4.1 Overview

Datacenters are the core of cloud computing, and their network is an essential component to allow distributed applications to run efficiently and predictably [17]. However, not all datacenters provide cloud computing. In fact, there are two main types of datacenters: production and cloud. Production datacenters are often shared by one tenant or among multiple (possibly competing) groups, services and applications, but with low rate of arrival and departure. They run data-analytics jobs with relatively little variation in demands, and their size varies from hundreds of servers to tens of thousands of servers. Cloud datacenters, in contrast, have high rate of tenant arrival and departure (churn) [75], run both user-facing applications and inward computation, require elasticity (since application demands are highly variable), and consist of tens to hundreds of thousands of physical servers [12]. Moreover, clouds can be composed of several datacenters spread around the world. As an example, Google, Microsoft and Amazon (three of the biggest players in the market) have datacenters in four continents; and each company has over 900,000 servers.

This chapter presents a in-depth study of datacenter networks (DCNs), relevant standards and operation. Our goal here is three-fold: i) provide a detailed view of the net-
working infrastructure connecting the set of servers of the datacenter via high-speed links and commodity off-the-shelf (COTS) switches [14]; ii) discuss the addressing and routing mechanisms employed in this kind of network; and iii) show how the nature of traffic may impact DCNs and affect design decisions.

Providers typically have three main goals when designing a DCN [36]: scalability, fault tolerance, and agility. First, the infrastructure must scale to a large number of servers (and preferably allow incremental expansion with commodity equipment and little effort). Second, a DCN should be fault tolerant against failures of both computing and network resources. Third, a DCN ideally needs to be agile enough to assign any VM (which is part of a service or application) to any server [35]. As a matter of fact, DCNs should ensure that computations are not bottlenecked on communication [77].

Currently, providers attempt to meet these goals by implementing the network as a multi-rooted tree [17], using LAN technology for VM addressing and two main strategies for routing: Equal-Cost MultiPath (ECMP) and Valiant Load Balancing (VLB). The shared nature of DCNs among a myriad of applications and tenants and high scalability requirements, however, introduce several challenges for architecture design, protocols and strategies employed inside the network. Furthermore, the type of traffic in DCNs is significantly different from traditional networks [39]. Therefore, we also survey recent proposals in the literature to address the limitations of technologies used in today’s DCNs.

We structure this chapter as follows. First, we begin by examining the typical multi-rooted tree topology used in current datacenters and discuss its benefits and drawbacks. Then, we take a look at novel topologies proposed in the literature, and how network expansion can be performed in a cost-efficient way for providers. After addressing the structure of the network, we look into the traffic characteristics of these high-performance, dynamic networks and discuss proposals for traffic management on top of existing topologies. Based on the aspects discussed so far, we present layer-2 and layer-3 routing, its requirements and strategies typically employed to perform such task. We also examine existing mechanisms used for VM addressing in the cloud platform and novel proposals to increase flexibility and isolation for tenants. Finally, we discuss the most relevant open research challenges and close this chapter with a brief summary of DCNs.

4.2 Topologies

In this section, we present an overview of datacenter topologies. The topology describes how devices (routers, switches and servers) are interconnected. More formally, this is represented as a graph, in which switches, routers and servers are the nodes, and links are the edges.

**Typical topology.** Figure 4.1 shows a canonical three-tiered multi-rooted tree-like physical topology, which is implemented in current datacenters [17, 20]. The three tiers are: i) the access (edge) layer, composed of the Top-of-Rack (ToR) switches that connect servers mounted on every rack; ii) the aggregation (distribution) layer, consisting of devices that interconnect ToR switches in the access layer; and iii) the core layer, formed by routers that interconnect switches in the aggregation layer. Furthermore, every ToR switch may be connected to multiple aggregation switches for redundancy (usually 1+1 redundancy) and every aggregation switch is connected to multiple core switches. Typically, a three-tiered network is implemented in datacenters with more than 8,000 servers [14]. In smaller datacenters, the core and aggregation layers are collapsed into one tier, resulting in a two-tiered datacenter topology (flat layer 2 topology) [20].
This multitiered topology has a significant amount of oversubscription, where servers attached to ToR switches have significantly more (possibly an order of magnitude) provisioned bandwidth between one another than they do with hosts in other racks [12]. Providers employ this technique in order to reduce costs and improve resource utilization, which are key properties to help them achieve economies of scale.

This topology, however, presents some drawbacks. First, the limited bisection bandwidth\(^1\) constrains server-to-server capacity, and resources eventually get fragmented (limiting agility) [28, 29]. Second, multiple paths are poorly exploited (for instance, only a single path is used within a layer-2 domain by spanning tree protocol), which may potentially cause congestion on some links even though other paths exist in the network and have available capacity. Third, the rigid structure hinders incremental expansion [76]. Fourth, the topology is inherently failure-prone due to the use of many links, switches and servers [53]. To address these limitations, novel network architectures have been recently proposed; they can be organized in three classes [62]: switch-oriented, hybrid switch/server and server only topologies.

**Switch-oriented topologies.** These proposals use commodity switches to perform routing functions, and follow a Clos-based design or leverage runtime reconfigurable optical devices. A Clos network [25] consists of multiple layers of switches; each switch in a layer is connected to all switches in the previous and next layers, which provides path diversity and graceful bandwidth degradation in case of failures. Two proposals follow the Clos design: VL2 [35] and Fat-Tree [14]. VL2, shown in Figure 4.2(a), is an architecture for large-scale datacenters and provides multiple uniform paths between servers and full bisection bandwidth (i.e., it is non-oversubscribed). Fat-Tree, in turn, is a folded Clos topology. The topology, shown in Figure 4.2(b), is organized in a non-oversubscribed k-ary tree-like structure, consisting of k-port switches. There are k two-layer pods with k/2 switches. Each k/2 switch in the lower layer is connected to k/2 servers and the remaining ports are connected to k/2 aggregation switches. Each of the (k/2)\(^2\) k-port core switches has one port connected to each of k pods. In general, a fat-tree built with k-port switches

---

\(^1\)The bisection bandwidth of the network is the worst-case segmentation (i.e., with minimum bandwidth) of the network in two equally-sized partitions [32].
supports \(k^{3/4}\) hosts. Despite the high capacity offered (agility is guaranteed), these architectures increase wiring costs (because of the number of links).

