from the perspective of Data Center 1, the route to 172.16/18 is through BGP autonomous system number 65,512 then 14,203. In order to do this, you must create a BGP export policy on the edge routers in each data center that rejects all of the IP fabric prefixes but instead advertises a single BGP aggregate.

When we talk about connecting out to the Internet, the design is a little different. The goal is that the IP fabric should have a default route of 0/0, but the edge routers should have a full Internet table. Each data center has its own public IP range that needs to be advertised out to each ISP as well. In summary, the edge routers perform the following actions:

- Advertise a default route into the IP fabric
- Advertise public IP ranges to each ISP
- Reject all other prefixes

BGP Design Summary

IP fabrics allow you to create some very large networks that can easily support overlay networking architectures in the data center. There are a few decision points that you must consider carefully when creating your own IP fabric. How many switches are you deploying? Do you want to design for a multivendor environment using BGP features? How many data centers are connected to each other? These are the questions you must ask yourself and consider into the overall design of each data center.

BGP Implementation

Let's get down to the brass tacks. Moving from the design phase to the implementation phase requires physical devices, configurations, and verification. This section walks through the implementation in detail using Junos OS. In this laboratory we have two spines and three leaves, as shown in Figure 11.

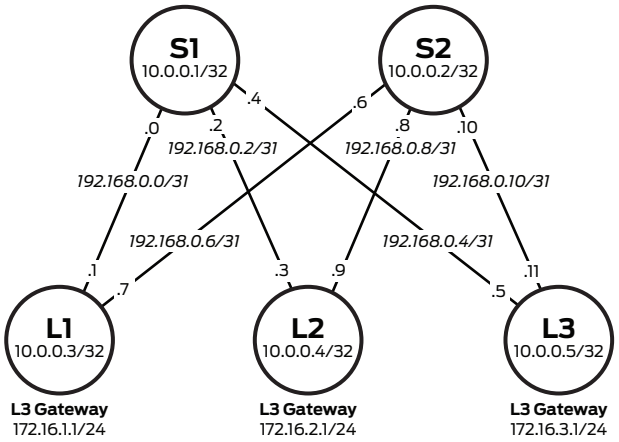

Figure 11: BGP implementation of an IP fabric

- Loopback address—Each switch uses a 32-bit loopback address from the 10/8 range.
- Point-to-point addresses—Each switch uses a 31-bit network on each point-to-point link starting from 192.168/24.
- Layer 3 server gateway—Servers connecting to the IP fabric require a default gateway. Each leaf has gateway services starting at 172.16.1/24.

Topology Configuration

The first step is to walk through the topology and understand how each switch is connected, what the BGP attributes are, and the IP address schemes. Each switch has a hostname, loopback, L3 gateway, and a BGP autonomous system number. Let's walk through them in Table 2.

Table 2: BGP Implementation Details

Switch	Loopback	L3 Gateway	BGP Autonomous System Number
S1	10.0.0.1/32	None	100
S2	10.0.0.2/32	None	101
L1	10.0.0.3/32	172.16.1.1/24	200
L2	10.0.0.4/32	172.16.2.1/24	201
L3	10.0.0.5/32	172.16.3.1/24	202

Interface and IP Configuration

Now let's take a look at the physical connection details of each switch. We review the interface names, point-to-point network, and IP addresses, as shown in Table 3.

Table 3: Interface and IP Implementation Details

Source Switch	Source Interface	Source IP	Network	Destination Switch	Destination Interface	Destination IP
L1	xe-0/0/14	.1	192.168.0.0/31	S1	xe-0/0/14	.0
L1	xe-0/0/15	.7	192.168.0.6/31	S2	xe-0/0/15	.6
L2	xe-0/0/16	.3	192.168.0.2/31	S1	xe-0/0/16	.2
L2	xe-0/0/17	.8	192.168.0.8/31	S2	xe-0/0/17	.8
L3	xe-0/0/18	.11	192.168.0.10/31	S1	xe-0/0/18	.10
L3	xe-0/0/19	.1	192.168.0.0/31	S2	xe-0/0/19	.0

Each leaf is connected to each spine, but notice that the spines aren't connected to each other. In an IP fabric, there's no requirement for the spines to be directly connected. Given any single link failure scenario, all leaves still have connectivity to each other. The other detail is that an IP fabric is all Layer 3. Traditional Layer 2 networks require connections between the spines to ensure proper flooding and propagation of the broadcast domains. Yet another reason to favor Layer 3 IP fabric—no need to interconnect the spines.

