Border Gateway Protocol (BGP) Traffic Engineering Server for Leaf/Spine Data Center Fabrics

Technical Whitepaper

Version 1.7

Authored by:

Nicholas Russo

CCDE #20160041

CCIE #42518 (EI/SP)

Review & Approval

Version and Date	Change	Responsible Person
20160619 Version 1.0	Initial draft release	Nicholas Russo
20160621 Version 1.1	Corrections and clarification	Nicholas Russo
20170219 Version 1.2	Sanitized for public distribution	Nicholas Russo
20201002 Version 1.3	Corrections and clarification	Nicholas Russo
20201205 Version 1.4	Legal disclaimers and cleanup	Nicholas Russo
20210924 Version 1.5	Full design review and automation tools	Nicholas Russo
20211001 Version 1.6	Added gRPC API automation section	Nicholas Russo
20211012 Version 1.7	Added REST API automation section	Nicholas Russo

Contents

Figures

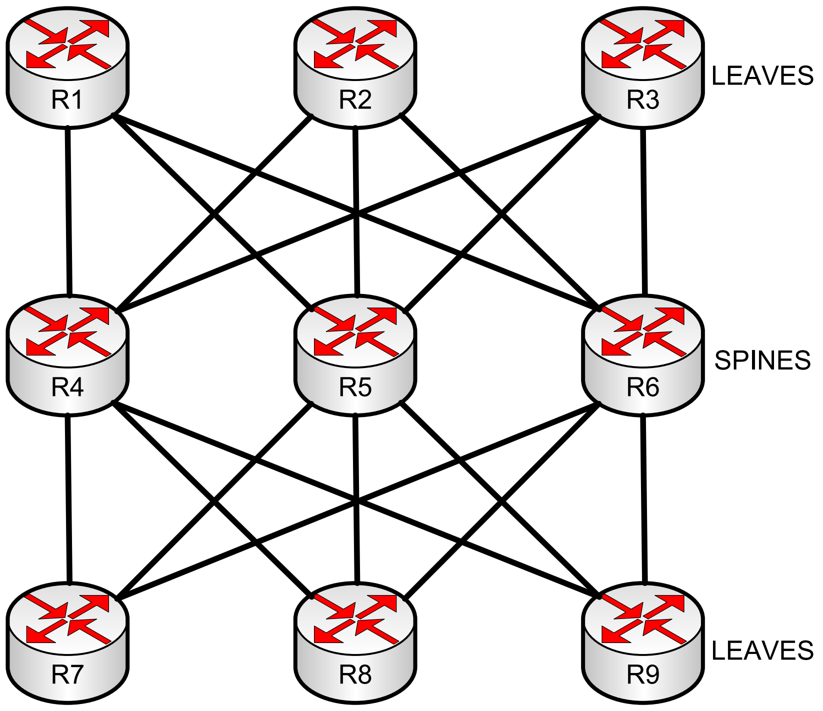
Tables

1. Leaf/Spine Architecture Overview

The popular leaf/spine architecture is based on a circuit-switched network design invented by Charles Clos and was published in 1952. All links in the fabric are considered "non-blocking", which in a packet-switched network, is akin to being "usable for concurrent forwarding". This design is commonly used in many data center (DC) fabrics today since it maximizes the amount of leaf-to-leaf bandwidth available in the network. Servers and other devices are only connected to leaves, never to spines. Spines only interconnect leaves and offer no services of their own. Devices in the same horizontal tier are never connected laterally, either. Adding bandwidth to a leaf/spine architecture is as simple as adding new spines, known as "scaling out". This action also improves availability as the network can tolerate more concurrently failed spines. Adding network capacity to host more services requires scaling out the leaves. Both of these two scaling actions are logically simple to accomplish and require only the assignment of new ports/subnets.

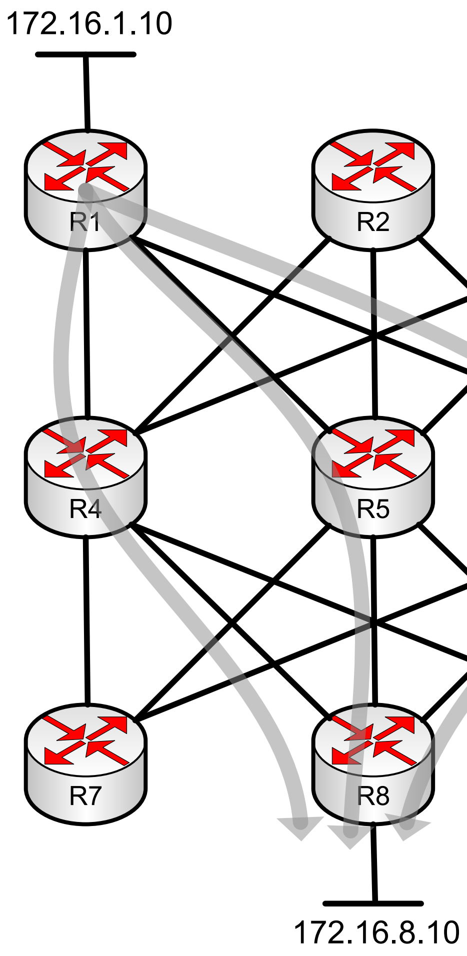
The network topology for this demonstration is a leaf/spine architecture consisting of a single tier of three spines and a single tier of six leaves. The leaves are drawn in two separate rows for clarity. Architecturally, all of the leaves are equal in terms of hierarchy and purpose.

Figure 1 - General Architecture



This document will detail how traffic engineering (TE) can be achieved for IP prefixes across the fabric using Border Gateway Protocol (BGP). BGP runs only on the leaves and is only used for TE; any interior gateway protocol (IGP) can be used in the fabric for normal forwarding which uses equal-cost multipath (ECMP) by default. It is recommended that an IGP be selected that supports a robust loop free alternate (LFA) feature set, such as Open Shortest Path First (OSPF) or Intermediate System to Intermediate System (IS-IS). This document uses OSPF as an example. The diagram below illustrates the paths taken by R1 to reach a LAN segment behind R8. Because there are three spines, R1 can use all of these transit nodes to reach the hosts behind R8.

Figure 2 - Default Forwarding with IGP-based ECMP



The lab environment uses the 10.0.0.0/8 network for transit links and device loopbacks following an intuitive format. Transit links use a /24 mask for simplicity and the middle two octets represent the routers at either end of the point-to-point link. For example, 10.3.6.0/24 represents the link between R3 and R6. The fourth octet is set to the router number for transit links and for loopbacks. For example, 10.0.0.7/32 is R7's loopback and 10.4.7.4/24 is R4's host address on the point-to-point link to R7. Last, the server LANs use the 172.16.0.0/16 network whereby the third octet is set to the router number. These details are only relevant in helping readers understand various "show" command outputs which illustrate complex technical concepts in this document.

2. BGP Traffic-Engineering

Using BGP to provide TE inside of a DC is not a new idea. Petr Lapukhov and others have detailed this in an Internet Engineering Task Force (IETF) draft focused on solving large-scale DC routing with BGP, along with TE mechanisms/solutions. This technique differs from the Lapukhov draft in that IGP is retained in the fabric and BGP is used for TE only, which is less scalable yet simpler. The Lapukhov draft is not discussed in detail but is linked in the references section.

Suppose that a unidirectional elephant flow (that is, a single flow that consumes a large quantity of bandwidth) exists in the DC flowing from 172.16.1.10 to 172.16.8.10. These represent sample host addresses behind R1 and R8, respectively. Combined with the remaining traffic along this forwarding path, the elephant flow causes congestion between R1-R4 and between R4-R8. OSPF has informed the Routing Information Base (RIB) of three ECMP paths, and all of them will be installed. The consequence is a Forwarding Information Base (FIB) with three ECMP paths that balances traffic per-source-per-destination. A single flow between a single pair of IPs cannot be spread across multiple paths unless per-packet load-sharing is enabled, which is seldom wise.

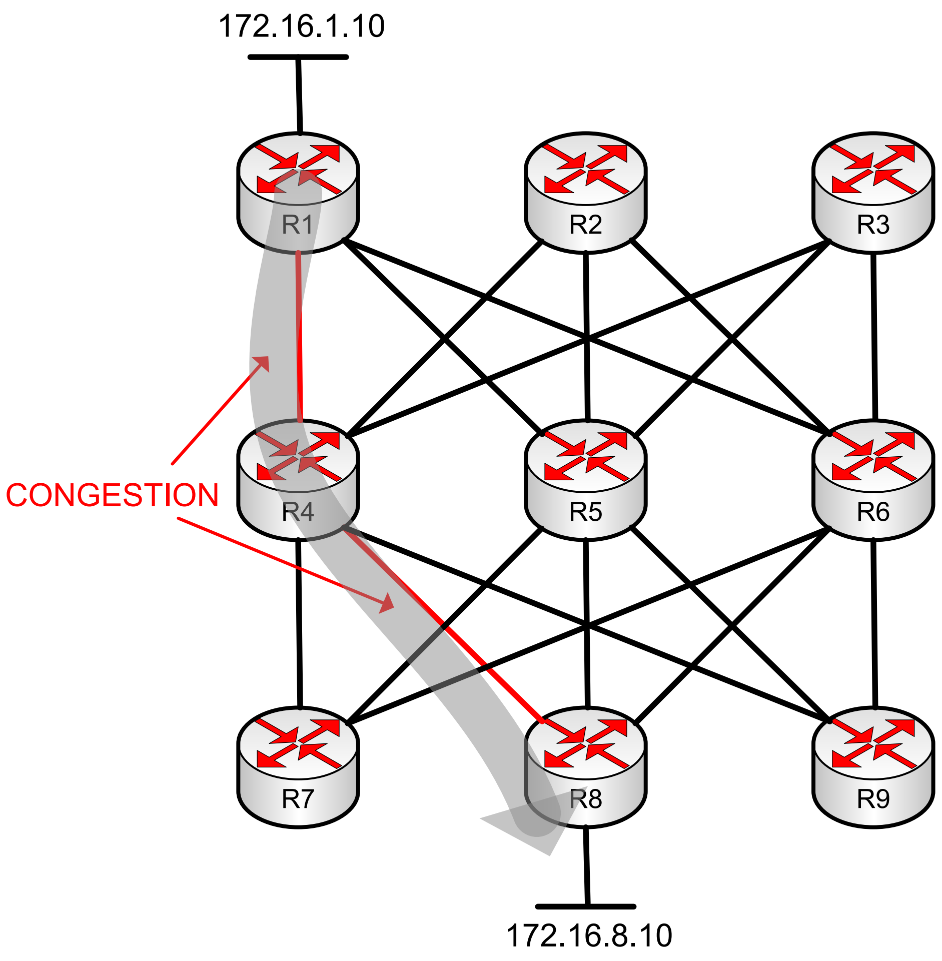
As a minor design point, consider assigning local stub networks such as server LANs and loopbacks to a non-area 0 area on each device when using OSPFv2. This ensures that any modification of these stub networks will not trigger a full SPF recalculation in the fabric. Additionally, it simplifies learning about fast-reroute on Cisco devices as it unlocks the “show ip ospf border-routers” command. This helps reveal various inter-node distances when computing backup paths, which is discussed in detail later.

The outputs below reveal the RIB, FIB, and source/destination-specific FIB entries for the flow in question. ECMP is utilized for this destination and R4 is selected as the next-hop.

```
R1#show ip route 172.16.8.10
Routing entry for 172.16.8.0/24
  Known via "ospf 1", distance 110, metric 3, type inter area
  Last update from 10.1.5.5 on GigabitEthernet2, 00:00:09 ago
  Routing Descriptor Blocks:
    10.1.6.6, from 10.0.0.8, 00:00:09 ago, via GigabitEthernet3
      Route metric is 3, traffic share count is 1
    10.1.5.5, from 10.0.0.8, 00:00:09 ago, via GigabitEthernet2
      Route metric is 3, traffic share count is 1
  * 10.1.4.4, from 10.0.0.8, 00:00:11 ago, via GigabitEthernet1
      Route metric is 3, traffic share count is 1

R1#show ip cef 172.16.8.10
172.16.8.0/24
  nexthop 10.1.4.4 GigabitEthernet1
  nexthop 10.1.5.5 GigabitEthernet2
  nexthop 10.1.6.6 GigabitEthernet3

R1#show ip cef exact-route 172.16.1.10 172.16.8.10
172.16.1.10 -> 172.16.8.10 =>IP adj out of GigabitEth1, addr 10.1.4.4
```

For those less interested in the technical specifics of Cisco devices, the diagram below illustrates the high-level problem. Congestion is likely to occur along the R1-R4-R8 path due to this elephant flow between servers in the data center.

Figure 3 - Elephant Flow Across the Fabric



To solve this problem, a set of BGP Traffic-Engineering Servers (BTS) can be used to inject routes to change the routing path through the fabric. Each leaf will form internal BGP (iBGP) sessions to the BTS. Using iBGP over external BGP (eBGP) has two key advantages:

1. Only one autonomous system (AS) number to manage.
2. Leaves that receive injected BGP routes will not advertise them to any peers by default. With eBGP, one would need to use some flavor of BGP filtering to prevent undesired

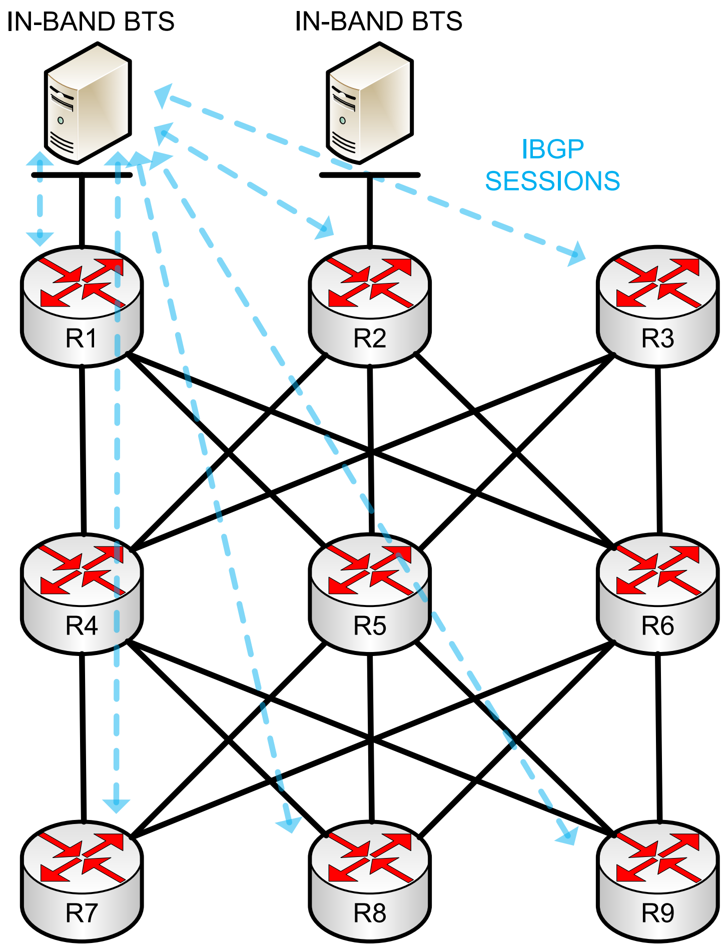
route advertisement. This could be the "no-advertise" community set outbound by the BTS, AS-path filtering applied outbound by the leaves, or any number of other options.