![Layer 1](image1)

![Layer 2](image2)

![Layer 3](image3)

(a) VL2.

![Layer 1](image4)

![Layer 2](image5)

![Layer 3](image6)

(b) Fat-Tree.

Figure 4.2 Clos-based topologies.

Optical Switching Architecture (OSA) [23], in turn, uses runtime reconfigurable optical devices to dynamically change physical topology and one-hop link capacities (within 10 ms). It employs hop-by-hop stitching of multiple optical links to provide all-to-all connectivity for the highly dynamic and variable network demands of cloud applications. This method is shown in the example of Figure 4.3. Suppose that demands change from the left table to the right table in the figure (with a new highlighted entry). The topology must be adapted to the new traffic pattern, otherwise there will be at least one congested link. One possible approach is to increase capacity of link F-G (by reducing capacity of links F-D and G-C), so congestion can be avoided. Despite the flexibility achieved, OSA suffers from scalability issues, since it is designed to connect only a few thousands of servers in a container, and latency-sensitive flows may be affected by link reconfiguration delays.

![Figure 4.3](image7)

Figure 4.3 OSA adapts according to demands (adapted from [23]).

**Hybrid switch/server topologies.** These architectures shift complexity from network devices to servers, i.e., servers perform routing, while low-end mini-switches interconnect a fixed number of hosts. They also can provide higher fault-tolerance, richer connectivity and improve innovation, because hosts are easier to customize than commodity switches. Two example topologies are DCell [36] and BCube [37], which can arguably scale up to millions of servers.

DCell [36] is a recursively built structure that forms a fully-connected graph using only commodity switches (as opposed to high-end switches of traditional DCNs). DCell aims to scale out to millions of servers with few recursion levels (it can hold 3.2 million servers with only 4 levels and 6 hosts per cell). A DCell network is built as follows. A level-0 DCell (DCell\(_0\)) is composed of servers connected to a n-port commodity switch. DCell\(_1\)
is formed with $n+1$ DCell; each DCell is connected to all other DCell with one bidirectional link. In general, a level-$k$ DCell is constructed with $n+1$ DCell in the same manner as DCell. Figure 4.4(a) shows an example of a two-level DCell topology. In this example, a commodity switch is connected with 4 servers ($n=4$) and, therefore, a DCell is constructed with 5 DCell. The set of DCell is interconnected in the following way: each server is represented by the tuple $(a_1,a_0)$, where $a_1$ and $a_0$ are level 1 and 0 identifiers, respectively; and a link is created between servers identified by the tuples $(i,j-1)$ and $(j,i)$, for every $i$ and every $j > i$.

Similarly to DCell, BCube is a recursively built structure that is easy to design and upgrade. Additionally, BCube provides low latency and graceful degradation of bandwidth upon link and switch failure. In this structure, clusters (a set of servers interconnected by a switch) are interconnected by commodity switches in a hypercube-based topology. More specifically, BCube is constructed as follows: BCube (level-0 BCube) consists of $n$ servers connected by a $n$-port switch; BCube is constructed from $n$ BCube and $n$ n-port switches; and BCube is constructed from $n$ BCube and $n$ n-port switches. Each server is represented by the tuple $(x_1,x_2)$, where $x_1$ is the cluster number and $x_2$ is the server number inside the cluster. Each switch, in turn, is represented by a tuple $(y_1,y_2)$, where $y_1$ is the level number and $y_2$ is the switch number inside the level. Links are created by connecting the level-$k$ port of the $i$-th server in the $j$-th BCube to the $j$-th port of the $i$-th level-$k$ switch. An example of two-level BCube with $n=4$ (4-port switches) is shown in Figure 4.4(b).

Despite the benefits, DCell and BCube require a high number of NIC ports at end-hosts – causing some overhead at servers – and increase wiring costs. In particular, DCell results in non-uniform multiple paths between hosts, and level-0 links are typically more utilized than other links (creating bottlenecks). BCube, in turn, provides uniform multiple paths, but uses more switches and links than DCell.

Server only topology. In this kind of topology, the network is composed of only servers that perform all network functions. An example of architecture is CamCube, which is inspired in Content Addressable Network (CAN) overlays and uses a 3D torus (k-ary 3-cube) topology with $k$ servers along each axis. Each server is connected directly to 6 other servers, and the edge servers are wrapped. Figure 4.5 shows a 3-ary CamCube topology, resulting in 27 servers. The three most positive aspects of CamCube are: i) providing robust fault-tolerance guarantees (unlikely to partition even with 50% of server
or link failures; ii) improving innovation with key-based server-to-server routing (content is hashed to a location in space defined by a server); and iii) allowing each application to define specific routing techniques. However, it does not hide topology from applications, has higher network diameter $O(\sqrt{N})$ (increasing latency and traffic in the network) and hinders network expansion.

![Figure 4.5 Example of 3-ary CamCube topology (adapted from [27]).](image)

Table 4.1 summarizes the benefits and limitations of these topologies by taking four properties into account: scalability, resiliency, agility and cost. The typical DCN topology has limited scalability (even though it can support hundreds of thousands of servers), as COTS switches have restricted memory size and need to maintain an entry in their Forwarding Information Base (FIB) for each VM. Furthermore, it presents low resiliency, since it provides only 1+1 redundancy, and its oversubscribed nature hinders agility. Despite the drawbacks, it can be implemented with only commodity switches, resulting in lower costs.

Table 4.1 Comparison among datacenter network topologies.