BGP Configuration

One of the first steps is to configure each spine to peer via EBGP to each leaf. One trick to speed up the BGP processing in Junos OS is to keep all the neighbors in a single BGP group. We can certainly do this because the import and export policies are identical, but only the peer autonomous system and neighbor IP vary from leaf to leaf. Let's take a look at the BGP configuration of S1:

```
protocols {
    bgp {
        log-updown;
        import bgp-clos-in;
        export bgp-clos-out;
        graceful-restart;
        group CLOS {
            type external;
            mtu-discovery;
            bfd-liveness-detection {
                minimum-interval 350;
                multiplier 3;
                session-mode single-hop;
            }
            multipath multiple-as;
            neighbor 192.168.0.1 {
                peer-as 200;
            }
            neighbor 192.168.0.3 {
                peer-as 201;
            }
            neighbor 192.168.0.5 {
                peer-as 202;
            }
        }
    }
}
```

Each leaf has its own neighbor statement with the proper IP address. In addition, each neighbor has its own specific peer autonomous system. This allows all of the leaves in the IP fabric to be placed under a single BGP group called `CLOS`.

There are a few global BGP options we want to enable, so that we don't have to specify them for each group and neighbor.

- `log-updown`—This enables tracking the state of all BGP sessions. All groups and neighbors inherit this option. Now we can keep track of the entire IP fabric from the point of view of each switch.
- Import and export policies—A common import and export policy is used across the entire IP fabric. It doesn't make a difference if it's a leaf or a spine. We review the policy statements in more detail later in this white paper.
- `graceful-restart`—Of course we want the ability to make policy changes to BGP without having to tear down existing sessions. To enable this functionality, we can enable the `graceful-restart` feature in Junos OS.

Under the `CLOS` BGP group, we also enable some high-level features. Let's walk through them one by one:

- `type external`—This enables EBGP for the entire BGP group. Given that the IP fabric is based on an EBGP design, there's no need to repeat this information for each neighbor.
- `mtu-discovery`—We're running jumbo frames on the physical interfaces. Allowing BGP to discover the larger MTU helps in processing control plane updates.
- BFD—To ensure that we have fast convergence, we offload the forwarding detection to BFD. In this example we're using a 350 ms interval with a 3x multiplier.
- `multipath multiple-as`—In order to allow for ECMP across a set of EBGP neighbors, we have to enable the `multipath multiple-as` option.

BGP Policy Configuration

The real trick is writing the BGP policy for importing and exporting the prefixes throughout the IP fabric. It's actually very straightforward. We can craft a common set of BGP policies to be used across both spines and leaves, which results in a simple copy and paste operation. Let's walk through them.

First up is the BGP export policy:

```
policy-options {
    policy-statement bgp-clos-out {
        term loopback {
            from {
                protocol direct;
                route-filter 10.0.0.0/24 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
        term server-L3-gw {
            from {
                protocol direct;
                route-filter 172.16.0.0/12 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
    }
}
```

There's a lot happening in this policy. Let's walk through each term:

- `term loopback`—The first order of business is to identify the switch's loopback address and export it to all other BGP peers. We can do this by looking at the directly connected interfaces that match a 10/24 or longer bitmask. This quickly identifies all loopback addresses across the entire IP fabric. Because we don't want to propagate all of the point-to-point addresses throughout the IP fabric, we use next-hop self to change the next hop of each advertised prefix to the egress interface of the switch. Now each switch in the IP fabric directly uses neighbor IP addresses in the forwarding table.
- `term server-L3-gw`—We already know that each leaf has Layer 3 gateway services for the servers connected to it. The range is 172.16/12. This matches all of the server gateway addresses on each leaf. Of course, we apply the next-hop self as well. Obviously, this has no effect on the spines and only works on the leaves. It's great being able to write a single policy for both switches.
- Default—Each BGP policy has a default term at the very end. It isn't configurable, but it follows the default rules of EBGP: Advertise all EBGP and IBGP prefixes to the neighbor—otherwise, deny all other prefixes. This simply means that other BGP prefixes in the routing table are advertised to other peers. Installing an explicit reject action at the end can stop this behavior, but in this case we want the IP fabric to propagate all BGP prefixes to all leaves.