Additionally, the spines remain BGP-free. Achieving a "BGP-free core" does not require Multi-Protocol Label Switching (MPLS) or any other tunnelling technologies in this design. Last, consider the placement of the BTS in the network. There are two main options:

1. **In-band:** Directly connect the BTS to a set of leaves on a routable LAN segment.
2. **Out-of-band:** Connect the BTS onto a dedicated network to which all spokes have direct access. This option is more reliable as no amount of BGP route injection would break the BGP control-plane but requires additional work to build the out-of-band network (assuming it does not already exist).

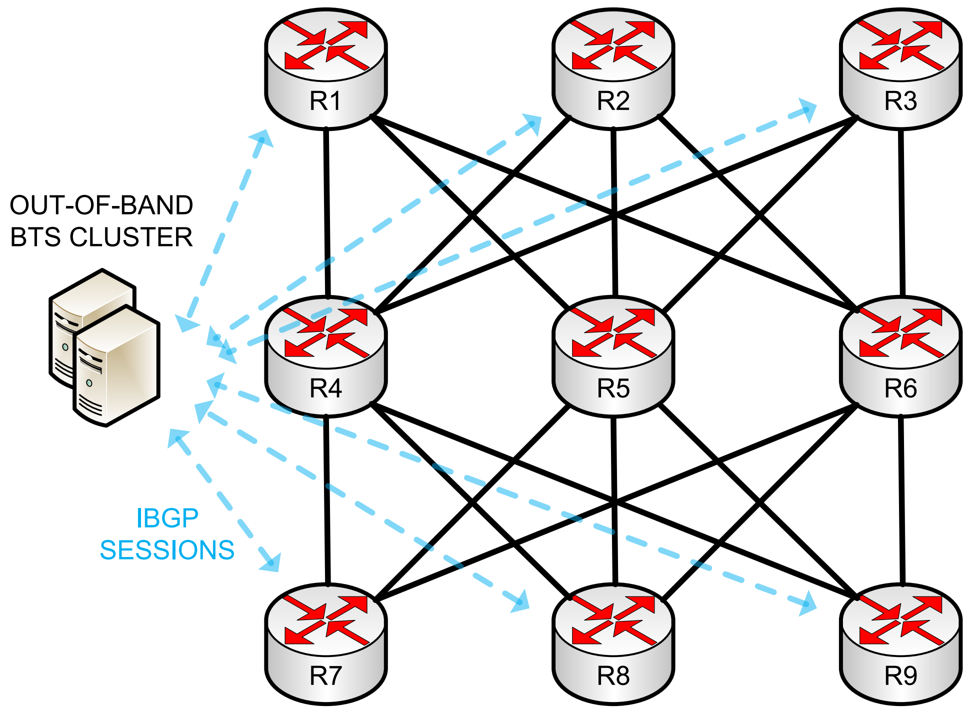
The diagram below illustrates the in-band BTS connectivity option. If high availability is desired, BTS devices should be placed behind different leaves. All leaves should peer with all BTS devices, but in the interest of cleanliness, only one set of iBGP sessions is drawn.

Figure 4 - In-band BTS iBGP Session Connectivity



The diagram below illustrates the out-of-band BTS connectivity option. Although this network is likely to lack high availability at the network level, multiple BTS devices should still be deployed and peered to all leaves.

Figure 5 - Out-of-band BTS iBGP Session Connectivity



2.1. Steering a Single TE Path

Consider the "elephant flow" scenario described earlier in this document whereby a host behind R1 is sending a large quantity of traffic to a host behind R8. Only one of the spines, namely R4, is being utilized to transport this traffic. The simplest solution is to inject a single BGP prefix from the BTS devices that either offloads the entire remote network (/24) or a subnet of it. If multiple BTS devices have been deployed, it is imperative that all of them be configured identically. This supports high availability and ensures the BGP control-plane remains synchronized. It is highly undesirable for leaves to receive conflicting routing instructions and would be difficult to troubleshoot. For brevity, the remainder of this document will describe the various technical characteristics of the solution using a single BTS device.

2.1.1. Basic Operation

The BTS advertises a longer-match route on R1 to shift traffic towards 172.16.8.0/28 across R5. Any prefix-length greater (more specific) than /24 can be used; /28 is just an example. Perhaps there are a group of servers in this range all contributing towards the congestion. Supporting variable-length subnets is more powerful and scalable than host-based TE alone. To achieve TE, the BTS sets the next-hop to the transit link connected to the remote spine/leaf pair through which traffic should flow. The example below shows a next-hop of 10.5.8.5, which is covered by the prefix 10.5.8.0/24. This connects R5 and R8 as indicated by the second and third octets in this demonstration. The actual host address of .5 is irrelevant so long as the subnet containing the address is correct.

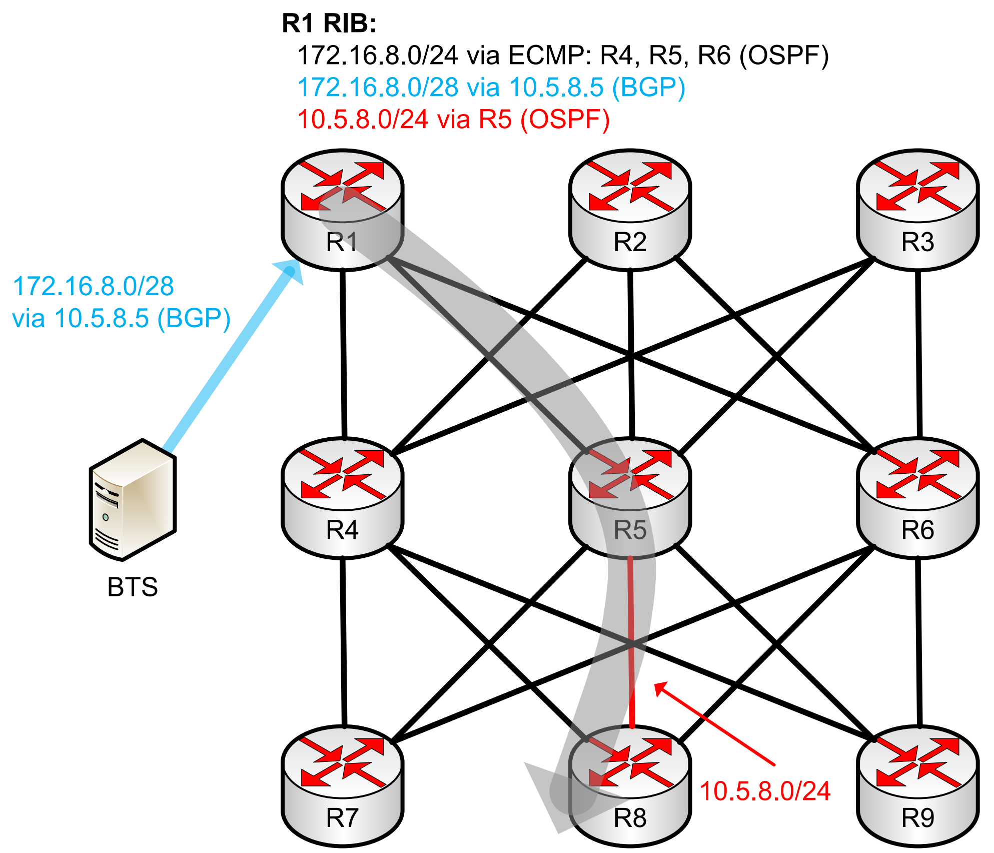
```
R1#show bgp ipv4 unicast 172.16.8.0/28
BGP routing table entry for 172.16.8.0/28, version 5
Paths: (1 available, best #1, table default)
  Flag: 0x100
  Not advertised to any peer
  Refresh Epoch 1
  Local
    10.5.8.5 (metric 2) from 192.168.0.99 (10.0.0.99)
      Origin IGP, localpref 100, valid, internal, best
      rx pathid: 0, tx pathid: 0x0
```

Also note that the administrative distance should be reduced to a value less than the fabric's IGP for iBGP routes, such as 30. This allows the BTS to totally overwrite an equal-match server LAN if necessary, effectively disabling ECMP. Generally speaking, it is smarter to inject more specific matches so that most traffic continues to be routed using the IGP-learned ECMP paths. In this way, the TE design follows a minimalist, tactical approach.

```
R1#show ip route 172.16.8.10
Routing entry for 172.16.8.0/28
  Known via "bgp 65001", distance 30, metric 0, type internal
  Last update from 10.5.8.5 00:01:40 ago
  Routing Descriptor Blocks:
  * 10.5.8.5, from 192.168.0.99, 00:01:40 ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: none
```

The diagram below depicts this process. Note that the spines remain BGP-free since they will always route to a leaf LAN via the directly connected link, so there is no need for them to have longer-matches injected by the BTS. As such, the spines are insulated from any BTS policy changes in the fabric. Note that there is no requirement to use /24 networks on transit links. The same design is valid (and has been tested) using /31 subnets for IP address conservation.

Figure 6 - Redirecting the Elephant Flow using BGP


R1 RIB:
172.16.8.0/24 via ECMP: R4, R5, R6 (OSPF)
172.16.8.0/28 via 10.5.8.5 (BGP)
10.5.8.0/24 via R5 (OSPF)
172.16.8.0/28
via 10.5.8.5 (BGP)
BTS
R1
R2
R3
R4
R5
R6
R7
R8
R9
10.5.8.0/24

R1's path to 10.5.8.0/24 will always be through R5, assuming there are no link or node failures in the fabric. One potential issue with BTS policy injection is that the automatic fast re-route (FRR) capability inherent with ECMP is diminished. A single IGP route to the BGP next-hop exists in the RIB/FIB with no load-sharing or failover mechanisms by default. If there is a failure on the R1-R5 link or if R5 fails entirely, traffic should be rapidly restored through any of the remaining spines.

This design implements a solution to this problem, and in this example, R6 was selected as a loop-free alternate (LFA) repair-path. Both R4 and R6 are equally good, but R1 will choose one as the preferred repair-path towards 10.5.8.0/24. Installing a pre-computed backup towards the BGP next-hop will minimize traffic loss during a failure of the link between R1 and R5, assuming the link detection and IGP topology change signalling happens quickly. Optimizing these timers is outside the scope of this document but is highly encouraged.

```
R1#show ip route 10.5.8.5
Routing entry for 10.5.8.0/24
  Known via "ospf 1", distance 110, metric 2, type intra area
  Last update from 10.1.5.5 on GigabitEthernet2, 00:04:19 ago
  Routing Descriptor Blocks:
```

```
  * 10.1.5.5, from 10.0.0.5, 00:04:19 ago, via GigabitEthernet2
      Route metric is 2, traffic share count is 1
      Repair Path: 10.1.6.6, via GigabitEthernet3
```

2.1.2. Loop Free Alternate (LFA) Terminology

LFAs are worth a brief discussion since the logic of LFA can be complex. The non-planar yet symmetrical characteristics of the leaf/spine fabric allow LFA to support the BTS architecture. Link-state protocols have improved the LFA process over time which relaxes the traditional LFA criterion used in other protocols. Several inequalities (RFC5286) are used to determine if a candidate backup path is, in fact, loop-free. Only the first three inequalities apply in this scenario. The terms S, E, N, and D are used as follows.

1. **S:** The source router (R1, the local leaf)
2. **E:** The primary next-hop router (R5, the primary spine)
3. **N:** The candidate next-hop routers (R4 and R6, the other spines)
4. **D:** The destination prefix (R5-R8 link, the BGP next-hop covered by 10.5.8.0/24)
5. **Dist(X,Y):** Function that computes the IGP "distance" (path cost) between X and Y

The next sections explain the criteria that must be satisfied to form LFAs of various types.