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scalability</td>
</tr>
<tr>
<td>Typical DCN</td>
<td>Low</td>
</tr>
<tr>
<td>Fat-Tree</td>
<td>High</td>
</tr>
<tr>
<td>VL2</td>
<td>High</td>
</tr>
<tr>
<td>OSA</td>
<td>Low</td>
</tr>
<tr>
<td>DCell</td>
<td>Huge</td>
</tr>
<tr>
<td>BCube</td>
<td>Huge</td>
</tr>
<tr>
<td>CamCube</td>
<td>Low</td>
</tr>
</tbody>
</table>
Fat-Tree and VL2 are both instances of a Clos topology with high scalability and full bisection bandwidth (guaranteed agility). Fat-Tree achieves average resiliency, as ToR switches are connected only to a subset of aggregation devices and has average overall costs (mostly because of increased wiring). VL2 scales through packet encapsulation, maintaining forwarding state only for switches in the network, achieves high resiliency by providing multiple shortest paths and by relying on a distributed lookup entity for handling address queries. As a downside, its deployment has increased costs (due to wiring, significant amount of exclusive resources for running the lookup system and the need of switch support for IP-in-IP encapsulation).

OSA was designed taking flexibility into account in order to improve resiliency (i.e., by using runtime reconfigurable optical devices to dynamically change physical topology and one-hop link capacities). However, it has low scalability (up to a few thousands of servers), no agility (as dynamically changing link capacities may result in congested links) and higher costs (devices should support optical reconfiguration).

DCell and BCube aim at scaling to millions of servers while ensuring high resiliency (rich connectivities between end-hosts). In contrast to BCube, DCell does not provide agility, as the set set of non-uniform multiple paths may be bottlenecked by links at level-0. Finally, their deployment costs may be significant, since they require a lot of wiring and more powerful servers in order to efficiently perform routing.

CamCube, in turn, is unlikely to partition even with 50% of server or link failures, thus achieving high resiliency. Its drawback, however, is related to scalability and agility; both properties can be hindered because of high network diameter, which indicates that, on average, more resources are needed for communication between VMs hosted by different servers. CamCube also has average deployment costs, mainly due to wiring and the need of powerful servers (to perform network functions).

As we can see, there is no perfect topology, since each proposal focus on specific aspects. Ultimately, providers are cost-driven: they choose the topology with the lowest costs, even if it cannot achieve all properties desired for a datacenter network running heterogenous applications from many tenants.

4.3 Network expansion

A key challenge concerning datacenter networks is dealing with the harmful effects that their ever-growing demand causes on scalability and performance. Because current DCN topologies are restricted to 1+1 redundancy and suffer from oversubscription, they can become under-provisioned quite fast. The lack of available bandwidth, in turn, may cause resource fragmentation (since it limits VM placement) [28] and reduce server utilization (as computations often depend on the data received from the network) [75]. In consequence, the DCN can lose its ability to accommodate more tenants (or offer elasticity to the current ones); even worse, applications using the network may start performing poorly, as they often rely on strict network guarantees.

These fundamental shortcomings have stimulated the development of novel DCN architectures (seen in Section 4.2) that provide large amounts of (or full) bisection bandwidth for up to millions of servers. Despite achieving high bisection bandwidth, their deployment

2For example, user-facing applications, such as Web Services, require low-latency for communication with users, while inward computation (e.g., Map-Reduce) requires reliability and bisection bandwidth in the intra-cloud network.
is hindered by the assumption of homogeneous sets of switches (with the same number of ports). For example, consider a Fat-Tree topology, where the entire structure is defined by the number of ports in switches. These homogeneous switches limit the structure in two ways: full bisection bandwidth can only be achieved with specific numbers of servers (e.g., 8192 and 27648) and incremental upgrade may require replacing every switch in the network [76].

In fact, most physical data center designs are unique: hence expansions and upgrades must be custom-designed and network performance (including bisection bandwidth, end-to-end latency and reliability) must be maximized while minimizing provider costs [28, 29]. Furthermore, organizations need to be able to incrementally expand their networks to meet the growing demands of tenants [76]. These facts have motivated recent studies [28, 29, 76, 77] to develop techniques to expand current DCNs to boost bisection bandwidth and reliability with heterogeneous sets of devices (i.e., without replacing every router and switch in the network). They are discussed next.

**Legup** [29]. Focused on tree-like networks, Legup is a system that aims at maximizing network performance at the design of network upgrades and expansions. It utilizes a linear model that combines three metrics (agility, reliability and flexibility), while being subject to the cloud provider’s budget and physical constraints. In an attempt to reduce costs, the authors of Legup develop the *Theory of Heterogeneous Clos Networks* to allow modern and legacy equipment to coexist in the network. Figure 4.6 depicts an overview of the system. Legup assumes an existing set of racks and, therefore, only needs to determine aggregation and core levels of the network (more precisely, the set of devices, how they interconnect, and how they connect to ToR switches). It employs a branch and bound optimization algorithm to explore the solution space only for aggregation switches, as core switches in a heterogeneous Clos network are restricted by aggregation ones. Given a set of aggregation switches in each step of the algorithm, Legup performs three actions. First, it computes the minimum cost for mapping aggregation switches to racks. Second, it finds the minimum cost distribution of core switches to connect to the set of aggregation switches. Third, the candidate solution is bounded to check its optimality and feasibility (by verifying if any constraint is violated, including provider’s budget and physical restrictions).

**Rewire** [28]. Recent advances in routing protocols may allow DCNs to shift from a rigid tree to a generic structure [10, 46, 59, 82]. Based on this observation, Rewire is a
framework that performs DCN expansion on arbitrary topologies. It has the goal of maximizing network performance (i.e., finding maximum bisection bandwidth and minimum end-to-end latency), while it minimizes costs and satisfies user-defined constraints. In particular, Rewire adopts a different definition of latency: while other studies model it by the worst-case hop-count in the network, Rewire also considers the speed of links and the processing time at switches (because unoptimized switches can add an order of magnitude more processing delay). Rewire uses Simulated Annealing (SA) [47] to search through candidate solutions and implements an approximation algorithm to efficiently compute their bisection bandwidth. The simulated annealing, however, does not take the addition of switches into account; it only optimizes the network for a given set of switches. Moreover, the process assumes uniform queuing delays for all switch ports, which is necessary because Rewire does not possess knowledge of network load.