Now let's take a look at the import policy configuration:

```
policy-options {
    policy-statement bgp-clos-in {
        term loopbacks {
            from {
                route-filter 10.0.0.0/24 orlonger;
            }
            then accept;
        }
        term server-L3-gw {
            from {
                route-filter 172.16.0.0/12 orlonger;
            }
            then accept;
        }
        term reject {
            then reject;
        }
    }
}
```

Again, there is a lot happening in the import policy. At a high level we want to be very selective about what types of prefixes we accept into the routing and forwarding table of each switch. Let's walk through each term in detail:

- `term loopbacks`—Obviously, we want each switch to have reachability to every other switch in the IP fabric via loopback addresses. We explicitly match on the 10/8 and allow all loopback addresses into the routing and forwarding table.
- `term server-L3-gw`—The same goes for server Layer 3 gateway addresses. Each leaf in the IP fabric needs to detect all other gateway addresses. We explicitly match on 172.16/12 to allow this.
- `term reject`—At this point we've had enough. We reject all other prefixes. The problem is that if we don't have a reject statement at the end of the import policy, the routing and forwarding tables are trashed by all of the point-to-point networks. There's no reason to have this information in each switch, because it's only relevant to the immediate neighbor of its respective switch.

We simply export and import loopbacks and Layer 3 server gateways, and we propagate all prefixes throughout the entire IP fabric. The best part is that we can reuse the same set of policies throughout the entire IP fabric as well—simply copy and paste.

ECMP Configuration

Recall that earlier we used the `multipath multiple-as` configuration knob? That alone only installs ECMP prefixes into the routing information base (RIB), which is also known as a routing table. In order to take full ECMP from the RIB and install it into the forwarding information base (FIB)—also known as a forwarding table—we need to create another policy that enables ECMP and install it into the FIB. Let's walk through it.

```
routing-options {
    forwarding-table {
        export PFE-LB;
    }
}
policy-options {
    policy-statement PFE-LB {
        then {
            load-balance per-packet;
        }
    }
}
```

What's happening here is that the PFE-LB policy simply states that for any packet being forwarded by the switch, enable load balancing—this enables full ECMP in the FIB. However, the existence of the PFE-LB policy by itself is useless. It must be applied into the FIB directly. This is done under routing-options forwarding-table and references the PFE-LB policy.

BGP Verification

Now that the IP fabric has been configured, the next step is to ensure that the control plane and data plane are functional. We can verify the IP fabric through the use of show commands to check the state of the BGP sessions, what prefixes are being exchanged, and the passing of packets through the network.

BGP State

Let's kick things off by logging into `S1` and checking the BGP sessions:

```
dhanks@S1> show bgp summary
Groups: 1 Peers: 3 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0
                     6          6          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
State|#Active/Received/Accepted/Damped...
192.168.0.1            200      12380      12334       0        3 3d 21:11:35 2/2/2/0
0/0/0/0
192.168.0.3            201      12383      12333       0        2 3d 21:11:35 2/2/2/0
0/0/0/0
192.168.0.5            202      12379      12333       0        2 3d 21:11:35 2/2/2/0
0/0/0/0
```

All is well, and each BGP session to each leaf is connected and exchanging prefixes. We can see that each session has two active, received, and accepted prefixes—these are the loopback and Layer 3 gateway addresses. So far everything is great.

Let's dig further down the rabbit hole. We need to verify—from a control plane perspective—ECMP, graceful restart, and BFD. Here it is:

```
dhanks@S1> show bgp neighbor 192.168.0.1
Peer: 192.168.0.1+60120 AS 200 Local: 192.168.0.0+179 AS 100
  Type: External    State: Established    Flags: <Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: Cease
  Export: [ bgp-clos-out ] Import: [ bgp-clos-in ]
  Options: <Preference LogUpDown PeerAS Multipath Refresh>
  Options: <MtuDiscovery MultipathAs BfdEnabled>
  Holdtime: 90 Preference: 170
  Number of flaps: 3
  Last flap event: Stop
  Error: 'Cease' Sent: 1 Recv: 1
  Peer ID: 10.0.0.3        Local ID: 10.0.0.1          Active Holdtime: 90
  Keepalive Interval: 30         Group index: 1    Peer index: 0
  BFD: enabled, up
  Local Interface: xe-0/0/14.0
  NLRI for restart configured on peer: inet-unicast
  NLRI advertised by peer: inet-unicast
  NLRI for this session: inet-unicast
  Peer supports Refresh capability (2)
  Stale routes from peer are kept for: 300
  Peer does not support Restarter functionality
  NLRI that restart is negotiated for: inet-unicast
  NLRI of received end-of-rib markers: inet-unicast
  NLRI of all end-of-rib markers sent: inet-unicast
  Peer supports 4 byte AS extension (peer-as 200)
  Peer does not support Addpath
  Table inet.0 Bit: 10000
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:              2
    Received prefixes:            2
    Accepted prefixes:            2
    Suppressed due to damping:    0
    Advertised prefixes:          3
  Last traffic (seconds): Received 1    Sent 25   Checked 42
  Input messages:  Total 12381Updates 3Refreshes 0Octets 235340
  Output messages: Total 12334Updates 7Refreshes 0Octets 234634
  Output Queue[0]: 0
```