2.1.3. Inequality 1: Determining if a Path is an LFA or Not

Inequality 1 determines whether a path is an LFA or not. For simplicity, assume all interfaces have an OSPF cost of 1. This includes transit links, loopbacks, and server LAN segments.

```
Dist(N,D) < Dist(N,S) + Dist(S,D)
2 < 1 + 2 --> TRUE
```

This inequality determines whether a candidate next-hop (N) is an LFA towards the destination (D). Dist(N,S) is the cost from the next-hop back to the source (S) and Dist(S,D) is the original cost from the source to the destination. Therefore, this LFA asserts that the cost from the candidate next-hop to the destination is strictly less (shorter) than the sum of the candidate's cost back to the source and the original end-to-end cost. Simply put, the candidate must be closer to the final destination than we are while also not looping through the original source node.

```
! Measures Dist(N,D)
R6#show ip ospf rib 10.5.8.0 255.255.255.0 | include cost
*>  10.5.8.0/24, Intra, cost 2, area 0

! Measures Dist(N,S)
R6#show ip ospf border-routers | include 10.0.0.1
i 10.0.0.1 [1] via 10.1.6.1, Ethernet0/2, ABR, Area 0, SPF 8
```

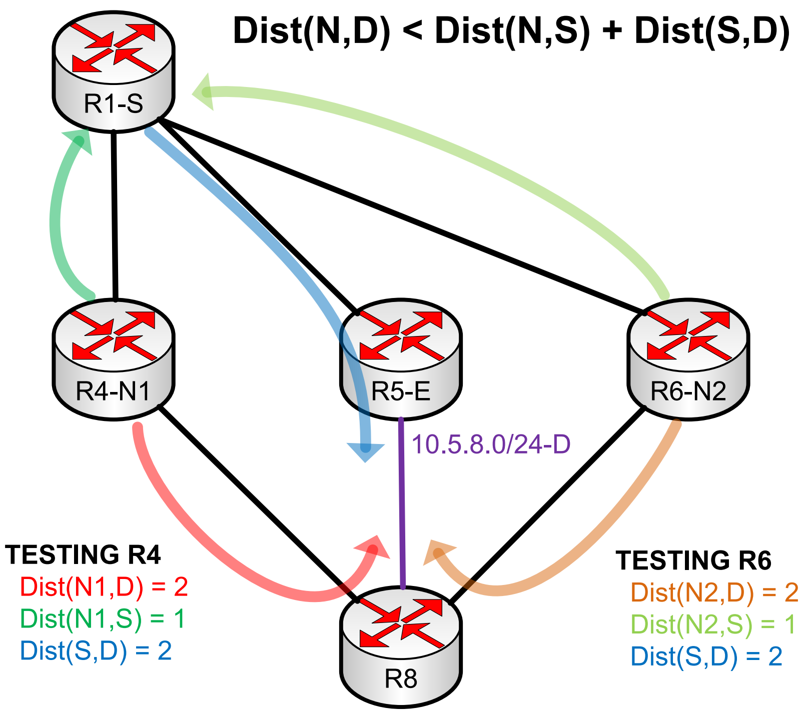
```
! Measures Dist(S,D)
R1#show ip ospf rib 10.5.8.0 255.255.255.0 | include cost
*>  10.5.8.0/24, Intra, cost 2, area 0
```

This inequality is relaxing the "feasibility condition" condition used in Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP). Distance/path vector protocols cannot evaluate Dist(N,S) since this relies upon a router to know the local cost of its connected peer. This is impossible without additional topology information being carried. For link-state protocols, this information is already known and retained in the link-state database. The number of potential LFAs through a leaf/spine fabric with N spines is N-1, yet only one will be selected. The debug confirms that both R4 and R6 can serve as LFA paths for 10.5.8.0/24, the transit link between R5 and R8.

```
R1#debug ip ospf fast-reroute rib
OSPF Loop-free FastReroute local RIB debugging is on

OSPF-1 FRRIB: Add to LRIB repair path 10.5.8.0/255.255.255.0 via
neighbor 10.0.0.4, area 0, type Intra, D(N,D)=2, ext2 metric 0
OSPF-1 FRRIB: For primary path via 10.1.5.5 Gi2 dist 2 repair path via
10.1.4.4 Gi1 flags are (Repair, IntfDj, BcastDj, NodeProt)

OSPF-1 FRRIB: Add to LRIB repair path 10.5.8.0/255.255.255.0 via
neighbor 10.0.0.6, area 0, type Intra, D(N,D)=2, ext2 metric 0
OSPF-1 FRRIB: For primary path via 10.1.5.5 Gi2 dist 2 repair path via
10.1.6.6 Gi3 flags are (Repair, IntfDj, BcastDj, NodeProt)
```

The diagram below illustrates this LFA inequality using color-coded arrows to indicate the various distance measurements. The various router roles (S, E, N, and D) have been appended to the router and prefix names as appropriate to aid in understanding.

Figure 7 - LFA Inequality 1; Basic Condition



2.1.4. Inequality 2: Determining if an LFA is Downstream

Inequality 2 is a stricter check on an LFA to see if N is closer to D than S. This inequality is almost identical to inequality 1 except removes Dist(N,S) from the formula.

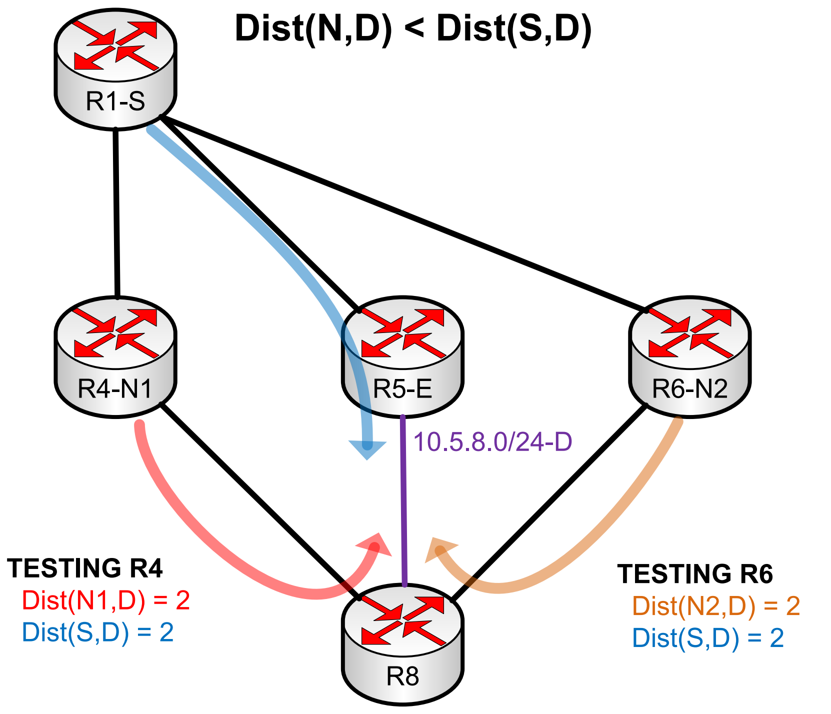
```
Dist(N,D) < Dist(S,D)
2 < 2 --> FALSE
```

This is logically identical to EIGRP's feasibility condition. If this inequality is true, the LFA is considered "downstream", which guarantees that micro-loops cannot form in the network. Within a leaf/spine fabric, assuming all costs are uniform, such micro-loops in these failure scenarios are impossible anyway. This inequality is highly applicable to less symmetric designs.

```
! Measures Dist(N,D)
R6#show ip ospf rib 10.5.8.0 255.255.255.0 | include cost
*>  10.5.8.0/24, Intra, cost 2, area 0

! Measures Dist(S,D)
R1#show ip ospf rib 10.5.8.0 255.255.255.0 | include cost
*>  10.5.8.0/24, Intra, cost 2, area 0
```

Note that, because this inequality is false, EIGRP would be a poor choice of IGP for this design since the depicted backup paths are not feasible successors. Dist(N,D) is equivalent to the EIGRP reported distance (RD) and Dist(S,D) is equivalent to the EIGRP feasible distance (FD). When they are equal, the feasibility condition is not satisfied. The diagram below illustrates inequality 2, proving that neither R4 nor R6 are downstream nodes.

Figure 8 - LFA Inequality 2; Downstream



2.1.5. Inequality 3: Determining Whether an LFA is Node-protecting

Inequality 3 determines whether an LFA offers node protection:

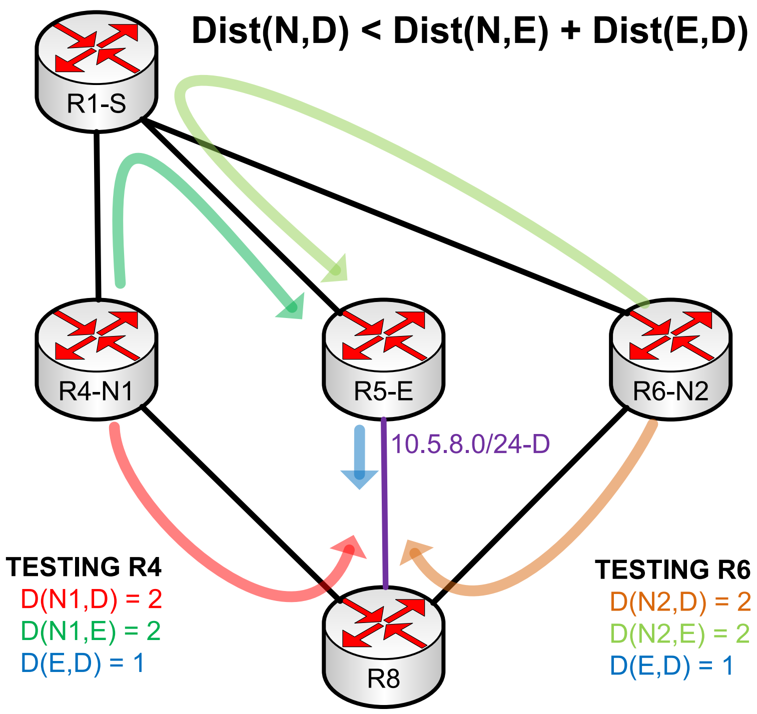
```
Dist(N,D) < Dist(N,E) + Dist(E,D)
2 < 2 + 1 --> TRUE
```

This means that the link R1-R5 or the node R5 can fail and the LFA will protect both cases by rerouting through R4 or R6. While all LFAs are link-protecting, only LFAs that satisfy this inequality are node-protecting. This extra protection ensures that a spine router failure minimally affects forwarding. The inequality is ensuring that N's (R4/R6) path to D (10.5.8.0/24) does not traverse E, which is true. Note that when using IS-IS with the overload-bit, Dist(N,E) cannot be evaluated at all (R4/R6 to R5), as this is a spine-to-spine calculation. The LFA process cannot consider the LFA as officially node-protecting, even though it is. Therefore, OSPF is best used for demonstrations of these inequalities, although from a performance perspective, there is no LFA-related difference regarding availability. Also, note that Dist(E,D) has a value of one, not zero, despite it being a “connected” route. The local OSPF interface cost is used instead of the RIB cost.

```
! Measures Dist(N,D)
R6#show ip ospf rib 10.5.8.0 255.255.255.0 | include cost
*>  10.5.8.0/24, Intra, cost 2, area 0

! Measures all possible Dist(N,E) paths
R6#show ip ospf border-routers | include 10.0.0.4
i 10.0.0.4 [2] via 10.1.6.1, Ethernet0/2, ABR, Area 0, SPF 11
i 10.0.0.4 [2] via 10.2.6.2, Ethernet0/0, ABR, Area 0, SPF 11
i 10.0.0.4 [2] via 10.3.6.3, Ethernet0/1, ABR, Area 0, SPF 11
i 10.0.0.4 [2] via 10.6.7.7, Ethernet1/2, ABR, Area 0, SPF 11
i 10.0.0.4 [2] via 10.6.8.8, Ethernet1/0, ABR, Area 0, SPF 11
i 10.0.0.4 [2] via 10.6.9.9, Ethernet1/1, ABR, Area 0, SPF 11

! Measures Dist(E,D)
R5#show ip ospf interface brief | include Cost|10.5.8.
Interface    PID   Area   IP Address/Mask    Cost  State Nbrs F/C
Et1/2        1     0      10.5.8.5/24        1     P2P   1/1
```

The RFC defines a fourth inequality which is only relevant on multi-access networks where a pseudonode exists. This is irrelevant in leaf/spine topologies and is not discussed here. The diagram below illustrates inequality 3, proving that both R4 and R6 are node-protecting LFAs.

Figure 9 - LFA Inequality 3; Node Protection



2.1.6. LFA Verification and Troubleshooting

The output below shows the result of the LFA in question. Note that the LFA is node-protecting but not downstream. Because the BGP TE mechanisms do not adjust IGP, there is no simple way to choose one LFA over another when they tie. This is largely irrelevant in a leaf/spine topology. LFAs need only be computed for BGP next-hops, which are always transit links in this design. Individual device loopbacks and server LANs may be excluded from the LFA calculation to conserve computing resources on leaf devices.

```
R1#show ip ospf rib 10.5.8.0 255.255.255.0 | begin 10.5.8.0
*>  10.5.8.0/24, Intra, cost 2, area 0
      SPF Instance 34, age 00:02:53
      Flags: RIB
       via 10.1.5.5, GigabitEthernet2
        Flags: RIB
        LSA: 1/10.0.0.5/10.0.0.5
        Source: 10.0.0.5 (area 0)
       repair path via 10.1.6.6, GigabitEthernet3, cost 3
        Flags: RIB, Repair, IntfDj, BcastDj, NodeProt, LoadShare
        LSA: 1/10.0.0.8/10.0.0.8
       repair path via 10.1.4.4, GigabitEthernet1, cost 3
        Flags: Ignore, Repair, IntfDj, BcastDj, NodeProt, LoadShare
        LSA: 1/10.0.0.8/10.0.0.8
```

The success of the BTS policy and the installation of the LFA is confirmed below. The elephant flow has been offloaded to R5 by following the BGP-injected /28 route via 10.5.8.5.

```
R1#show ip cef 172.16.8.10 detail
172.16.8.0/28, epoch 2
  recursive via 10.5.8.5
    recursive via 10.5.8.0/24
      nexthop 10.1.5.5 GigabitEthernet2
        repair: attached-nexthop 10.1.6.6 GigabitEthernet3
      nexthop 10.1.6.6 GigabitEthernet3, repair

R1#show ip cef exact-route 172.16.1.10 172.16.8.10
172.16.1.10 -> 172.16.8.10 =>IP adj out of GigabitEth2, addr 10.1.5.5
```

All other traffic destined for 172.16.8.0/24 that falls outside of 172.16.8.0/28 is forwarded using IGP-derived ECMP paths, as expected. ECMP utilization is the default OSPF behavior which requires no additional modification on Cisco routers.

```
R1#show ip route 172.16.8.100 | include entry|from
Routing entry for 172.16.8.0/24
  Last update from 10.1.4.4 on GigabitEthernet1, 00:20:58 ago
  * 10.1.6.6, from 10.0.0.8, 00:20:58 ago, via GigabitEthernet3
    10.1.5.5, from 10.0.0.8, 00:20:58 ago, via GigabitEthernet2
    10.1.4.4, from 10.0.0.8, 00:20:58 ago, via GigabitEthernet1
```

The routing lookup on R5, the new transit spine, reveals no new information. The path to R8 remains via the direct R5-R8 link using IGP; this will never change. The spines are guaranteed to always route traffic correctly whether BGP state exists on the leaves or not.

```
R5#show ip route 172.16.8.0
Routing entry for 172.16.8.0/24
  Known via "ospf 1", distance 110, metric 2, type inter area
  Last update from 10.5.8.8 on Ethernet1/2, 00:13:12 ago
  Routing Descriptor Blocks:
  * 10.5.8.8, from 10.0.0.8, 00:13:12 ago, via Ethernet1/2
      Route metric is 2, traffic share count is 1
```

It is worth noting that the reverse path from R8 to R1 remains unchanged. Since the elephant flow was unidirectional, there is no operational need to create symmetric routing across the fabric. R8 continues to load-share using all available paths, providing higher bandwidth availability and automatic FRR without LFAs. Both R5 and R8 remain unaware that "exception routing" has been introduced on R1. If the elephant flow was bidirectional, the same BGP route injection technique can be used on R8 to influence its path to R1. This is not demonstrated as the process is identical in the reverse direction.