**Jellyfish** [76]. End-to-end throughput of a network is quantitatively proved to depend on two factors: i) the capacity of the network; and ii) the average path length (i.e., throughput is inversely proportional to the capacity consumed to deliver each byte). Furthermore, as noted above, rigid DCN structures hinder incremental expansion. Consequently, a degree-bounded\(^3\) random graph topology among ToR switches, called Jellyfish, is introduced, with the goal of providing high bandwidth and flexibility. It supports device heterogeneity, different degrees of oversubscription and easy incremental expansion (by naturally allowing the addition of heterogeneous devices). Figure 4.7 shows a comparison of Fat-Tree and Jellyfish with identical equipment and same diameter (i.e., 6). Each ring in the figure contains servers reachable within the number of hops in the labels. We see that Jellyfish can reach more servers in fewer hops, because some links are not useful from a path-length perspective in a Fat-Tree (e.g., links marked with “x”). Despite its benefits, Jellyfish’s random design brings up some challenges, such as routing and the physical layout. Routing, in particular, is a critical feature needed, because it allows the use of the topology’s high capacity. However, results show that the commonly used ECMP does not utilize the entire capacity of Jellyfish, and the authors propose the use of k-shortest paths and MultiPath TCP [82] to improve throughput and fairness.

\(^3\)Degree-bounded, in this context, means that the number of connections per node is limited by the number of ports in switches.
Random graph based topologies [77]. Singla et al. analyze the throughput achieved by random graphs for topologies with both homogeneous and heterogeneous switches, while taking optimization into account. They obtain the following results: random graphs achieve throughput close to the optimal upper-bound under uniform traffic patterns for homogeneous switches; and heterogeneous networks with distinct connectivity arrangements can provide nearly identical high throughput. Then, the acquired knowledge is used as a building block for designing large-scale random networks with heterogeneous switches. In particular, they utilize the VL2 deployed in Microsoft’s datacenters as a case-study, showing that its throughput can be significantly improved (up to 43%) by only rewiring the same devices.

4.4 Traffic

Proposals of topologies for datacenter networks presented in Sections 4.2 and 4.3 share a common goal: provide high bisection bandwidth for tenants and their applications. It is intuitive that a higher bisection bandwidth will benefit tenants, since the communication between VMs will be less prone to interference. Nonetheless, it is unclear how strong is the impact of the bisection bandwidth. This section addresses this question by surveying several recent measurement studies of DCNs. Then, it reviews proposals for dealing with related limitations. More specifically, it discusses traffic patterns – highlighting their properties and implications for both providers and tenants – and shows how literature is using such information to help designing and managing DCNs.

Traffic can be divided in two broad categories: north/south and east/west communication. North/south traffic (also known as extra-cloud) corresponds to the communication between a source and a destination host where one of the ends is located outside the cloud platform. In contrast, east/west traffic (also known as intra-cloud) is the communication in which both ends are located inside the cloud. These types of traffic usually depend on the kind and mix of applications: user-facing applications (e.g., web services) typically exchange data with users and, thus, generate north/south communication, while inward computation (such as MapReduce) requires coordination among its VMs, generating east/west communication. Studies [69] indicate that north/south and east-west traffic correspond to around 25% and 75% of traffic volume, respectively. They also point that both are increasing in absolute terms, but east/west is growing on a larger scale [69]. Towards understanding traffic characteristics and how it influences the proposal of novel mechanisms, we first discuss traffic properties defined by measurement studies in the literature [20, 21, 22, 45] and, then, examine traffic management and its most relevant proposals for large-scale cloud datacenters.

Properties. Traffic in the cloud network is characterized by flows; each flow is identified by sequences of packets from a source to a destination node (i.e., a flow is defined by a set packet header fields, such as source and destination addresses and ports and transport protocol). Typically, a bimodal flow classification scheme is employed, using elephant and mice classes. Elephant flows are composed of a large number of packets injected in the network over a short amount of time, are usually long-lived and exhibit bursty behavior. In comparison, mice flows have a small number of packets and are short-lived [12]. Several measurement studies [20, 21, 22, 45, 58] were conducted to characterize network traffic and its flows. We summarize their findings as follows:
Traffic asymmetry. Requests from users to cloud services are abundant, but small in most occasions. Cloud services, however, process these requests and typically send responses which are comparatively larger.

Nature of traffic. Network traffic is highly volatile and bursty, with links running close to their capacity at several times during a day. Traffic demands change quickly, with some transient spikes and other longer ones (possibly requiring more than half the full-duplex bisection bandwidth) [52]. Moreover, traffic is unpredictable at long time-scales (e.g., 100 seconds or more). However, it can be predictable on shorter timescales (at 1 or 2 seconds). Despite the predictability over small timescales, it is difficult for traditional schemes, such as statistical multiplexing, to make a reliable estimate of bandwidth demands for VMs [81].

General traffic location and exchange. Most traffic generated by servers (on average 80%) stays within racks. Server pairs from the same rack and from different racks exchange data with a probability of only 11% and 0.5%, respectively. Probabilities for intra- and extra-rack communication are as follows: servers either talk with fewer than 25% or to almost all servers of the same rack; and servers communicate with less than 10% or do not communicate with servers located outside its rack.

Intra- and inter-application communication. Most volume of traffic (55%) represents data exchange between different applications. However, the communication matrix between them is sparse; only 2% of application pairs exchange data, with the top 5% of pairs accounting for 99% of inter-application traffic volume. Consequently, communicating applications form several highly-connected components, with few applications connected to hundreds of other applications in star-like topologies. In comparison, intra-application communication represents 45% of the total traffic, with 18% of applications generating 99% of this traffic volume.

Flow size, duration and number. Mice flows represent around 99% of the total number of flows in the network. They usually have less than 10 kilobytes and last only a few hundreds of milliseconds. Elephant flows, in turn, represent only 1% of the number of flows, but account for more than half of the total traffic volume. They may have tens of megabytes and last for several seconds. With respect to flow duration, flows of up to 10 seconds represent 80% of flows, while flows of 200 seconds are less than 0.1% (and contribute to less than 20% of the total traffic volume). Further, flows of 25 seconds or less account for more than 50% of bytes. Finally, it has been estimated that a typical rack has around 10,000 active flows per second, which means that a network comprising 100,000 servers can have over 25,000,000 active flows.

Flow arrival patterns. Arrival patterns can be characterized by heavy-tailed distributions with a positive skew. They best fit a log-normal curve having ON and OFF periods (at both 15ms and 100ms granularities). In particular, inter arrival times at both servers and ToR switches have periodic modes spaced apart by approximately 15 milliseconds, and the tail of these distributions is long (servers may experience flows spaced apart by 10 seconds).

