Scalable network infrastructure in Layer 3 with BGP

Growth Spurt

Large environments such as clouds pose demands on the network, some of which cannot be met with Layer 2 solutions. The Border Gateway Protocol jumps into the breach in Layer 3 and ensures seamlessly scalable networks. By Martin Loschwitz

Large-scale virtualization environments have ousted typical small setups. Whereas a company previously purchased a few physical servers to deploy an application, today, the entire workload of a new setup ends up on virtual machines running on a cloud service provider’s platform.

A physical layout often is based on a tree structure (Figure 1), with the admin connecting all the servers to one or two central switches and adding more switches if the number of ports on a switch is not sufficient. Together, the switches and network adapters form a large physical segment in OSI Layer 2.

In this article, I describe how you can build an almost arbitrarily scalable network for your environments with Layer 3 tools. As long as two hosts have any kind of physical communication path, communication on Layer 3 works, even if the hosts in question reside in different Layer 2 segments. The Border Gateway Protocol (BGP) makes this possible by providing a way to let each server know how to reach other servers; “IP fabric” describes data center interconnectivity via IP connections.

Virtually

New setups in virtual environments deliver much shorter time to market for the customer: Admins no longer need to order the hardware and suffer annoying waits for delivery, installation, and roll-outs. Many benefits also arise for cloud operators: Virtual environments such as public clouds are far more uniform than a variety of individual setups and can be managed

Figure 1: Classic network architectures that follow a tree structure are not suitable for scale-out platforms.
more efficiently. Also, horizontal scaling is easier because these platforms can be expanded almost at will. The changes also affect planning in the IT environment. Previously, IT designed a single setup, built it, and operated it until a new solution replaced the old one. In contrast, massively scalable environments are designed not only for the next five years, but well into the future.

Add the size factor: A cloud environment starts life as a basic setup and grows continuously as the corresponding user demand increases. When planning a public cloud, the planners do not know the target size and must be suitably cautious. If a company makes an error, the consequences that appear later in everyday business can be fatal, making the company put considerable effort into building workarounds to compensate for the flaw in the design of the solution.

Conventional wisdom says the earlier a design flaw is identified while planning a platform, the cheaper it is to remedy. According to a speech by Barry Boehm at EQUITY 2007 [1], the cost of working around design bugs after the requirements have been specified increases non-linearly as the project moves through design (5x), coding (10x), development testing (20x), acceptance testing (50x), and production (>150x): If the design bug is identified and removed in the design phase, the costs are manageable, but if the fault only becomes apparent when the platform is in production, the costs multiply [but see [2] for a dissenting opinion].

Toolbox

On the software side, admins can now access a toolkit to help build large environments. Clouds like OpenStack or container-based solutions such as Kubernetes are factory-built for scalability. Off-the-shelf hardware that is not directly designed for horizontal scaling out of the box is in many cases nevertheless integrated into a scale-out setup by the software: Ceph, for example, easily turns ordinary servers into a scalable object store that can provide a capacity of multiple petabytes.

Scaling, however, still has one major challenge: the network. Clouds like OpenStack make demands on both the logical network and the physical network on the hardware side that are virtually unsolvable with conventional network designs. Whereas software-defined networking (SDN) has long since asserted itself in several variants for logical networks, the physical level can be a tight squeeze for several reasons.

Conventional Tree Structure

Typical network layouts do not work in massively scalable environments because, if an enterprise is planning the network for a classic standalone setup, the maximum target size is known and usually limited to a certain number of servers. If more ports are required, the switch cascade continues on the underlying switch levels, illustrating the disadvantages of the tree structure. On the one hand, the admin is confronted sooner or later with the Spanning Tree Protocol (STP) – long-suffering networkers can tell many a tale of this – and on the other hand, only a fraction of the performance that the main switch could provide actually reaches the final members of such a cascade.

In massively scalable environments, the central premise on which the tree approach described here is founded falls away – the target scale-out is completely unknown. A new customer might want to launch 600 virtual machines on the fly. Depending on the configuration, for the provider, this means they need to add dozens of servers to the racks virtually overnight because the customer will otherwise lease from Amazon, Microsoft, or Google.

At least the total number of required ports is a known value on which planners can base their calculations for the setup. If dozens of servers suddenly find their way into the data center, the network infrastructure needs to grow at the same rate, which cannot be done with tree-like setups and switch cascades.

Admins come under attack from another corner: It is by no means certain three years after the original setup that you will still be able to buy the same network hardware on which you initially relied. Even with devices by the same manufacturer, later models are not guaranteed to be compatible with their predecessors. Detailed tests are therefore needed in such scale-out cases, as they are in cases where admins are looking to replace legacy network hardware with newer, more powerful components.

If you need to install devices from other manufacturers for a later scale-out, you risk a total meltdown: Although all the relevant network protocols are standardized, if you have ever tried to combine devices from different manufacturers, you are well aware that the interesting thing about standards is that there are so many of them.

What Next?