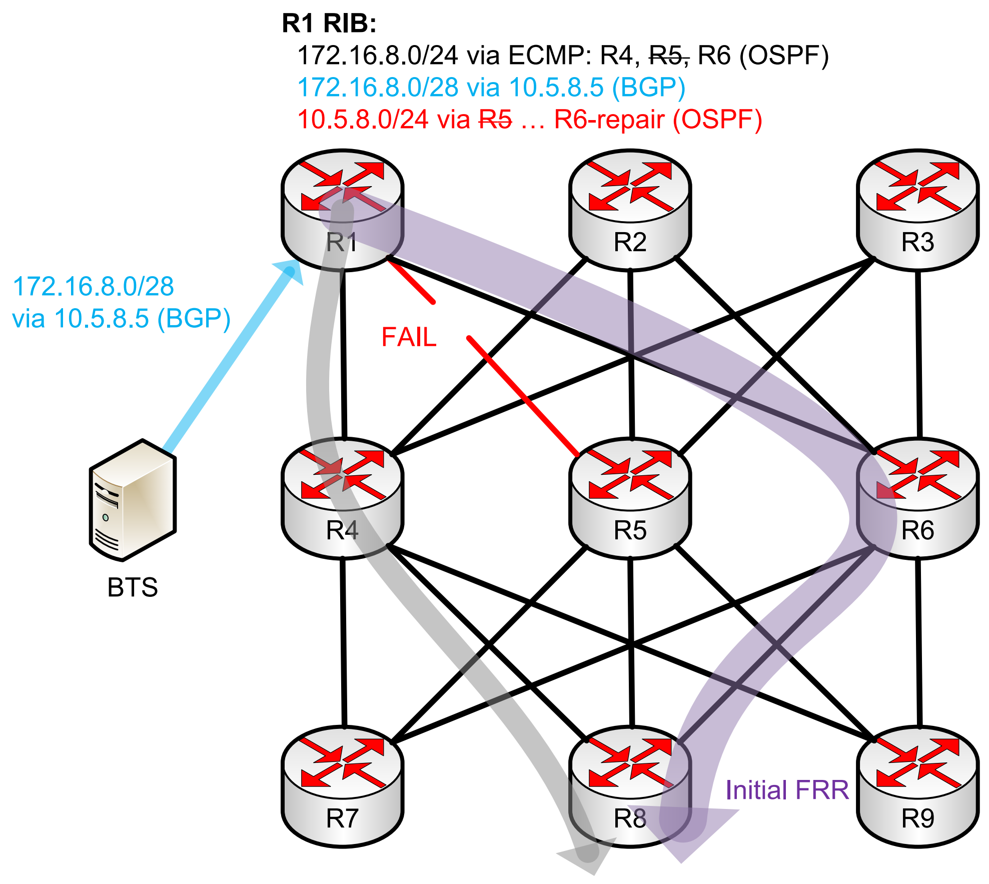
```
R8#show ip route 172.16.1.10 | include entry|from
Routing entry for 172.16.1.0/24
  Last update from 10.6.8.6 on Ethernet1/0, 00:09:08 ago
    10.6.8.6, from 10.0.0.1, 00:09:08 ago, via Ethernet1/0
  * 10.5.8.5, from 10.0.0.1, 00:09:08 ago, via Ethernet1/2
    10.4.8.4, from 10.0.0.1, 00:09:08 ago, via Ethernet1/1
```

2.1.7. Failure Scenarios

It's important to consider how a network will react to various failures. This section illustrates three failures that significantly impact how BTS-injected routes operate.

First, consider a local link failure. This link connects the source leaf (R1) to the preferred spine (R5) based on a BTS announcement. Initially, traffic towards 172.16.8.0/28 will prefer the LFA path via R6 as indicated by the outputs above. This minimizes traffic loss while IGP reconverges. Once IGP reconvergence is complete, both R4 and R6 will be used for ECMP because 10.5.8.0/24 is equidistant from both remaining spines. This load-sharing nullifies the effectiveness of BGP traffic engineering for the duration of the link failure but minimizes packet loss across the fabric. The diagram below illustrates this failure scenario. Note that the "initial FRR" arrow might be infinitesimally short-lived as some platforms may rapidly utilize both LFAs via R4 and R6 despite only one being installed in the FIB.

Figure 10 - Handling Local Link Failures

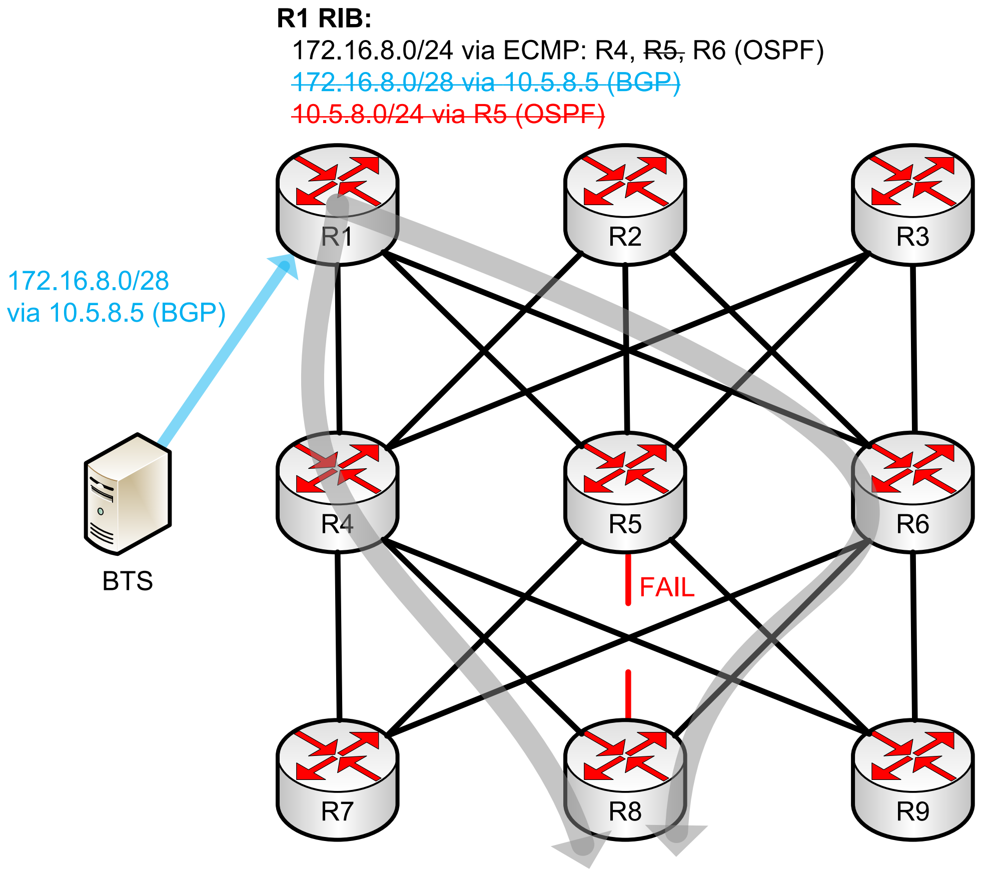

R1 RIB:
172.16.8.0/24 via ECMP: R4, ~~R5,~~ R6 (OSPF)
172.16.8.0/28 via 10.5.8.5 (BGP)
10.5.8.0/24 via ~~R5~~ … R6-repair (OSPF)
172.16.8.0/28
via 10.5.8.5 (BGP)
FAIL
BTS
R1
R2
R3
R4
R5
R6
R7
R8
R9
Initial FRR

After the network converges, the BGP route remains intact as its next-hop of 10.5.8.5 is still accessible. Both R4 and R6 are used in an ECMP fashion while referencing each other as LFAs.

```
R1#show ip cef 172.16.8.0/28 detail
172.16.8.0/28, epoch 2
  recursive via 10.5.8.5
    recursive via 10.5.8.0/24
      nexthop 10.1.4.4 GigabitEthernet1
        repair: attached-nexthop 10.1.6.6 GigabitEthernet3
      nexthop 10.1.6.6 GigabitEthernet3
        repair: attached-nexthop 10.1.4.4 GigabitEthernet1
```

Next, consider what happens if the R5-R8 link fails, which is the remote link between the preferred spine (R5) and target leaf (R8). This will cause the 10.5.8.0/24 prefix to be removed from the topology, making the BGP next-hop for 172.16.8.0/28 unreachable. Assuming BGP next-hop tracking (NHT) is enabled, which is the default on modern devices, the BGP route will be quickly removed from the RIB due to an inaccessible next-hop. Additionally, R5 is no longer a suitable ECMP path using OSPF, so traffic from R1 to R8 will be load-shared across R4 and R6. The diagram depicts traffic flowing over both paths, although the elephant flow in question would choose only one. Although suboptimal, this at least ensures a fast failover to ECMP rather than wholesale loss of traffic. This failure is more severe than the local link failure as R1 must be aware of 10.5.8.0/24 being withdrawn from IGP, making IGP convergence timing tuning more important. The diagram below illustrates this failure case.

Figure 11 - Handling Remote Link Failures


R1 RIB:
172.16.8.0/24 via ECMP: R4, ~~R5,~~ R6 (OSPF)
~~172.16.8.0/28 via 10.5.8.5 (BGP)~~
~~10.5.8.0/24 via R5 (OSPF)~~
172.16.8.0/28
via 10.5.8.5 (BGP)
BTS
R1
R2
R3
R4
R5
R6
R7
R8
R9
FAIL

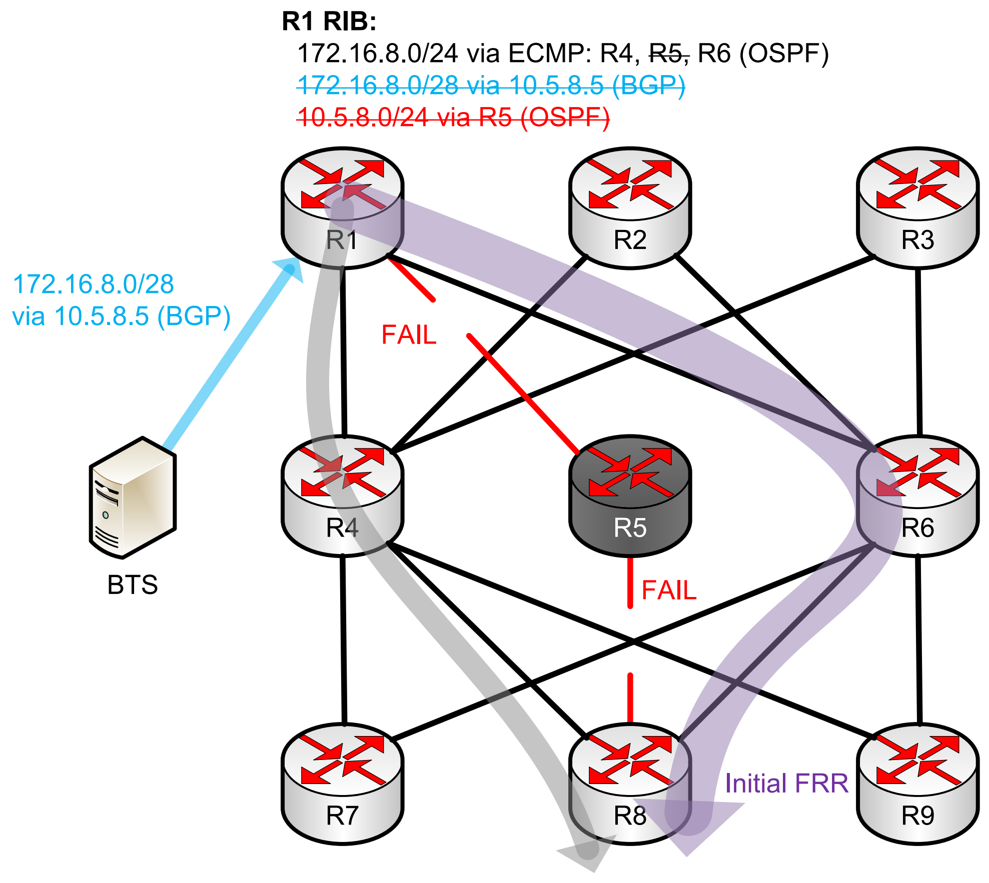
When 10.5.8.5 is inaccessible, the BTS-injected BGP route 172.16.8.0/28 cannot be used, and once IGP reconverges, R1 load shares across R4 and R6 just like the previous failure scenario.

```
R1#show ip cef 172.16.8.0/28 detail
172.16.8.0/28, epoch 2
  recursive via 10.5.8.5, unresolved

R1#show ip cef 172.16.8.10 detail
172.16.8.0/24, epoch 2, per-destination sharing
  nexthop 10.1.4.4 GigabitEthernet1
    repair: attached-nexthop 10.1.6.6 GigabitEthernet3
```

```
nexthop 10.1.6.6 GigabitEthernet3
  repair: attached-nexthop 10.1.4.4 GigabitEthernet1
```

Last, consider how the network will react when the preferred spine (R5) suffers a node failure. This has the additional effect of bringing both the local link (R1-R5) and remote link (R5-R8) down as well. Given that R1 will probably detect its local link failure before anything else, it will fail over to the LFA path via R6 immediately. Shortly thereafter, the 10.5.8.0/24 will be withdrawn form IGP, the BTS-injected BGP route will be unusable due to an inaccessible next-hop, and R1 will fall back to using the IGP-learned route for 172.16.8.0/24 for ECMP across all remaining spines. This behavior more closely resembles the local link failure than the remote link failure, at least initially. Once the remote link prefix of 10.5.8.0/24 becomes unreachable, the BGP route 172.16.8.0/28 becomes unusable. The diagram below illustrates this failure scenario where R5 suffers a node failure.