The important bits are highlighted. Take a closer look at the two lines of Options. We can see:

- The state of the BGP session is logged
- ECMP is supported
- Graceful restart is supported
- MTU discovery is enabled
- BFD is bound to BGP

BGP Prefixes

Now that BGP itself is configured correctly, let's examine what BGP is doing. Let's take a closer look at S1 and see what prefixes are being advertised to L1.

```
dhanks@S1> show route advertising-protocol bgp 192.168.0.1 extensive

inet.0: 53 destinations, 53 routes (52 active, 0 holddown, 1 hidden)
* 10.0.0.1/32 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self
     Flags: Nexthop Change
     AS path: [100] I

* 10.0.0.4/32 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self (rib-out 192.168.0.3)
     AS path: [100] 201 I

* 10.0.0.5/32 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self (rib-out 192.168.0.5)
     AS path: [100] 202 I

* 172.16.2.0/24 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self (rib-out 192.168.0.3)
     AS path: [100] 201 I

* 172.16.3.0/24 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self (rib-out 192.168.0.5)
     AS path: [100] 202 I
```

Things are really looking great. `S1` is advertising five prefixes to `L1`. Let's break them down:

- 10.0.0.1/32—This is the loopback address on S1 itself. We're advertising this prefix to L1.
- 10.0.0.4/32—This is the loopback address for L2. We're simply passing this prefix on to L1. You can see that the autonomous system path is [100] 201 I, which means that the route origin was internal and you can simply follow the autonomous system itself back to L2.
- 10.0.0.5/32—The same goes for the loopback address of L3. It is passed on to L1.
- 172.16.2.0/24—This is the Layer 3 gateway address for L2. It is passed on to L1.
- 172.16.3.0/24—The same goes for the Layer 3 gateway address for L3. It is passed on to L1.

Now let's take a look at what we're receiving from the other leaves.

```
dhanks@S1> show route receive-protocol bgp 192.168.0.1
```

```
inet.0: 53 destinations, 53 routes (52 active, 0 holddown, 1 hidden)
  Prefix  Nexthop      MED     Lclpref    AS path
* 10.0.0.3/32            192.168.0.1                             200 I
* 172.16.1.0/24          192.168.0.1                             200 I

dhanks@S1> show route receive-protocol bgp 192.168.0.3

inet.0: 53 destinations, 53 routes (52 active, 0 holddown, 1 hidden)
  Prefix  Nexthop      MED     Lclpref    AS path
* 10.0.0.4/32            192.168.0.3                             201 I
* 172.16.2.0/24          192.168.0.3                             201 I

dhanks@S1> show route receive-protocol bgp 192.168.0.5

inet.0: 53 destinations, 53 routes (52 active, 0 holddown, 1 hidden)
  Prefix  Nexthop      MED     Lclpref    AS path
* 10.0.0.5/32            192.168.0.5                             202 I
* 172.16.3.0/24          192.168.0.5                             202 I
```

Once again, we can confirm that each leaf is only advertising its loopback and Layer 3 gateway address into the spine.

Routing Table

Now that we have verified the prefixes are being exchanged correctly between the switches, let's make sure that the RIB is being populated correctly. The easiest way to verify this is to log in to `L1` and verify that we see ECMP to the loopback address of `L3`:

```
dhanks@L1> show route 172.16.3.1/24 exact

inet.0: 54 destinations, 58 routes (53 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both
172.16.3.0/24      *[BGP/170] 3d 10:55:14, localpref 100, from 192.168.0.6
                      AS path: 101 202 I
                    > to 192.168.0.0 via xe-0/0/14.0
                      to 192.168.0.6 via xe-0/0/15.0
                    [BGP/170] 3d 10:55:14, localpref 100
                      AS path: 100 202 I
                    > to 192.168.0.0 via xe-0/0/14.0
```

What we see here is that there are two next hops to `L3` from `L1`. This is a result of a proper BGP configuration using the `multipath multiple-as` knob.