Link utilization. Utilization is, on average, low in all layers but the core; in fact, in the core, a subset of links (up to 25% of all core links) often experience high utilization. In general, link utilization varies according to temporal patterns (time of day, day of week and month of year), but variations can be an order of magnitude higher at core
links than at aggregation and access links. Due to these variations and the bursty nature of traffic, highly utilized links can happen quite often; 86% and 15% of links may experience congestion lasting at least 10 seconds and 100 seconds, respectively, while longer periods of congestion tend to be localized to a small set of links.

- **Hot-spots.** They are usually located at core links and can appear quite frequently, but the number of hot-spots never exceeds 25% of core links.

- **Packet losses.** Losses occur frequently even at underutilized links. Given the bursty nature of traffic, an underutilized network (e.g., with mean load of 10%) can experience lots of packet drops. Measurement studies found that packet losses occur usually at links with low average utilization (but with traffic bursts that go beyond 100% of link capacity); more specifically, such behavior happens at links of the aggregation layer and not at links of the access and core layers. Ideally, topologies with full bisection bandwidth (such as a Fat-Tree) should experience no loss, but the employed routing mechanisms cannot utilize the full capacity provided by the set of multiple paths and, consequently, there is some packet loss in such networks as well [21].

Traffic management. Other set of papers [33, 54, 72, 80] demonstrate that available bandwidth for VMs inside the datacenter can vary by a factor of five or more in the worst-case scenario. Such variability results in poor and unpredictable network performance and reduced overall application performance [17, 74, 86], since VMs usually depend on the data received from the network to execute the subsequent computation.

The lack of bandwidth guarantees is related to two main factors, as follows. First, the canonical cloud topology is typically oversubscribed, with more bandwidth available in leaf nodes than in the core. When periods of traffic bursts happen, the lack of bandwidth up the tree (i.e., at aggregation and core layers) results in contention and, therefore, packet discards at congested links (leading to subsequent retransmissions). Since the duration of the timeout period is typically one or two orders of magnitude more than the round-trip time, latency is increased, becoming a significant source of performance variability [12]. Second, TCP congestion control does not provide robust isolation among flows. Consequently, elephant flows can cause contention in congested links shared with mice flows, leading to discarded packets from the smaller flows [75].

Recent proposals address this issue either by employing proportional sharing or by providing bandwidth guarantees. Most of them use the hose model [31] for network virtualization and take advantage of rate-limiting at hypervisors [66], VM placement [44] or virtual network embedding [85] in order to increase their robustness.

Proportional sharing. Seawall [75] and NetShare [50] allocate bandwidth at flow-level based on weights assigned to entities (such as VMs or services running inside these VMs) that generate traffic in the network. While both assign weights based on administrator specified policies, NetShare also supports automatic weight assignment. Both schemes are work-conserving (i.e., available bandwidth can be used by any flow that needs more bandwidth), provide max-min fair sharing and achieve high utilization through statistical multiplexing. However, as bandwidth allocation is performed per flow, such methods may introduce substantial management overhead in large datacenter networks (with over 10,000 flows per rack per second [20]). FairCloud [63] takes a different approach and proposes three allocation policies to explore the trade-off among network proportionality, minimum guarantees and high utilization. Unlike Seawall and NetShare, FairCloud does not allocate bandwidth along congested links at flow-level, but in proportion to the number of VMs of
each tenant. Despite the benefits, FairCloud requires customized hardware in switches and is designed specifically for tree-like topologies.

**Bandwidth guarantees.** SecondNet [38], Gatekeeper [70], Oktopus [17], Proteus [83] and Hadrian [18] provide minimum bandwidth guarantees by isolating applications in virtual networks. In particular, SecondNet is a virtualization architecture that distributes the virtual-to-physical mapping, routing and bandwidth reservation state in server hypervisors. Gatekeeper configures each VM virtual NIC with both minimum and maximum bandwidth rates, which allow the network to be shared in a work-conserving manner. Oktopus maps tenants’ virtual network requests (with or without oversubscription) onto the physical infrastructure and enforces these mappings in hypervisors. Proteus is built based on the observation that allocating the peak bandwidth requirements for applications leads to underutilization of resources. Hence, it quantifies the temporal bandwidth demands of applications and allocates each one of them in a different virtual network. Hadrian extends previous schemes by also taking inter-tenant communication into account and allocating applications according to a hierarchical hose model (i.e., per VM minimum bandwidth for intra-application communication and per tenant minimum guarantees for inter-tenant traffic). In contrast, a group of related proposals attempt to provide some level bandwidth sharing among applications of distinct tenants [42, 56, 64]. The approach introduced by Marcon et al. [56] groups applications in virtual networks, taking mutually trusting relationships between tenants into account when allocating each application. It provides work-conserving network sharing, but assumes that trust relationships are determined in advance. ElasticSwitch [64] assumes there exists an allocation method in the cloud platform and focuses on providing minimum bandwidth guarantees with a work-conserving sharing mechanism (when there is spare capacity in the network). Nevertheless, it requires two extra management layers for defining the amount of bandwidth for each flow, which may add overhead. Finally, EyeQ [42] leverages high bisection bandwidth of DCNs to support minimum and maximum bandwidth rates for VMs. Therefore, it provides work-conserving sharing, but depends upon the core of the network to be congestion-free. None of these approaches can be readily deployed, as they demand modifications in hypervisor source code.

### 4.5 Routing

Datacenter networks often require specially tailored routing protocols, with different requirements from traditional enterprise networks. While the latter presents only a handful of paths between hosts and predictable communication patterns, DCNs require multiple paths to achieve horizontal scaling of hosts with unpredictable traffic matrices [14, 35]. In fact, datacenter topologies (such as the ones discussed in Section 4.2) typically present path diversity, in which multiple paths exist between servers (hosts) in the network. Furthermore, many cloud applications (ranging from web search to MapReduce) require substantial (possibly full bisection) bandwidth [15]. Thus, routing protocols must enable the network to deliver high bandwidth by using all possible paths in the structure. We organize the discussion according to the layer involved, starting with the network layer.