Figure 12 - Handling Preferred Spine (Node) Failures


R1 RIB:
172.16.8.0/24 via ECMP: R4, ~~R5,~~ R6 (OSPF)
~~172.16.8.0/28 via 10.5.8.5 (BGP)~~
~~10.5.8.0/24 via R5 (OSPF)~~
R1
R2
R3
172.16.8.0/28
via 10.5.8.5 (BGP)
FAIL
R4
R5
R6
BTS
FAIL
R7
R8
Initial FRR
R9

2.1.8. Failed Alternative: Using Spine Loopbacks as BGP Next-hops

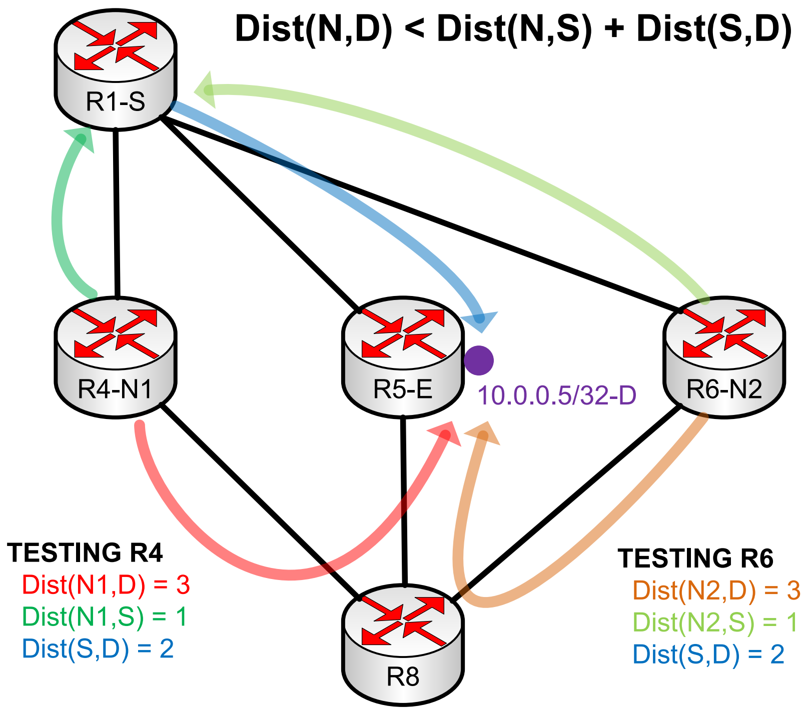
It is more intuitive for BTS-injected BGP routes to identify the loopback of the targeted spine rather than a remote transit link. This design also has the advantage of working in environments whereby the transit links are filtered from routing reachability. However, there is a significant operational drawback; leaves will never have LFA paths available for spine loopbacks. This is seldom a problem because no services are hosted from spines; there is no requirement or expectation of highly-available connectivity to these devices within the fabric.

Recall that inequality 1 must be satisfied for any path to be considered an LFA. Dist(N,S) and Dist(S,D) remain the same as the previous example, but Dist(N,D) has increased. The candidate next-hops, which are the non-primary spines, could potentially loop through S to reach D. Therefore, these paths can never be LFAs.

```
Dist(N,D) < Dist(N,S) + Dist(S,D)
3 < 1 + 2 --> FALSE
```

The diagram below illustrates the problem. Though the red and orange lines representing N-to-D paths traverse R8 in the diagram, they could just as easily transit any leaf, including R1 (S). Traversing R1 to reach 10.0.0.5/32, which is R5's loopback (D), means that the path is not guaranteed to be loop free.

Figure 13 - Inequality 1 Not Satisfied for Spine Loopbacks



This is further proven by examining the OSPF RIB for R5's loopback from the perspective of any leaf in the fabric. Unlike the remote transit links, R1 has no LFA paths for this destination.

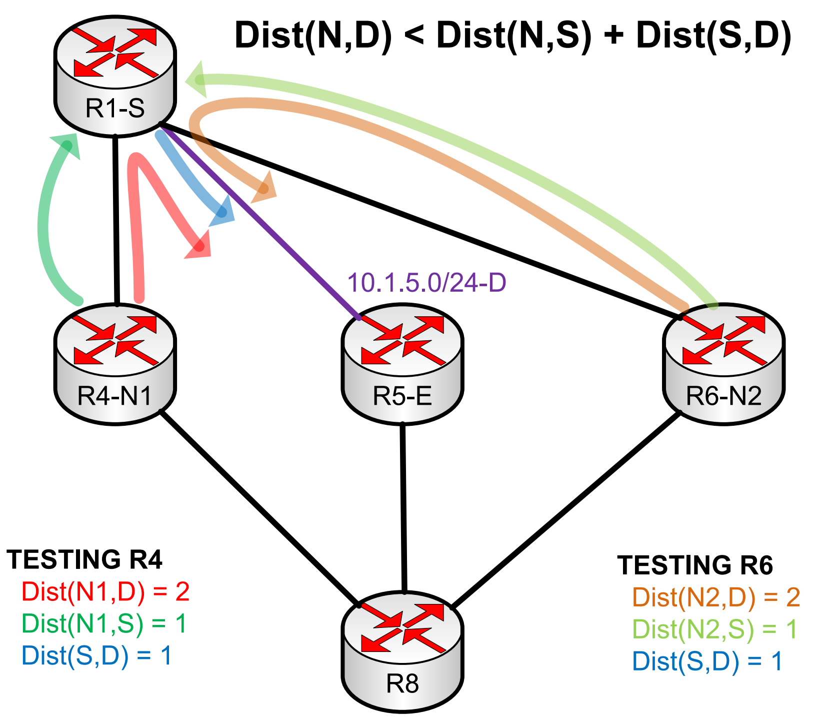
```
R1#show ip ospf rib 10.0.0.5 255.255.255.255 | begin 10.0.0.5
*>  10.0.0.5/32, Inter, cost 2, area 0
     SPF Instance 22, age 00:00:20
      contributing LSA: 3/10.0.0.5/10.0.0.5 (area 0)
     Flags: RIB, HiPrio
      via 10.1.5.5, GigabitEthernet2
       Flags: RIB
       LSA: 3/10.0.0.5/10.0.0.5
       Source: 10.0.0.5 (area 0)


R1#debug ip ospf fast-reroute rib
OSPF Loop-free FastReroute local RIB debugging is on

OSPF-1 FRRIB: Add to LRIB repair path 10.0.0.5/255.255.255.255 via
neighbor 10.0.0.4, area 0, type Inter, D(N,D)=3, ext2 metric 0
OSPF-1 FRRIB: Not an LFA; D(N,D)=3, D(N,S)=1, D(S,D)=2

OSPF-1 FRRIB: Add to LRIB repair path 10.0.0.5/255.255.255.255 via
neighbor 10.0.0.6, area 0, type Inter, D(N,D)=3, ext2 metric 0
OSPF-1 FRRIB: Not an LFA; D(N,D)=3, D(N,S)=1, D(S,D)=2
```

2.1.9. Failed Alternative: Using Local Link IPs as BGP Next-hops

Another alternative is to use the directly connected interface IP addresses as BGP next-hops. Using the same example from earlier, this would use a BGP next-hop of 10.1.5.5, the IP address of R5 on the R1-R5 link, for 172.16.8.0/28 as injected by the BTS. Much like the previous example where it was proven that leaves will never have LFAs for spine loopbacks, leaves will also never have LFAs for local link subnets.

The diagram below visually explains the reason. The shortest path from any non-connected spine (R4 or R6) to reach the local link connecting the source leaf (R1) and preferred spine (R5) will always route through the source leaf. Said another way, R4/R6 will always route through R1 to reach a network that is connected to R1 as opposed to transiting some other leaf. The diagram makes this clear with the red and orange arrows as the shortest Dist(N,D) always routes through R1. Note that many routers would never consider LFAs for connected routes anyway, so this inequality evaluation is likely theoretical more so than practical.

Figure 14 - Inequality 1 Not Satisfied for Direct Local Links

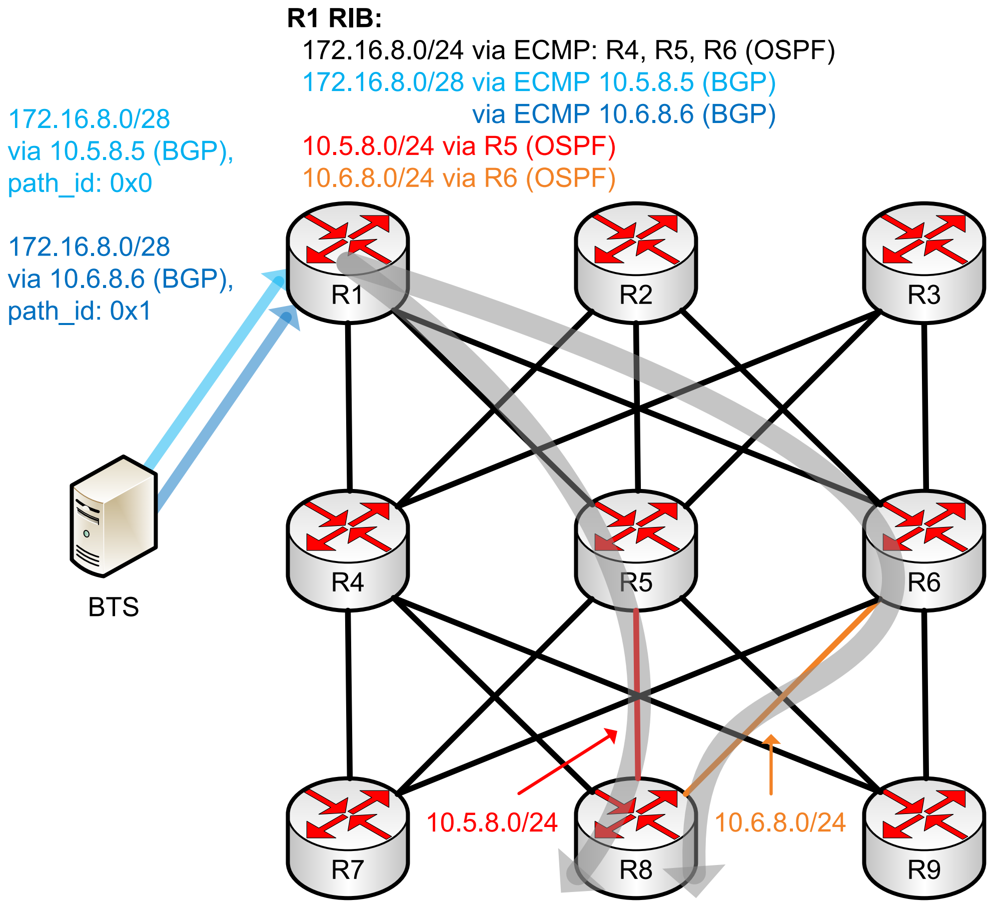


2.2. Steering Multiple TE Paths

Some environments require more complicated load-sharing techniques, including both ECMP and unequal-cost multi-path (UCMP) designs. Although a single flow cannot be distributed across multiple spines, BGP can inject multiple next-hops for a single prefix to utilize a subset of spines rather than only a single spine.

From a BGP perspective, this can be accomplished in two ways. The simpler and more modern approach is to use BGP additional-paths. This enables BGP to advertise multiple paths for a single prefix across a single session. The paths can carry different attributes, such as next-hops and communities, allowing a single BTS (or a single pair of BTS' for availability) to enable ECMP and UCMP designs. The diagram below depicts a single BTS announcing 172.16.8.0/28 with next-hops of 10.5.8.5 and 10.6.8.6, effectively selecting R5 and R6 for load sharing towards a subset of R8's server LAN. Each path carries a unique 32-bit path ID for differentiation.

Figure 15 - Single BTS using BGP Additional-Paths


R1 RIB:
172.16.8.0/24 via ECMP: R4, R5, R6 (OSPF)
172.16.8.0/28 via ECMP 10.5.8.5 (BGP)
via ECMP 10.6.8.6 (BGP)
10.5.8.0/24 via R5 (OSPF)
10.6.8.0/24 via R6 (OSPF)
172.16.8.0/28 via 10.5.8.5 (BGP), path_id: 0x0
172.16.8.0/28 via 10.6.8.6 (BGP), path_id: 0x1
R1
R2
R3
R4
R5
R6
R7
R8
R9
BTS
10.5.8.0/24
10.6.8.0/24

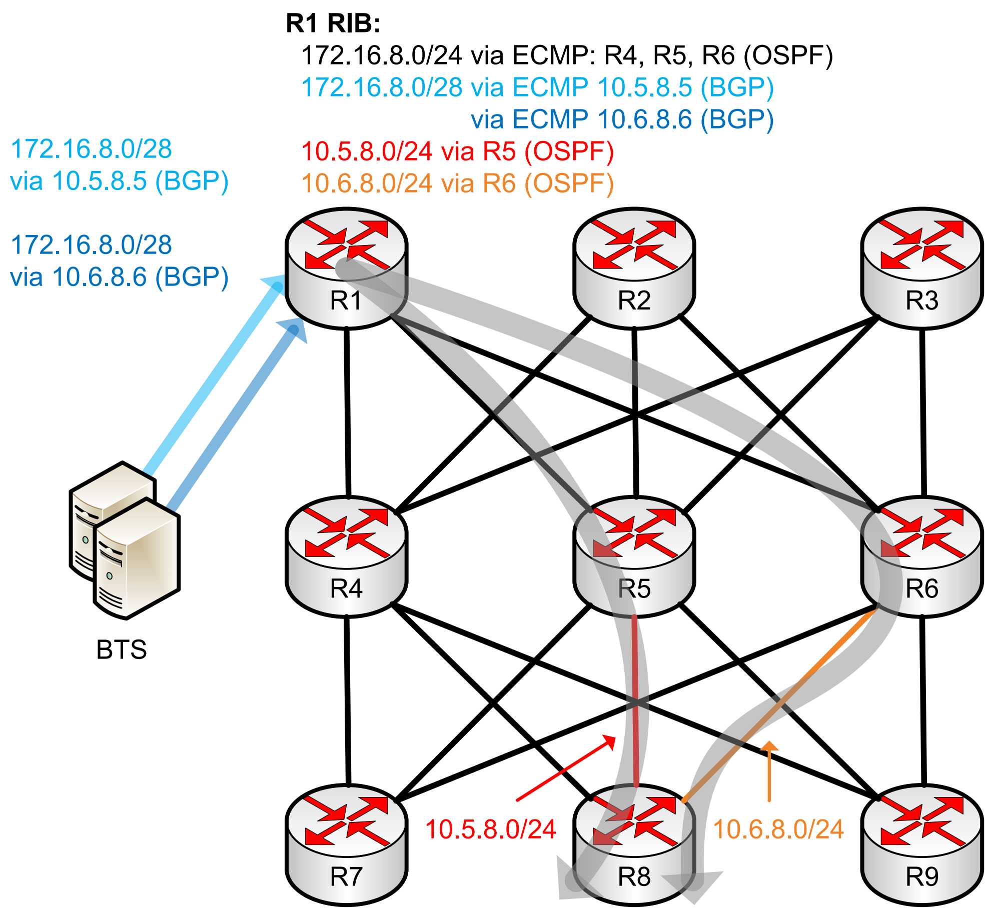
This example is easily demonstrated using ECMP. The existing BTS has negotiated BGP additional-paths to each leaf and announces two versions of 172.16.8.0/28 to R1. One has a next-hop of 10.5.8.5 and one has a next-hop of 10.6.8.6, ensuring that R5 and R6 are used in equal proportion as transit spines. Each leaf must enable iBGP multipathing for at least two paths for load-sharing to occur. For those interested in minimalist implementations, the BTS must offer the “send additional-paths” capability while the leaves must offer the “receive additional-paths” capability. Because BGP announcements and withdrawals occur in one direction (from BTS to leaf), BGP speakers don’t require bidirectional additional-path exchange.

```
R1#show bgp ipv4 unicast neighbors 192.168.0.99 | include Additional
  Additional Paths send capability: received
  Additional Paths receive capability: advertised

R1#show bgp ipv4 unicast 172.16.8.0/28
BGP routing table entry for 172.16.8.0/28, version 24
Paths: (2 available, best #2, table default)
Multipath: iBGP
```

```
Not advertised to any peer
Refresh Epoch 1
Local
  10.6.8.6 (metric 2) from 192.168.0.99 (10.0.0.99)
    Origin IGP, localpref 100, valid, internal, multipath(oldest)
    rx pathid: 0x1, tx pathid: 0
Refresh Epoch 1
Local
  10.5.8.5 (metric 2) from 192.168.0.99 (10.0.0.99)
    Origin IGP, localpref 100, valid, internal, multipath, best
    rx pathid: 0x0, tx pathid: 0x0
```