The clinch for the admin is how to design the network so that it will still scale horizontally in 10 or more years but without becoming unmanageable. In your mind’s eye, you almost automatically picture not being able to see the wood for the trees. Building giant computer networks is a solved problem; the necessary technology has existed for decades. After all, the Internet is just a huge contiguous network that is divided into a number of physical segments. What could be so obvious as applying the available technology to the local network?

Layer 3 Basics

The differences between Layer 2 networks and Layer 3 networks – in terms of planning and implementation – are huge. First, you need to ditch a central assumption that is essential for Layer 2 networks: that each host is a member of the same network segment. The direct connection between two hosts is handled in the Layer 2 world with the help
of the Address Resolution Protocol (ARP). A network design based on the Layer 3, however, no longer relies on all servers residing on the same physical network segment. Routing is a core function of Layer 3.

The Layer 3 setup is based in part on technical work by global corporations such as Google and Facebook, who implemented suitable concepts years ago using the Internet Protocol (IP) as the common fabric in the data center, thus engendering the name “IP fabric.” In a direct comparison with Layer 2 networks, the IP fabric principle means that additional components are required at various points of the setup. Why are they needed? A look under the hood helps to understand the details of IP fabrics; the need for additional tools in the setup is then almost automatic.

**Details**

A Layer 3-based network also uses Layer 2, but with different basic assumptions. A host within the scope of a network can basically talk to different targets: On the one hand, it can reach directly the servers that reside in the same network segment. Two servers on the same switch use ARP to find each other and then exchange the desired communication. If a host server wants to communicate outside its own network segment, it happens exclusively on Layer 3 of the OSI model: IPv4 is the most common example.

The communication source knows its own IP address as well as the address of another server on the same network, which forwards its packets to the target computer (gateway). IP fabrics, which work on the basis of OSI Layer 3, take advantage of exactly this property: The basic assumption is simply that every other host is only reachable via a gateway. Traffic for other servers in such a setup is thus always via a gateway.

For this principle to work, each server needs to know how to reach the other servers in the setup, which is where BGP comes into play. Using the BGP protocol, each host that is part of the IP fabric announces the routes over which it can be reached. The switches also play an important role: They form the BGP counterpart to the individual servers and distribute the routes they have learned across the entire setup. Any host in the setup can thus communicate with any other host via OSI Layer 3 using the switches as gateways. Packets that are directed to servers outside of the setup, such as Internet traffic, are routed via external gateways on routes learned from BGP.

Three components are thus necessary for an IP fabric: (1) an external gateway to communicate with the outside world, (2) a smart switch, and (3) a similarly smart node that uses BGP to announce its routing information. Because BGP is a standardized protocol, such a setup can be created in many ways with various components. The following example is just one implementation option – concrete hardware-based examples will follow later.

**Configuring the Hosts**

What is crucial for IP fabrics is that each host in the setup speaks BGP; that is in practical terms, each host becomes a router itself. BGP is based on the idea of routing announcements: A host uses such an announcement in the BGP protocol to specify which target networks it lets you reach.

For this to work, at least one local area network and one local IP address must be defined for each participating host that can be reached on the respective local network – this is the IP address that other hosts will use later to communicate with the server in question. In technical jargon, these networks, which mostly comprise four IP addresses (subnet mask /30) taken from the local IPv4 address space, are typically known as transfer networks.

The target IP – strictly speaking also a separate network with an IP address, and thus a subnet mask /32 – can reside on any network interface of the server, including the loopback interface lo. The server uses BGP to announce that the network with the target IP can be reached via the described transfer network. The admin has the choice between two BGP solutions that have asserted themselves on Linux: the veteran Quagga [3] or the more lightweight Bird [4] (Figure 2). Which of the two solutions you choose is ultimately a matter of personal taste. Both Quagga and Bird can be easily rolled out with today’s automation solutions.

The architecture of the two services differs considerably: Quagga comprises several components and an interactive BGP shell, which you can use to send commands directly to the service. Bird also offers a command-line tool for querying an active instance of the service (birdc); all told, it is significantly less complex than

![Figure 2: Bird supports BGP and is a lightweight alternative to Quagga.](image-url)
Quagga. Both services are suitable for IP fabrics because only a minimal set of BGP commands is used. If you have not worked with either of the two services before, you will probably find it easier to get started with Bird. Redundancy plays an important role in such a setup, and it is easy to implement with BGP. After all, the number of transfer networks per host is not limited, and it makes sense to define one for each physical network port. The server then uses BGP to announce a number of paths to the destination IP on the network, matching the number of ports you configured. This arrangement is smart and efficient and allows you to use network interface cards (NICs) at the same time without any limitations for as long as they work. If a NIC fails, the BGP announcement of the respective path is dropped, leaving just the other working path.

**Switch Helpers**

As described, BGP is based on the principle that routers use the protocol to exchange routing information about the hosts on the network. For the setup to work in practice, the switch needs to be actively involved: It must speak BGP and act as a peer for the hosts in the scope of the BGP protocol. The switch plays two central roles: On the one hand, it collects the incoming BGP announcements from the servers connected to it and maintains a central routing table that it provides to all other routers on the network – both hosts and other switches. On the other hand, the switch acts as a physical router: The individual hosts send packets to the address that they learned from the BGP and which is more or less the default gateway in this case. The second usable IP address on the transfer network is good for this purpose; the address is configured on the switch for the respective port. The switch changes the target MAC address for these packets on the basis of routing information, reduces their time-to-live (TTL) by 1, and forwards them to the target computer.