**Layer 3.** To take advantage of the multiple paths available between a source and its destination, providers usually employ two techniques: Equal-Cost MultiPath (ECMP) [41] and Valiant Load Balancing (VLB) [34, 35, 79]. Both strategies use distinct paths for different flows. ECMP attempts to load balance traffic in the network and utilize all paths which have the same cost (calculated by the routing protocol) by uniformly spreading
traffic among them using flow hashing. VLB randomly selects an intermediate router (occasionally, a L3 switch) to forward the incoming flow to its destination.

Recent studies in the literature [15, 38, 40, 43] propose other routing techniques for DCNs. As a matter of fact, the static flow-to-path mapping performed by ECMP does not take flow size and network utilization into account [65]. This may result in saturating commodity switch L3 buffers and degrading overall network performance [15]. Therefore, a system called Hedera is introduced to allow dynamic flow scheduling for general multi-rooted trees with extensive path diversity. Hedera is designed to maximize network utilization with low scheduling overhead of active flows. In general, the system performs the following steps: i) detects large flows at ToR switches; ii) estimates network demands of these large flows (with a novel algorithm that considers bandwidth consumption according to a max-min fair resource allocation); iii) invokes a placement algorithm to compute paths for them; and iv) installs the set of new paths on switches.

Hedera uses a central OpenFlow controller [57] with a global view of the network to query devices, obtain flow statistics and install new paths on devices after computing their routes. With information collected from switches, Hedera treats the flow-to-path mapping as an optimization problem and uses a simulated annealing metaheuristic to efficiently look for feasible solutions close to the optimal one in the search space. SA reduces the search space by allowing only a single core switch to be used for each destination. Overall, the system delivers close to optimal performance and up to four times more bandwidth than ECMP.

Port-Switching based Source Routing (PSSR) [38] is proposed for the SecondNet architecture with arbitrary topologies and commodity switches. PSSR uses source routing, which requires that every node in the network know the complete path to reach a destination. It takes advantage of the fact that a datacenter is administered by a single entity (i.e., the intra-cloud topology is known in advance) and represents a path as a sequence of output ports in switches, which is stored in the packet header. More specifically, the hypervisor of the source VM inserts the routing path in the packet header, commodity switches perform the routing process with PSSR and the destination hypervisor removes PSSR information from the packet header and delivers the packet to the destination VM. PSSR also introduces the use of virtual ports, because servers may have multiple neighbors via a single physical port (e.g., in DCell and BCube topologies). The process performed by a switch is shown in Figure 4.8. Switches read the pointer field in the packet header to get the exact next output port number (step 1), verify the next port number in the lookup virtual-port table (step 2), get the physical port number (step 3) and, in step 4, update the pointer field and forward the packet. This routing method introduces some overhead (since routing information must be included in the packet header), but, according to the authors, can be easily implemented on commodity switches using Multi-Protocol Label Switching (MPLS) [71].

Bounded Congestion Multicast Scheduling (BCMS) [40], introduced to efficiently route flows in Fat-trees under the hose traffic model, aims at achieving bounded congestion and high network utilization. By using multicast, it can reduce traffic, thus minimizing performance interference and increasing application throughput [51]. BCMS is an online multicast scheduling algorithm that leverages OpenFlow to (i) collect bandwidth demands of incoming flows; (ii) monitor network load; (iii) compute routing paths for each flow; and (iv) configure switches (i.e., installing appropriate rules to route flows). The algorithm has three main steps, as follows. First, it checks the conditions of uplinks out of source

4We will not focus our discussion in OpenFlow in this chapter. It is discussed in Chapter 6.
ToR switches (as flows are initially routed towards core switches). Second, it carefully selects a subset of core switches in order to avoid congestion. Third, it further improves traffic load balance by allowing ToR switches to connect to core switches with most residual bandwidth. Despite its advantages, BCMS relies on a centralized controller, which may not scale to large datacenters under highly dynamic traffic patterns such as the cloud.

Like BCMS, Code-Oriented eXplicit multicast (COXcast) [43] also focuses on routing application flows through the use of multicasting techniques (as a means of improving network resource sharing and reducing traffic). COXcast uses source routing, so all information regarding destinations are added to the packet header. More specifically, the forwarding information is encoded into an identifier in the packet header and, at each network device, is resolved into an output port bitmap by a node-specific key. COXcast can support a large number of multicast groups, but it adds some overhead to packets (since all information regarding routing must be stored in the packet).

Layer 2. In the Spanning Tree Protocol (STP) [2], all switches agree on a subset of links to be used among them, which forms a spanning tree and ensures a loop-free network. Despite being typically employed in Ethernet networks, it does not scale, since it cannot use the high-capacity provided by topologies with rich connectivities (such as Fat-Trees [59]), limiting application network performance [61]. Therefore, only a single path is used between hosts, creating bottlenecks and reducing overall network utilization.

STP’s shortcomings are addressed by other protocols, including Multiple Spanning Tree Protocol (MSTP) [4], Transparent Interconnect of Lots of Links (TRILL) [10] and Link Aggregation Control Protocol (LACP) [1]. MSTP was proposed in an attempt to use the path diversity available in DCNs more efficiently. It is an extension of STP to allow switches to create various spanning trees over a single topology. Therefore, different Virtual LANs (VLANs) [3] can utilize different spanning trees, enabling the use of more links in the network than with a single spanning tree. Despite its objective, implementations only allow up to 16 different spanning trees, which may not be sufficient to fully utilize the high-capacity available in DCNs [9].

TRILL is a link-state routing protocol implemented on top of layer-2 technologies, but bellow layer-3, and designed specifically to address limitations of STP. It discovers and calculates shortest paths between TRILL devices (called routing bridges or, in short, RBRidges), which enables shortest-path multihop routing in order to use all available paths in networks with rich connectivities. RBRidges run Intermediate System to Intermediate System (IS-IS) routing protocol (RFC 1195) and handle frames in the following manner:

---

**Figure 4.8** PSSR overview (adapted from [38]).
the first RBridge (ingress node) encapsulates the incoming frame with a TRILL header (outer MAC header) that specifies the last TRILL node as the destination (egress node), which will decapsulate the frame.