The second option is a more traditional approach which simply adds more BTS' with divergent policies. If the additional-paths feature is unavailable or undesirable, each BTS (or each pair of BTS' for availability) will announce a separate route with separate attributes, such as next-hops and communities. This approach does not require any new technology but scales poorly as each concurrent announcement requires an additional BTS or BTS pair. The diagram below depicts two BTS' whereby each BTS announces 172.16.8.0/28 with two different next-hops. The operational effect is identical to the BGP additional-path option.

Figure 16 - Dual BTS using Unique Advertisements


R1 RIB:
172.16.8.0/24 via ECMP: R4, R5, R6 (OSPF)
172.16.8.0/28 via ECMP 10.5.8.5 (BGP)
via ECMP 10.6.8.6 (BGP)
10.5.8.0/24 via R5 (OSPF)
10.6.8.0/24 via R6 (OSPF)
172.16.8.0/28 via 10.5.8.5 (BGP)
172.16.8.0/28 via 10.6.8.6 (BGP)
R1
R2
R3
R4
R5
R6
R7
R8
R9
BTS
10.5.8.0/24
10.6.8.0/24

BGP supports UCMP using the "bandwidth" extended community. This is sometimes known as the "Demilitarized Zone (DMZ) link bandwidth" community. The exact values of the bandwidth extended community are irrelevant as only the ratio matters. Note that omitting the bandwidth extended community would result in ECMP, which is another valid (but less granular) option. The output below illustrates the second option with two divergent BTS' devices, each announcing the same prefix with different next-hop and bandwidth community values.

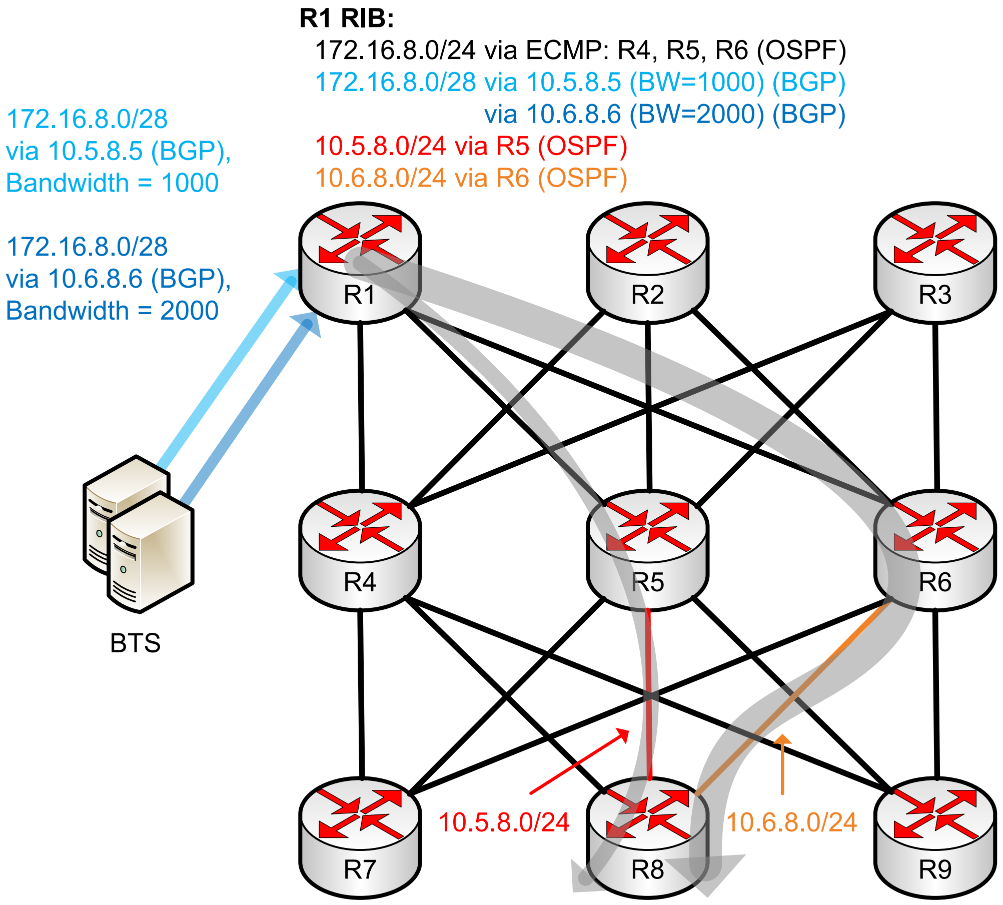
```
R1#show bgp ipv4 unicast 172.16.8.0/28
BGP routing table entry for 172.16.8.0/28, version 4
Paths: (2 available, best #2, not advertised to any peer)
Multipath: iBGP
  Not advertised to any peer
  Refresh Epoch 1
  Local
    10.6.8.6 (metric 2) from 192.168.0.98 (10.0.0.98)
      Origin IGP, localpref 100, valid, internal, multipath(oldest)
      DMZ-Link Bw 2000 kbytes
      rxpathid: 0, txpathid: 0
  Refresh Epoch 1
  Local
    10.5.8.5 (metric 2) from 192.168.0.99 (10.0.0.99)
      Origin IGP, localpref 100, valid, internal, multipath, best
      DMZ-Link Bw 1000 kbytes
      rxpathid: 0, txpathid: 0x0
```

The result of this unequal-cost weighting ensures that for every two flows forwarded through R6, one flow will be forwarded through R5. This technique is useful for further distributing sets of flows confined to a contiguous subnet across multiple spines. This technique cannot be used to break apart a single flow, so it is best deployed when sets of flows must be exception-routed. All BGP next-hops will always be protected by node-protecting LFAs for additional fault tolerance as proven earlier. If, for example, the link between R6-R8 fails, the BGP route is immediately purged from the table without waiting for a BGP WITHDRAW message from the BTS. This is the result of IGP removing the prefix from the routing table once convergence is complete. The remaining BTS paths, such as the one via R5, will be used alone.

```
R1#show ip route 172.16.8.10
Routing entry for 172.16.8.0/28
  Known via "bgp 65001", distance 30, metric 0, type internal
  Last update from 10.5.8.5 00:01:16 ago
  Routing Descriptor Blocks:
  * 10.6.8.6, from 192.168.0.98, 00:01:16 ago
      Route metric is 0, traffic share count is 2
      AS Hops 0
      MPLS label: none
    10.5.8.5, from 192.168.0.99, 00:01:16 ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: none
```

The diagram below summarizes the relevant forwarding paths. Note that UCMP does not necessarily require multiple BTS devices. BGP additional-paths can carry extended communities within each additional-path if desired. Note that the hardware-level programming of UCMP routes is highly platform-dependent; even if the BGP table and RIB appear to show multi-pathing, be sure to test it before deploying it in production.

Figure 17 - BGP-influenced UCMP Across Multiple Paths


R1 RIB:
172.16.8.0/24 via ECMP: R4, R5, R6 (OSPF)
172.16.8.0/28 via 10.5.8.5 (BW=1000) (BGP)
via 10.6.8.6 (BW=2000) (BGP)
10.5.8.0/24 via R5 (OSPF)
10.6.8.0/24 via R6 (OSPF)
172.16.8.0/28 via 10.5.8.5 (BGP), Bandwidth = 1000
172.16.8.0/28 via 10.6.8.6 (BGP), Bandwidth = 2000
BTS
R1
R2
R3
R4
R5
R6
R7
R8
R9
10.5.8.0/24
10.6.8.0/24

3. Automated Traffic Management

While the aforementioned technical design is fully functional, it is not manageable at scale. No reasonable organization would require its operators to individually evaluate thousands (or millions) of flows and manually determine the best way to optimize bandwidth across the fabric. Even if this were possible, manually configuring all the BGP routing announcements and withdrawals is not feasible. This section discusses four automated approaches to BTS management. This is not a comprehensive list and other solutions certainly exist.

Python code for solutions 2 through 4 from this section can be found here: https://github.com/nickrusso42518/bts

3.1. Solution 1: Cisco IOS via Device Configuration

When this document was first published in 2016, automation was not even considered, let alone implemented. Instead, the BTS was simply a Cisco IOS router with manually configured BGP routes. This solution clearly will not scale and is only suitable for proof-of-concept testing.

However, the BTS can still be a regular router, just managed using automated techniques. Broadly speaking, such techniques fall into two categories:

1. **CLI-driven programmability:** A centralized system can log into the BTS using SSH and perform the BGP route configuration in an automated fashion. Common tools/libraries to accomplish this include Python paramiko, Python netmiko, Python scrapcli, Ansible using "network_cli" and associated "*_config" modules, and more. Ansible and Python Nornir can provide concurrency when there are multiple BTS devices to be managed.
2. **Model-driven programmability:** Rather than issuing commands to a device CLI through an automated mechanism, the underlying data can be modelled in a structured way. Modern devices use the Yet Another Next Generation (YANG) language to define the structure of the data, such as how the hierarchy of BGP route injection works. These models can be vendor-specific (native) or standards-based. Protocols such as NETCONF, RESTCONF, and Google Remote Procedure Call (gRPC) can transport YANG-modelled data, often encoded in JavaScript Object Notation (JSON) or eXtensible Markup Language (XML), between the device and automation system. This approach decouples device management from the underlying CLI commands which improves scale, efficacy, and more.

Network automation scripts to implement this type of solution can be found all over the Internet. They are not discussed in this document nor included in the code repository at this time. Despite such scripts being common, using traditional routers to serve as the BTS is unwise. The configuration is extensive, injecting/filtering routes for advertisement to specific peers is complex, and the solution is not nearly as scalable as the alternatives discussed next.

3.2. Solution 2: Exabgp via HTTP API

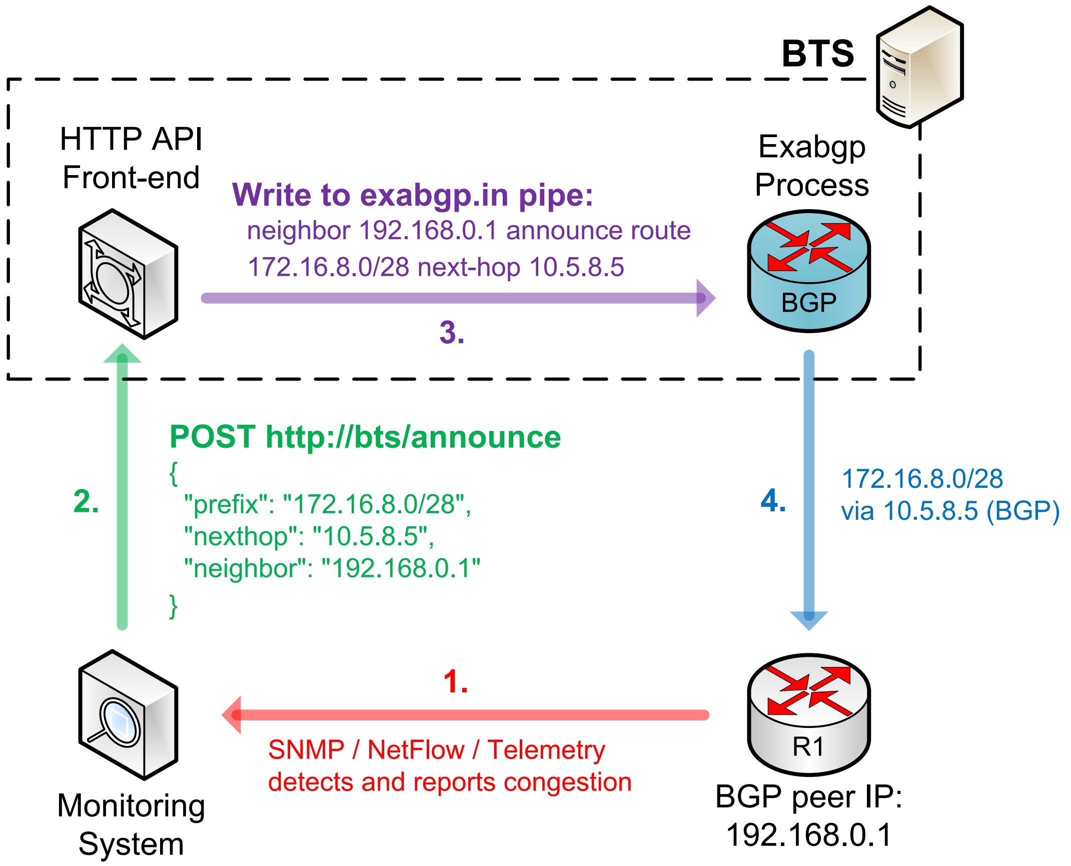
Instead of using a router for the simple purpose of injecting BGP routes, a network administrator can use a purpose-built solution instead. Exabgp is one such solution, an open-source BGP implementation written in Python. Using an intuitive configuration file, Exabgp forms neighbors with peers and offers a simple CLI to interact with the system. Exabgp does not come with an Application Programming Interface (API) but building a simple HyperText Transport Protocol (HTTP) front-end is relatively straightforward.