A special role takes on external traffic: in this case, traffic that is not addressed to one of the local transfer networks. The switches use BGP to pass this traffic directly to the gateways set up especially for this purpose and, hence, known as border gateways. In contrast to a classic tree architecture, an IP fabric can have any number of these gateways. The packets ultimately take paths based on the routing information from BGP and not on the basis of the existing Layer 2 network.

What applies to border gateways also applies to the number of switches in the setup: Additional switches can be connected to any existing switch at any time, and the switches then seamlessly integrate into the existing BGP setup. Although somewhat unorthodox, it is possible without assuming any disadvantages. The approach is typically different in practice. The following example therefore points to best practices and also cites concrete examples of usable hardware.

**Top of Rack, Core, Leaf, Spine**

The kind of IP fabric designs presented here use many alternative terms. Manufacturers like to talk about leaf-spine architectures, or top-of-rack (ToR) switches, and core switches. They almost always mean the same thing: Assuming good planning, IP fabrics let you achieve a far higher number of ports and genuine scalability. Commonly, each rack is assigned a separate switch, with the switches often mounted at the top of the rack (hence the name “top of rack”). Cross-cabling of multiple racks can make sense. For example, if you have two racks side by side, each with a Mellanox SN2410 switch (48x25Gbps Ethernet), you could first connect the servers in the racks with the ToR switch in the same rack and then with the ToR switch in the rack next door to achieve redundancy. If the 48 ports are not enough, you can install several of these devices.

You thus need more powerful hardware for the switch layer: You might want to deploy the Mellanox SN2700, which offers 32 genuine 100Gbps Ethernet ports (**Figure 3**), which you can split up using break-out cables (see the box “Tricky Bit: The Switches”). By connecting the ToR switches from various racks to one or multiple core switches of this model, you can give the individual racks high-performance uplinks. After all, it is the core switches that are connected directly to the core routers facing the Internet and thus enable external connectivity. A setup planned in this way achieves maximum flexibility: Additional racks can always be connected to the existing core switches, and if you run out of ports, you can add an additional core switch. If you want, you can even add a third layer of switches during operation, which, especially in light of the imminent availability of 200Gbps Ethernet switches, could be a worthwhile alternative. Undoubtedly, as long as some physical connection is available from a new switch down to the existing switch.

![Figure 3: The Mellanox SN2700 offers 32 genuine 100Gbps Ethernet ports and is suitable as a core switch for connecting multiple top-of-rack switches.](image-url)
network, new switches can be added with no worries. Even without downtime, if you briefly turn off a switch to replace it with a more powerful model, the other BGP routes remain unaffected, and the network continues to function normally. You can even add additional layers of switches during operation, without worrying about a maintenance window.

### Existing Setups

The network design described here is also interesting for existing setups that suffer from the disadvantages of the tree layout. The BGP-based setup can be quite easily introduced retroactively. To do so, you do not need to do much more than install additional network cards in the existing servers, which you then configure via BGP.

You can also use BGP to connect separate networks that were previously unable to communicate for historical reasons. In such setups, you will very likely find isolated networks that prevent changes to the central infrastructure of the core routers. An IP fabric solution provides a reliable solution in this case.

---

**Tricky Bit: The Switches**

It is no coincidence that I mentioned Spectrum series switches by Mellanox twice. They offer a feature that is especially relevant in the context of IP fabrics: You can run Cumulus Linux on them, a special version of Debian GNU/Linux that has been specifically customized for operation on network devices. With all the neat benefits this setup offers, if you implement BGP on the hosts using Quagga or Bird, you can do the same on your Cumulus switches (Figure 4). Quagga is included and Bird can be installed with apt-get, although the manufacturer does not provide any support for this component. Thanks to Debian, the switches thus integrate seamlessly into an existing automation solution. Like the other hosts in the setup, the switch also gets its configuration via Ansible. Of course, you can set up an IP fabric like the one I described with third-party network hardware and without setting up Cumulus Linux. The industry has finally taken note of the desire to use switches by the two kings of the hill, Cisco and Juniper, as routers. Both operating systems, NX-OS and Junos OS, respectively, now offer corresponding features, but an expensive additional license is typically required to do so. Additionally, NX-OS and Junos OS will not integrate so easily with an existing automation environment. If you have the choice, you should at least evaluate Cumulus.

Another thing in favor of Cumulus: If the switches on the network are only simple servers with Linux, any good system administrator can manage them, which is not true of Cisco or Juniper, for which you need a specialist. In some ways, IP fabrics thus also drive the need for IT specialists to change and become all-rounders. Sys admins without previous knowledge of networks can look forward to a steep learning curve because BGP is fairly complex, but generally, if you can use BGP on Linux, you will get by with Cumulus, Bird, or Quagga.

![Figure 4: Cumulus on switches offers many advantages: You can resort to a real Linux distribution in which each port is a separate interface.](image-url)