Link Aggregation Control Protocol (LACP) is another layer-2 protocol used in DCNs. It transparently aggregates multiple physical links into one logical link known as Link Aggregation Group (LAG). LAGs only handle outgoing flows; they have no control over incoming traffic. They provide flow-level load balancing among links in the group by hashing packet header fields. LACP can dynamically add or remove links in LAGs, but requires that both ends of a link run the protocol.

There are also some recent studies that propose novel strategies for routing frames in DCNs, namely SPAIN (Smart Path Assignment in Networks) [59] and Portland [61]. SPAIN focuses on providing efficient multipath forwarding using COTS switches over arbitrary topologies. It has three components: i) path computation; ii) path setup; and iii) path selection. The first two components run on a centralized controller with global network visibility. The controller first pre-computes a set of paths to exploit the rich connectivities in the DCN topology, in order to use all available capacity of the physical infrastructure and to support fast failover. After the path computation phase, the controller combines these multiple paths into a set of trees, with each tree belonging to a distinct VLAN. Then, these VLANs are installed on switches. The third component (path selection) runs at end-hosts for each new flow; it selects paths for flows with the goals of spreading load across the pre-computed routes (by the path setup component) and minimizing network bottlenecks. With this configuration, end-hosts can select different VLANs for communication (i.e., different flows between the same source and destination can use distinct VLANs for routing). To provide these functionalities, however, SPAIN requires some modification to end-hosts, adding an algorithm to choose among pre-installed paths for each flow.

Portland [61] is designed and built based on the observation that Ethernet/IP protocols may have some inherent limitations when designing large-scale arbitrary topologies, such as limited support for VM migration, difficult management and inflexible communication. It is a layer-2 routing and forwarding protocol with plug-and-play support for multi-rooted Fat-Tree topologies. Portland uses a logically centralized controller (called fabric manager) with global visibility and maintains soft state about network configuration. It assigns unique hierarchical Pseudo MAC (PMAC) addresses for each VM to provide efficient, provably loop-free frame forwarding; VMs, however, do not have the knowledge of their PMAC and believe they use their Actual MAC (AMAC). The mapping between PMAC and AMAC and the subsequent frame header rewriting is performed by edge (ToR) switches. PMACs are structured as pod.position.port.vmid, where each field respectively corresponds to the pod number of the edge switch, its position inside the pod, the port number in which the physical server is connected to and the identifier of the VM inside the server. With PMACs, Portland transparently provides location-independent addresses for VMs and requires no modification in commodity switches. However, it has two main shortcomings: i) it requires a Fat-Tree topology (instead of the traditional multi-rooted oversubscribed tree); and ii) at least half of the ToR switch ports should be connected to servers (which, in fact, is a limitation of Fat-Trees) [19].

4.6 Addressing

Each server (or more specifically each VM) must be represented by a unique canonical address that enables the routing protocol to determine paths in the network. Cloud providers
typically employ LAN technologies for addressing VMs in datacenters, which means there is a single address space to be sliced among tenants and their applications. Consequently, tenants have neither flexibility in designing their application layer-2 and layer-3 addresses nor network isolation from other applications.

Some isolation is achieved by the use of Virtual LANs (VLANs), usually one VLAN per tenant. However, VLANs are ill-suited for datacenters for four main reasons [55, 56, 60, 78]: i) they do not provide flexibility for tenants to design their layer-2 and layer-3 address spaces; ii) they use the spanning tree protocol, which cannot utilize the high-capacity available in DCN topologies (as discussed in the previous section); iii) they have poor scalability, since no more than 4094 VLANs can be created and this is insufficient for large datacenters; and iv) they do not provide location-independent addresses for tenants to design their own address spaces (independently of other tenants) and for performing seamless VM migration. Therefore, providers need to use other mechanisms to allow address space flexibility, isolation and location independence for tenants while multiplexing them in the same physical infrastructure. We structure the discussion in three main topics: emerging technologies, separation of name and locator and full address space virtualization.

Emerging technologies. Some technologies employed in DCNs are: Virtual eXtensible Local Area Network (VXLAN) [11], Amazon Virtual Private Cloud (VPC) [5] and Microsoft Hyper-V [7]. VXLAN [11] is an Internet draft being developed to address scalability and multipath usage in DCNs when providing logical isolation among tenants. VXLAN works by creating overlay (virtual layer-2) networks on top of the actual layer-2 or on top of UDP/IP. In fact, using MAC-in-UDP encapsulation abstracts VM location (VMs can only view the virtual layer-2) and, therefore, enables a VXLAN network to be composed of nodes within distinct domains (DCNs), increasing flexibility for tenants using multi-datacenter cloud platforms. VXLAN adds a 24-bit segment ID field in the packet header (allowing up to 16 million different logical networks), uses ECMP to distribute load along multiple paths and requires IGMP (Internet Group Management Protocol) for forwarding frames to unknown destinations, or multicast and broadcast addresses. Despite the benefits, VXLAN header adds 50 bytes to the frame size, and multicast and network hardware may limit the usable number of overlay networks in some deployments.

Amazon VPC [5] provides full IP address space virtualization, allowing tenants to design layer-3 logical isolated virtual networks. However, it does not virtualize layer-2, which does not allow tenants to send multicast and broadcast frames [60]. Microsoft Hyper-V [7] is a hypervisor-based system that provides virtual networks for tenants to design their own address spaces; Hyper-V enables IP overlapping in different virtual networks without using VLANs. Furthermore, Hyper-V switches are software-based layer-2 network switches with capabilities to connect VMs among themselves, with other virtual networks and with the physical network. Hyper-V, nonetheless, tends to consume more resources than other hypervisors with the same load [8].