There are two main actions that any BTS implementation must support: the ability to "announce" (advertise) a route and the ability to "withdraw" (un-advertise) a route. The HTTP API should expose two endpoints to perform these actions. To keep things simple, the HTTP body contains a flat JSON dictionary containing key/value pairs to specific the prefix, next-hop, neighbor IP, and BGP additional-path ID. For announcements, the prefix and next-hop are minimally required. For withdrawals, only the prefix is minimally required.

When an elephant flow is detected, the automation system issues an HTTP POST request to the "/announce" endpoint containing the required parameters in the JSON body. The API then translates this request by assembling the proper Exabgp CLI command and sending it to the Exabgp process via an input pipe. Exabgp is very fast, so the API synchronously waits for Exabgp to respond, reading the reply text via an output pipe and ultimately returning it to the automation system. The process is identical for withdrawals.

The diagram below illustrates this API exchange process. Two other pieces of communication are not depicted for brevity: the Exabgp output from the announcement back to flask via an output pipe and flask's HTTP response to the monitoring system. Examples of these HTTP requests and their responses are available in the code repository in the form of a Postman collection.

Figure 18 - Automated Route Injection Architecture using Flask (HTTP) and Exabgp



Note that the specific implementation of the API is highly variable, and this document does not require any design. So long as announcements and withdrawals are easily signalled, any stable solution is acceptable. As a catch-all, creating a general-purpose endpoint to execute raw Exabgp commands is useful. While brittle, this approach unlocks the full power of the Exabgp CLI without having to define customer HTTP endpoints for every Exabgp command.

3.3. Solution 3: Exabgp via gRPC API

While HTTP APIs are easy to understand and operate, they often lead to divergent client implementations, especially across different programming languages. A Python-based client application might use the popular "requests" or "httpx" packages. The code might be further encapsulated into an object-oriented class with custom methods to perform common actions such as announce(), withdraw(), and more. Rebuilding such an object-oriented class (effectively an API wrapper) would be required in every non-Python language, complicating the adoption of HTTP APIs in multi-language environments or applications.

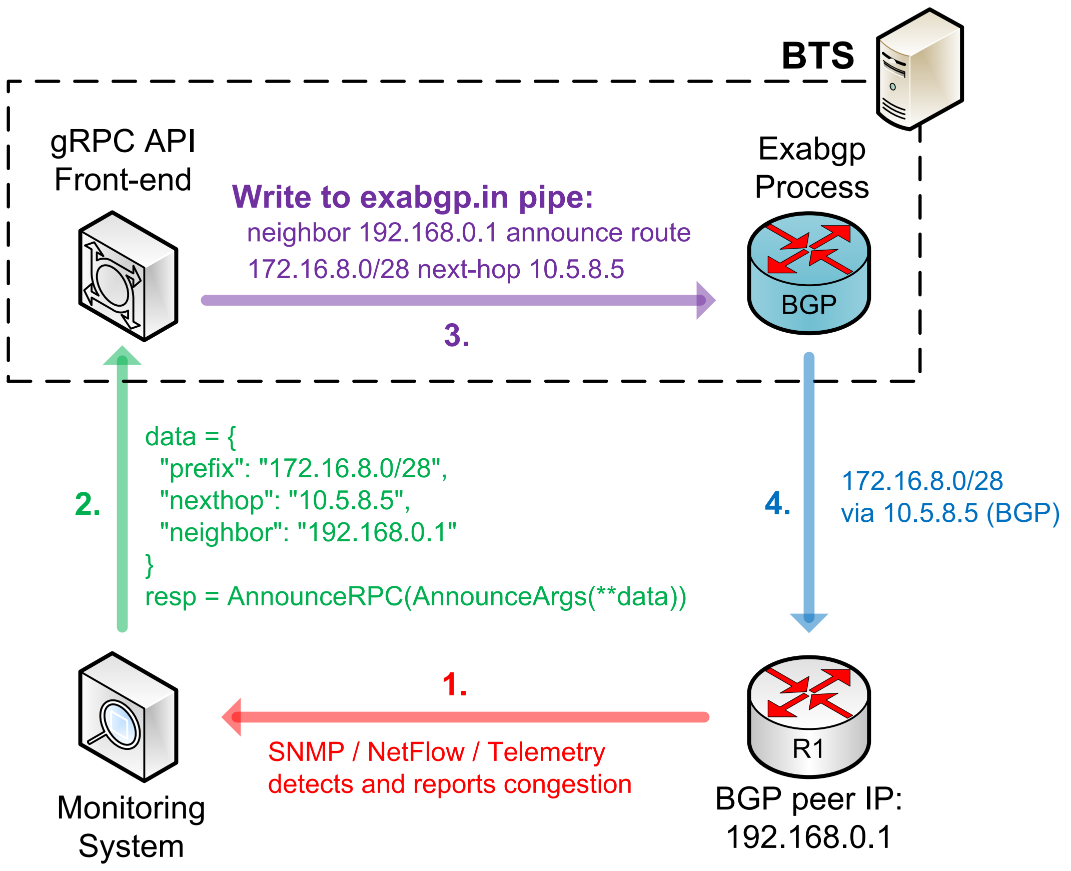
Google Remote Procedure Call (gRPC) solves this problem with a service definition file. Using a new mechanism called "protocol buffers", a programmer defines which RPCs (methods) should be available on the server, along with their inputs and outputs. This service definition file is then compiled into source code for a target language, such as C#, Java, Python, Go, and many more.

The resulting source code files include client and server-side implementations that handle the transmission and reception of RPCs, allowing for easy and well-formed inter-language communications. gRPC is based on HTTP 2.0 and is connection-oriented.

Logically, it makes sense to have an RPC defined in the services file for every HTTP API endpoint. Client applications will call the AnnounceRPC() method rather than issuing an HTTP POST to the /announce endpoint. Clients will include an AnnounceArgs object as an argument to AnnounceRPC() containing the attributes required for a new route announcement instead of including a JSON body in an HTTP request. Servers will respond with an AnnounceReply object instead of returning an HTTP body containing a status code and Exabgp output. The conceptual semantics are similar but the syntax and operating behaviors are different.

The diagram below illustrates this design. The data exchange with Exabgp works identically for gRPC APIs as it does for HTTP APIs. Upon receiving an RPC, the server unpacks the RPC arguments to build the proper Exabgp command. This command is then written to the input pipe, effectively transmitting the message to Exabgp. The server reads Exabgp's response from the output pipe and parses the text output into structured data aligning with the proper RPC reply object. For cleanliness, the gRPC response back to the monitoring system and the Exabgp command response back to the gRPC API are not depicted.

Figure 19 - Automated Route Injection Architecture using gRPC and Exabgp


BTS
gRPC API Front-end
Exabgp Process
Write to exabgp.in pipe:
neighbor 192.168.0.1 announce route 172.16.8.0/28 next-hop 10.5.8.5
3.
BGP
2.
4.
172.16.8.0/28 via 10.5.8.5 (BGP)
1.
R1
SNMP / NetFlow / Telemetry detects and reports congestion
Monitoring System
BGP peer IP: 192.168.0.1

```
data = {
  "prefix": "172.16.8.0/28",
  "nexthop": "10.5.8.5",
  "neighbor": "192.168.0.1"
}
resp = AnnounceRPC(AnnounceArgs(**data))
```

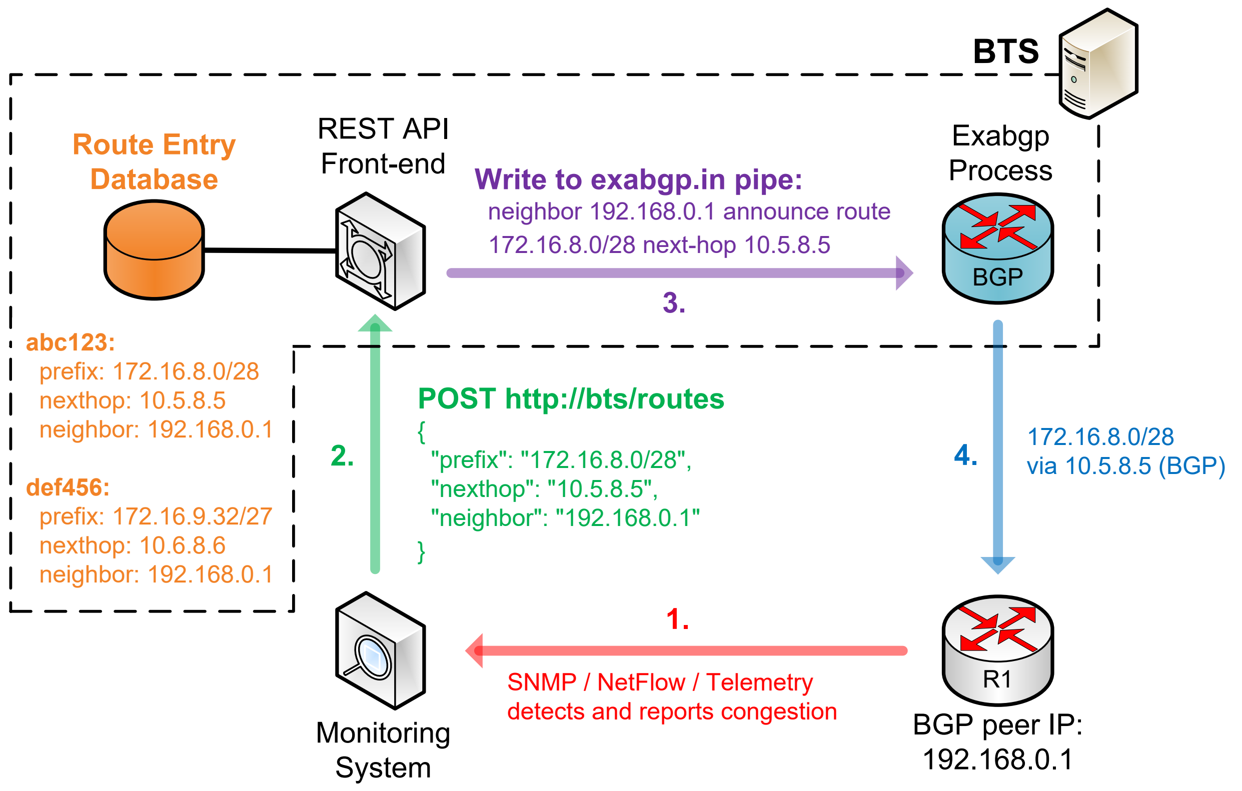
3.4. Solution 4: Exabgp via REST API

The HTTP and gRPC APIs discussed earlier both technically qualify as RPC-based APIs. A client requests that the server perform some action and supplies some set of arguments. When new routes are created, the client doesn't have direct access to those resources, but rather collects route information using generalized methods. In a Representation State Transfer (REST) architecture, each route must be directly accessible, along with many other constraints.

For example, an HTTP POST request to the /routes resource would create a new route. The new route would be accessible via its SHA-256 hash appended to the URL. Using the dummy hash of "abc123", issuing an HTTP GET request to /routes/abc123 would return the information specific to the newly-announced route. An HTTP GET request to /routes without any specified hash would return all announced routes. Last, an HTTP DELETE request to the /routes/abc123 resource would withdraw the specified route. Exabgp does not care about these hashes.

In a REST API design, endpoints such as /announce and /withdraw are no longer needed. The actions taken by the client target the collection of routes (the noun) rather than the individual operations that the API can perform (the verbs). Although the Flask package can be used to build REST APIs, the example included in the repository uses FastAPI for variety. The syntax and semantics are quite like Flask, but this package is specifically designed for HTTP-based APIs and not general-purpose web server features. The diagram below illustrates the REST API design. This includes a route database to retain the announced routes. In this simple example, routes are keyed by their unique SHA-256 hashes with the route attributes nested beneath it.

Figure 20 - Automated Route Injection Architecture using FastAPI (REST) and Exabgp



4. Complexity Assessment

This section objectively addresses the complexity of each solution using the State/Optimization/Surface (SOS) model. This model was formalized by White and Tantsura*("Navigating Network Complexity: Next-generation routing with SDN, service virtualization, and service chaining", R. White / J. Tantsura, Addison-Wesley 2016)* and is used as a quantifiable measurement of network complexity. This section is relevant when comparing the BTS solution to more advanced/complex software-defined networking (SDN) solutions.

4.1. State

State quantifies the amount of control-plane data present and the rate at which state changes in the network. While generally considered something to be minimized, some network state is always required. The manner in which a solution scales, typically with respect to time and/or space complexity, is a good measurement of network state.

The state retained across the fabric scales linearly as nodes are added. Assume that X is the number of spines and Y is the number of leaves in the fabric:

a. Adding a leaf causes X more links in the topology (one per spine).
b. Adding a spine causes Y more links in the topology (one per leaf).

Thus, adding a node in either role has a linear increase in graphical complexity and the state required to maintain the graph.

With respect to BTS-injected prefixes, the state is more difficult to assess since the control-plane is reactive as congestion is detected. A fabric with no congestion has no BGP state, which scales in constant time/space. A fabric with significant congestion will have significant BGP state, which scales as poorly as quadratic when computing the product of leaves and prefixes within the fabric. The most realistic general case is in the middle where each leaf has a few BGP routes for a few specific destinations. The amount of state is also directly proportional to the oversubscription ratio. More oversubscription increases the likelihood of more congestion, which in turns causes more exception routing via BGP.

4.2. Optimization

Unlike state and surface, optimization has a positive connotation and is often the target of any design. Optimization is a general term that represents the process of meeting a set of design goals to the maximum extent possible; certain designs will be optimized against certain criteria. Common optimization designs will revolve around minimizing cost, minimizing convergence time, minimizing network overhead, maximizing utilization, maximizing manageability, maximizing user experience, etc.