Forwarding Table

The next step is to make sure that the RIB is correctly programming the forwarding table. We verify the same way. Start on `L1` and verify to `L3`:

```
dhanks@L1> show route forwarding-table destination 172.16.3.1
Routing table: default.inet
Internet:
Destination        Type RtRef Next hop           Type Index NhRef Netif
172.16.3.0/24      user     0                    ulst 131070     5
                              192.168.0.0        ucst  1702     5 xe-0/0/14.0
                              192.168.0.6        ucst  1691     5 xe-0/0/15.0
```

Of course, what we see here are two next hops—one toward **S1 (xe-0/0/14)** and the other toward **S2 (xe-0/0/15)**.

Ping

A simple way to verify the data plane connectivity is to log in to **L1** and source a ping from its Layer 2 gateway address and ping **L3**. This forces traffic through the spine of the network.

```
dhanks@L1> ping source 172.16.1.1 172.16.3.1 count 5
PING 172.16.3.1 (172.16.3.1): 56 data bytes
64 bytes from 172.16.3.1: icmp_seq=0 ttl=63 time=3.009 ms
64 bytes from 172.16.3.1: icmp_seq=1 ttl=63 time=2.163 ms
64 bytes from 172.16.3.1: icmp_seq=2 ttl=63 time=2.243 ms
64 bytes from 172.16.3.1: icmp_seq=3 ttl=63 time=2.302 ms
64 bytes from 172.16.3.1: icmp_seq=4 ttl=63 time=1.723 ms

--- 172.16.3.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.723/2.288/3.009/0.414 ms
```

So far, so good. The trick here is to source the ping from the Layer 3 gateway address. That way, we know that **L3** has a return route for **L1**.

Traceroute

To get the next level of detail, let's use traceroute. We can verify that traffic is moving through the spine of the IP fabric. Let's mix it up a bit and use the loopback addresses instead.

```
dhanks@L1> traceroute source 10.0.0.3 10.0.0.5
traceroute to 10.0.0.5 (10.0.0.5) from 10.0.0.3, 30 hops max, 40 byte packets
 1  192.168.0.6http://bb06-cclab-lo0.spglab.juniper.net/ (192.168.0.6)  2.031 ms
1.932 ms 192.168.0.0 (192.168.0.0)  2.121 ms
 2  10.0.0.5 (10.0.0.5)  2.339 ms  2.342 ms  2.196 ms
```

What's interesting here is that we can see that the traceroute went through **S2** to get to **L3**. This is just a result of how the traceroute traffic was hashed by the forwarding table of **L1**.

Configurations

If you want to build your own IP fabric and use this laboratory as a foundation, use the configuration. For the sake of page count, the author shares a single spine and leaf switch. The astute reader can figure out the rest.

S1

```
interfaces {
    xe-0/0/14 {
        mtu 9216;
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.0/31;
            }
        }
    }
    xe-0/0/16 {
        mtu 9216;
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.2/31;
```

```
                }
            }
        }
        xe-0/0/18 {
            mtu 9216;
            unit 0 {
                family inet {
                    mtu 9000;
                    address 192.168.0.4/31;
                }
            }
        }
        lo0 {
            unit 0 {
                family inet {
                    address 10.0.0.1/32;
                }
            }
        }
    }
    routing-options {
        router-id 10.0.0.1;
        autonomous-system 100;
        forwarding-table {
            export PFE-LB;
        }
    }
    protocols {
        bgp {
            log-updown;
            import bgp-clos-in;
            export bgp-clos-out;
            graceful-restart;
            group CLOS {
                type external;
                mtu-discovery;
                bfd-liveness-detection {
                    minimum-interval 350;
                    multiplier 3;
                    session-mode single-hop;
                }
                multipath multiple-as;
                neighbor 192.168.0.1 {
                    peer-as 200;
                }
                neighbor 192.168.0.3 {
                    peer-as 201;
                }
                neighbor 192.168.0.5 {
```



```
                peer-as 202;
            }
        }
    }
    policy-options {
        policy-statement PFE-LB {
            then {
                load-balance per-packet;
            }
        }
        policy-statement bgp-clos-in {
            term loopbacks {
                from {
                    route-filter 10.0.0.0/24 orlonger;
                }
                then accept;
            }
            term server-L3-gw {
                from {
                    route-filter 172.16.0.0/12 orlonger;
                }
                then accept;
            }
            term reject {
                then reject;
            }
        }
        policy-statement bgp-clos-out {
            term loopback {
                from {
                    protocol direct;
                    route-filter 10.0.0.0/24 orlonger;
                }
                then {
                    next-hop self;
                    accept;
                }
            }
            term server-L3-gw {
                from {
                    protocol direct;
                    route-filter 172.16.0.0/12 orlonger;
                }
                then {
                    next-hop self;
                    accept;
                }
            }
        }
```

```
}
L1
interfaces {
    interface-range ALL-SERVER {
        member-range xe-0/0/0 to xe-0/0/13;
        member-range xe-0/0/16 to xe-0/0/47;
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members SERVER;
                }
            }
        }
    }
    xe-0/0/14 {
        mtu 9216;
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.1/31;
            }
        }
    }
    xe-0/0/15 {
        mtu 9216;
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.7/31;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.0.3/32;
            }
        }
    }
    irb
        mtu 9216;
        unit 1 {
            family inet {
                mtu 9000;
                address 172.16.1.1/24;
            }
        }
    }
```

```
}
routing-options {
    router-id 10.0.0.3;
    autonomous-system 200;
    forwarding-table {
        export PFE-LB;
    }
}
protocols {
    bgp {
        log-updown;
        import bgp-clos-in;
        export bgp-clos-out;
        graceful-restart;
        group CLOS {
            type external;
            mtu-discovery;
            bfd-liveness-detection {
                minimum-interval 350;
                multiplier 3;
                session-mode single-hop;
            }
            multipath multiple-as;
            neighbor 192.168.0.0 {
                peer-as 100;
            }
            neighbor 192.168.0.6 {
                peer-as 101;
            }
        }
    }
}
policy-options {
    policy-statement PFE-LB {
        then {
            load-balance per-packet;
        }
    }
    policy-statement bgp-clos-in {
        term loopbacks {
            from {
                route-filter 10.0.0.0/24 orlonger;
            }
            then accept;
        }
        term server-L3-gw {
            from {
                route-filter 172.16.0.0/12 orlonger;
            }
```

```
                then accept;
            }
            term reject {
                then reject;
            }
        }
        policy-statement bgp-clos-out {
            term loopback {
                from {
                    protocol direct;
                    route-filter 10.0.0.0/24 orlonger;
                }
                then {
                    next-hop self;
                    accept;
                }
            }
            term server-L3-gw {
                from {
                    protocol direct;
                    route-filter 172.16.0.0/12 orlonger;
                }
                then {
                    next-hop self;
                    accept;
                }
            }
            term reject {
                then reject;
            }
        }
    }
    vlans {
        SERVER {
            vlan-id 1;
            l3-interface irb.1;
        }
    }
```

Conclusion

This white paper covered the basic ways to build an IP fabric. More important, it reviewed the decision points you must take into account when building an IP fabric. There are various options in the control plane that impact what features are required on the platform. Finally, we reviewed in great detail how to implement BGP in an IP fabric. We walked through all of the interfaces, IP addresses, BGP configurations, and policies. To wrap things up, we verified that BGP is working across the IP fabric and ran some tests on the data plane to make sure traffic can get from leaf to leaf.

Building an IP fabric is very straightforward and serves as a great foundation to overlay technologies such as VMware NSX and Juniper Networks Contrail. Basing the design on only Layer 3 makes the IP fabric very resilient to failure and offers very fast end-to-end convergence with BFD. Build your next IP fabric with the QFX5100 Switch.

For additional details, look for the forthcoming Juniper Networks Technical Library book, “The QFX Series,” to be published in Q4 2014 by O’Reilly Media. Portions of this white paper will be included and elaborated upon in that book. See all the books in the Technical Library at **www.juniper.net/books**.

About Juniper Networks

Juniper Networks is in the business of network innovation. From devices to data centers, from consumers to cloud providers, Juniper Networks delivers the software, silicon and systems that transform the experience and economics of networking. The company serves customers and partners worldwide. Additional information can be found at www.juniper.net.

Corporate and Sales Headquarters
Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089 USA
Phone: 888.JUNIPER (888.586.4737)
or +1.408.745.2000
Fax: +1.408.745.2100
www.juniper.net

APAC and EMEA Headquarters
Juniper Networks International B.V.
Boeing Avenue 240
1119 PZ Schiphol-Rijk
Amsterdam, The Netherlands
Phone: +31.0.207.125.700
Fax: +31.0.207.125.701

2000565-003-EN Sept 2015