As state increases in the network to avoid congestion in the fabric, optimization is directly proportional and likewise increases. A network with no BGP state could, in a technical sense, be called "un-optimized" since no specific path information is used. This "un-optimized" network is also the ideal state as a fabric without congestion does not require exception routing. The direct correlation between state and optimization in this architecture represents a clear trade-off. The sensitivity of the BTS system with respect to injecting prefixes should be set to an appropriate threshold based on the needs of the network administrator.

4.3. Surface

Surface defines how tightly intertwined components of a network interact. Surface is a two-dimensional attribute that measures both breadth and depth of interactions between said components. The breadth of interaction is typically measured by the number of places in the network some interaction occurs, whereas the depth of interaction helps describe how closely coupled two things operate.

Surface interactions are kept minimally deep in this network. The spines have almost no surface interactions between protocols at all since they only run IGP. The leaves rely on BTS-injected BGP prefixes for TE, which use IGP next-hops. This surface interaction is wide as it exists on all leaves in the topology, but very shallow since this BGP/IGP interaction is a natural function of route recursion. Notwithstanding BGP's assessment of the IGP cost in the best-path selection algorithm, no other leaky abstractions exist between the two. Other complex surface interactions such as redistribution are likewise absent.

Routing on the spines is entirely deterministic as they have exactly one path to each leaf. Since this logic cannot be modified without violating the fundamental logic of a leaf/spine fabric, BGP is not required on any spine router, so BTS-injected routes are local to a leaf's BGP process only.

5. Other Considerations

5.1. IP Multicast

Since flows are being manually manipulated in the network, Protocol Independent Multicast (PIM) reverse path forwarding (RPF) paths can be affected. If BTS injects a longer match which covers a source or rendezvous point (RP), the corresponding (S,G) or (*,G) joins will be re-issued out of the proper RPF interfaces as needed. This happens as quickly as IGP can converge, so there are no special considerations for supporting multicast with this design provided the network remains loop-free.

5.2. IPv6

IPv6 has no special considerations other than feature availability. Some platforms or vendors may not yet support IP LFA for OSPFv3 or IS-IS IPv6 prefixes, while others do. The same is true for the BGP bandwidth extended community. As such, IPv4 is demonstrated here since the entire feature set requirement for this architecture is supported on a wider variety of platforms. However, there is nothing IPv4-specific about this architecture.

5.3. Routing Security

As discussed earlier, all leaves are configured to only receive BGP routes but never advertise them. The BTS systems should alternatively be configured to only advertise routes and never receive them. This ensures that the routing exchange occurs in the correct direction. The fabric leaves can be configured for a maximum prefix limit to protect against memory exhaustion attacks should the BTS systems be compromised.

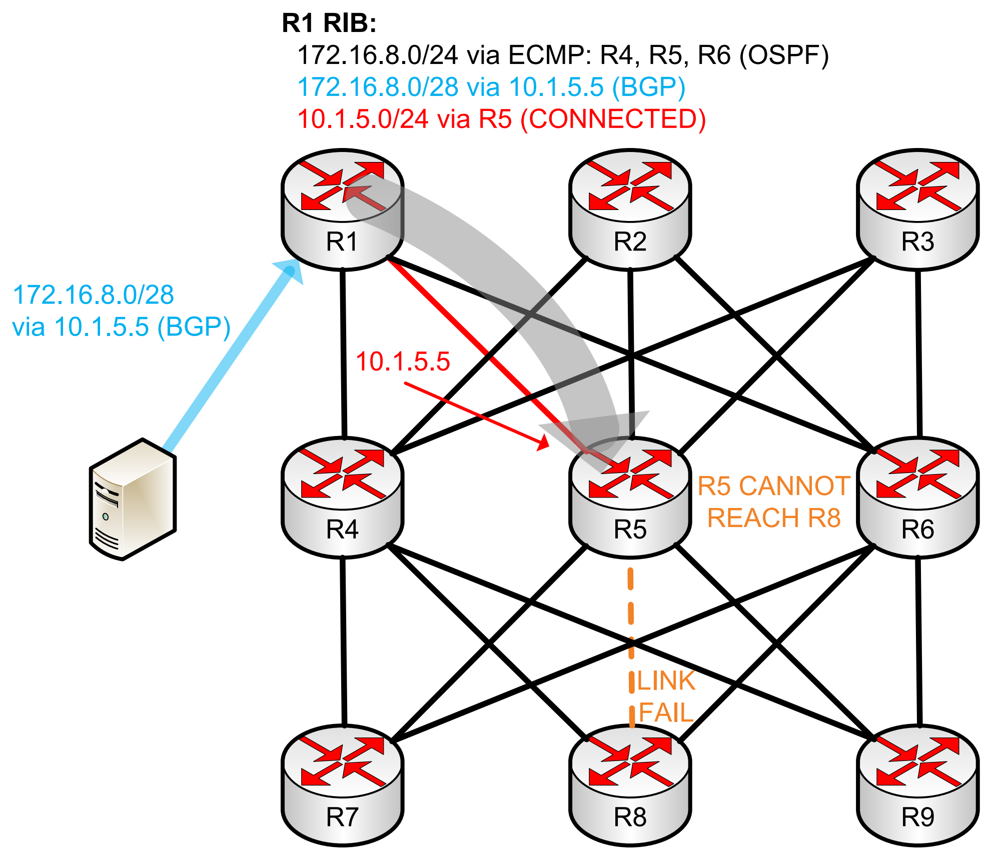
Any BTS-injected routes should only use next-hop IPs that are covered by remote transit links between leaves and spines in the network. This ensures that maliciously adjusted (or misconfigured) BGP next-hops advertised outbound from the BTS will have minimal impact. For example, failing to adjust the next-hop at all would direct traffic towards the BTS itself, which would black-hole traffic and likely crash the BTS. Setting the BGP next-hop to an incorrect remote transit link, in the worst case, directs traffic towards the wrong spine. The spine will always route the packet correctly since it has exactly one path to each leaf, so the destructiveness of such an attack/misconfiguration is relatively low.

There are two exceptions to this rule:

a. If a BGP speaker learns a BGP prefix with a next-hop set to one if its local interfaces, the route is discarded. This is desirable as the fabric ECMP paths will be used instead.
b. If a BGP-speaking leaf learns a BGP prefix with a next-hop set to one of the directly-connected spine interfaces, forwarding can still work, but new issues arise. In this case,

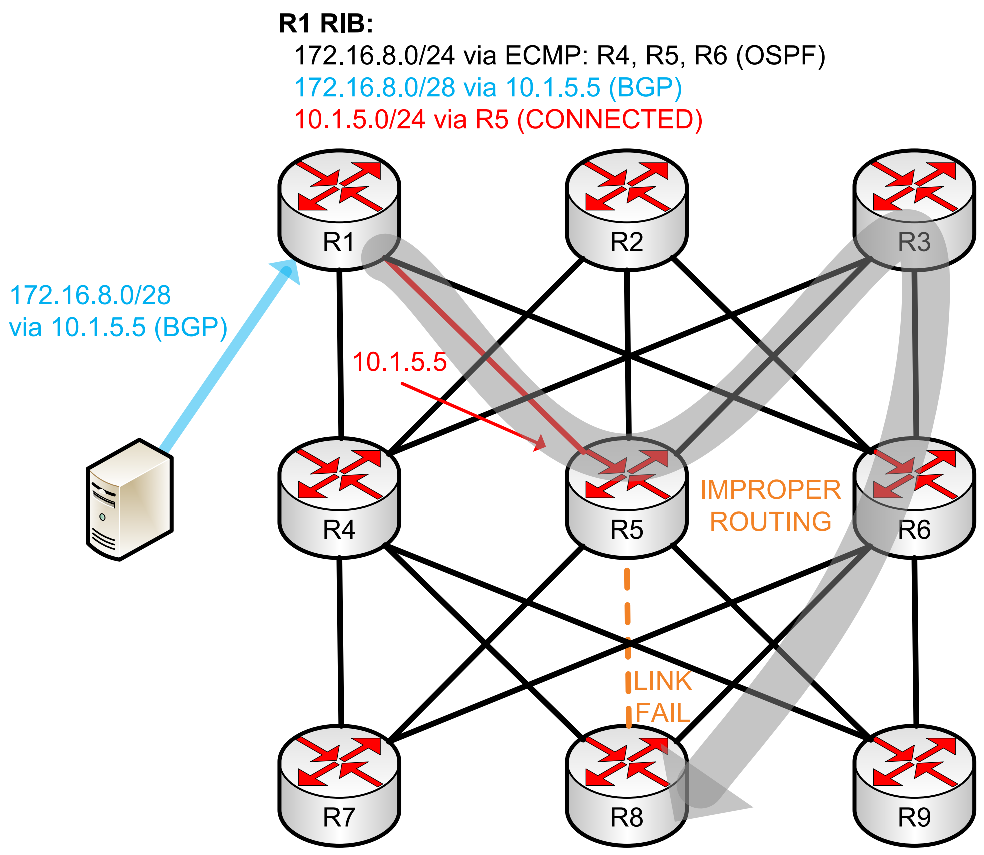
BGP is selecting only the connected outgoing interface (i.e., spine selection only) yet there is no visibility into the remote spine-to-leaf link. The diagrams below illustrate the potential black-hole or suboptimal routing issues (depending on RIB entry criteria) that may result. If the spines are not permitted to route to leaves through other leaves, which can be achieved using the EIGRP stub or IS-IS overload-bit features, a black-hole results.

Figure 21 - Routing Black-Hole when using Connected Next-Hops


R1 RIB:
172.16.8.0/24 via ECMP: R4, R5, R6 (OSPF)
172.16.8.0/28 via 10.1.5.5 (BGP)
10.1.5.0/24 via R5 (CONNECTED)
172.16.8.0/28
via 10.1.5.5 (BGP)
10.1.5.5
R5 CANNOT
REACH R8
LINK
FAIL
R1
R2
R3
R4
R5
R6
R7
R8
R9

If spine-to-spine routing is permitted, a massively suboptimal forwarding path is formed. The originally selected spine must route through some other leaf to reach the final destination, causing a second spine to be selected. To prevent both the blackhole and suboptimal routing cases, only non-connected next-hop prefixes are considered valid BGP next-hops.

Figure 22 - Suboptimal Routing when using Connected Next-Hops


R1 RIB:
172.16.8.0/24 via ECMP: R4, R5, R6 (OSPF)
172.16.8.0/28 via 10.1.5.5 (BGP)
10.1.5.0/24 via R5 (CONNECTED)
172.16.8.0/28
via 10.1.5.5 (BGP)
10.1.5.5
R1
R2
R3
R4
R5
R6
IMPROPER
ROUTING
LINK
FAIL
R7
R8
R9

Internal BGP route-reflection should not be used anywhere in the network as it is entirely unnecessary. For additional protection, BTS-injected routes can use the “no-advertise” community to ensure a leaf uses the TE path only for itself. This is more relevant in eBGP designs because iBGP will automatically prevent further advertisement of iBGP-learned routes. The BTS should pass both standard and extended communities to all leaves, but since the leaves need not advertise any BGP prefixes to the BTS, the opposite is not true.

Note that a total failure of the BGP control-plane means the fabric falls back to ECMP-based forwarding indefinitely, which is the best possible fail-back mechanism. Congestion points cannot be effectively mitigated during this failure scenario, but the fabric performs as expected in every other aspect. This is the ideal state of the network since BTS was specifically introduced to alleviate congestion; a fabric without congestion is operating optimally.

5.4. Standardized Solutions/Protocols

No solutions or techniques in this design are proprietary to any vendor. Cisco originally developed the BGP bandwidth extended community in a draft which has since expired, but other vendors have implemented it. Even without this extended community, ECMP or strict BGP failover paths can be used on leaves when multiple BGP prefixes are injected from a BTS.

5.5. Topology Limitations

This technique can only work in fabrics that have exactly one tier of spines between any pair of leaves. Additional levels of hierarchy within leaf/spine networks are certainly desirable for larger-scale fabrics, but this technique does not work in such topologies without extending BGP into the spine. At that point, the problem becomes better solved with a more advanced centralization technique versus the simple BGP method described in this document. As such, this architecture is best suited for small to medium DCs using a 3-stage fabric of spines and leaves.

Appendix A – Acronyms

Acronym	Definition
API	Application Programming Interface
AS	Autonomous System
BGP	Border Gateway Protocol
BTS	BGP Traffic-Engineering Server
CLI	Command Line Interface
DC	Data Center
DMZ	DeMilitarized Zone
eBGP	External BGP
ECMP	Equal Cost Multi Path
EIGRP	Enhanced Interior Gateway Routing Protocol
FD	Feasible Distance (EIGRP)
FIB	Forwarding Information Base
FRR	Fast ReRoute
gRPC	Google Remote Procedure Call
HTTP	HyperText Transport Protocol
iBGP	Internal BGP
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IS-IS	Intermediate-System to Intermediate-System
JSON	JavaScript Object Notation

Acronym	Definition
LAN	Local Area Network
LFA	Loop-Free Alternate
MPLS	Multi-Protocol Label Switching
NHT	Next Hop Tracking
OSPF	Open Shortest Path First
PIM	Protocol Independent Multicast
RD	Reported Distance (EIGRP)
REST	Representation State Transfer
RIB	Routing Information Base
RP	Rendezvous Point
RPF	Reverse Path Forwarding
SDN	Software Defined Networking
SOS	State/Optimization/Surface
TE	Traffic Engineering
UCMP	Unequal Cost Multi-Path
XML	eXtensible Markup Language
YANG	Yet Another Next Generation (data modeling language)

Appendix B – References

BGP Bandwidth Extended Community (IETF Draft v07)

BGP Version 4 (IETF RFC-4271)

Clos Fabric (Wikipedia)

FastAPI (HTTP API framework) Documentation

Flask (HTTP framework) Documentation

gRPC Documentation

IP LFA Selection (IETF RFC-5286)

Large Scale DC Routing using BGP (IETF RFC-7938)

Navigating Network Complexity (White and Tantsura)

NETCONF Protocol (IETF RFC-6241)

OSPF Version 2 (IETF RFC-2328)

RESTCONF Protocol (IETF RFC-8040)

YANG Data Modeling Language (IETF RFC-7950)