Separation of name and locator. VL2 [35] and Crossroads [55] focus on providing location independence for VMs, so that providers can easily grow or shrink allocations and migrate VMs inside or across datacenters. VL2 [35] uses two types of addresses: location-specific addresses (LAs), which are the actual addresses in the network, used for routing; and application-specific addresses (AAs), permanent address assigned to VMs that remain the same even after migrations. VL2 uses a directory system to enforce isolation among applications (through access control policies) and to perform the mapping between names and locators; each server with an AA is associated with the LA from the ToR it is connected to. Figure 4.9 depicts how address translation in VL2 is performed: the source hypervisor encapsulates the AA address with the LA address of the destination ToR for
each packet sent; packets are forwarded in the network through shortest-paths calculated by the routing protocol, using both ECMP and VLB; when packets arrive at the destination ToR switch, LAs are removed (packets are decapsulated) and original packets are sent to the correct VMs using AAs. To provide location-independent addresses, VL2 requires that hypervisors run a shim layer (VL2 agent), and that switches support IP-over-IP.

Figure 4.9 Architecture for address translation in VL2.

Crossroads [55], in turn, is a network fabric developed to provide layer agnostic and seamless VM migration inside and across DCNs. It takes advantage of the Software-defined Networking (SDN) paradigm [48] and extends an OpenFlow controller to allow VM location-independence without modifications to layer-2 and layer-3 network infrastructure. In Crossroads, each VM possess two addresses: a PMAC and a Pseudo IP (PIP), both with location and topological information embedded in them. The first one ensures that traffic originated from one datacenter and en route to a second datacenter (to which the VM was migrated) can be maintained at layer-2, while the second guarantees that all traffic destined to a migrated VM can be routed across layer-3 domains. Despite its benefits, Crossroads introduces some network overhead, as nodes must be identified by two more addresses (PMAC and PIP) in addition to the existing MAC and IP.

Full address space virtualization. Cloud datacenters typically provide limited support for multi-tenancy, since tenants should be able to design their own address spaces (similar to a private environment) [60]. Consequently, a multi-tenant virtual datacenter architecture to enable specific-tailored layer-2 and layer-3 address spaces for tenants, called NetLord, is proposed. At hypervisors, NetLord runs an agent that performs Ethernet+IP (L2+L3) encapsulation over tenants’ layer-2 frames and transfers them through the network using SPAIN [59] for multi-pathing, exploring features of both layers. More specifically, the process of encapsulating/decapsulating is shown in Figure 4.10 and occurs in three steps, as follows: (i) the agent at the source hypervisor creates L2 and L3 headers (with source IP being a tenant-assigned MAC address space identifier, illustrated as MAC_AS_ID) in order to direct frames through the L2 network to the correct edge switch; (ii) the edge switch forwards the packet to the correct server based on the IP destination address in the virtualized layer-3 header; (iii) the hypervisor at the destination server removes the virtual L2 and L3 headers and uses the IP destination address to deliver the original packet from the source VM to the correct VM. NetLord can be run on commodity switches and scale
the network hundreds of thousands of VMs. However, it requires an agent running on hypervisors (which may add some overhead) and support for IP forwarding on commodity edge (ToR) switches.

![Diagram](File:netlord_encapsulation_decapsulation.png)

**Figure 4.10** NetLord’s encapsulation/decapsulation process (adapted from [60]).

### 4.7 Research Challenges

In this section, we analyze and discuss open research challenges and future directions regarding datacenter networks. As previously mentioned, DCNs (a) present some distinct requirements from traditional networks (e.g., high scalability and resiliency); (b) have significantly different (often more complex) traffic patterns; and (c) may not be fully utilized, because of limitations in current deployed mechanisms and protocols (for instance, ECMP). Such aspects introduce some challenges, which are discussed next.

**Heterogeneous and optimal DCN design.** Presently, many Internet services and applications rely on large-scale datacenters to provide availability while scaling in and out according to incoming demands. This is essential in order to offer low response time for users, without incurring excessive costs for owners. Therefore, datacenter providers must build infrastructures to support large and dynamic numbers of applications and guarantee quality of service (QoS) for tenants. In this context, the network is an essential component of the whole infrastructure, as it represents a significant fraction of investment and contributes to future revenues by allowing efficient use of datacenter resources [62]. According to Zhang et al. [87], network requirements include (i) scalability, so that a large number of servers can be accommodated (while allowing incremental expansion); (ii) high server-to-server capacity, to enable intensive communication between any pair of servers (i.e., at full speed of their NICs); (iii) agility, so applications can use any available server when they need more resources (and not only servers located near their current VMs); (iv) uniform network utilization to avoid bottlenecks; and (v) fault tolerance to cope with server, switch and link failures. In fact, guaranteeing such requirements is a difficult challenge. Looking at these challenges from the providers point-of-view make them even more difficult to address and overcome, since reducing the cost of building and maintaining the network is seen as a key enabler for maximizing profits [62].

As discussed in Section 4.2, several topologies (e.g., [14, 23, 35, 36, 37]) have been proposed to achieve the desired requirements, with varying costs. Nonetheless, they (a) focus on homogeneous networks (all devices with the same capabilities); and (b) do not
provide theoretical foundations regarding optimality. Singla et al. [77], in turn, take an initial step towards addressing heterogeneity and optimality, as they (i) measure the upper-bound on network throughput for homogeneous topologies with uniform traffic patterns; and (ii) show an initial analysis of possible gains with heterogeneous networks. Despite this fact, a lot remains to be investigated in order to enable the development of more efficient, robust large-scale networks with heterogeneous sets of devices. In summary, very little is known about heterogeneous DCN design, even though current DCNs are typically composed of heterogenous equipment.

**Efficient and incremental expansion.** Providers need to be constantly expanding their datacenter infrastructures to accommodate ever-growing demands. For instance, Facebook has been expanding its datacenters for some years [6, 16, 26, 67]. This expansion is crucial for business, as the increase of demand may negatively impact scalability and performance (e.g., by creating bottlenecks in the network). When the whole infrastructure is upgraded, the network must be expanded accordingly, with a careful design plan, in order to allow efficient utilization of resources and to avoid fragmentation. To address this challenge, some proposals in the literature [28, 29, 76, 77] have been introduced to enlarge current DCNs without replacing legacy hardware. They aim at maximizing high bisection bandwidth and reliability. However, they often make strong assumptions (e.g., Legup [29] is designed for tree-like networks, and Jellyfish [76] requires new mechanisms for routing). Given the importance of datacenters nowadays (as home of hundreds of thousands of services and applications), the need for efficient and effective expansion of large-scale networks is a key challenge for improving provider profit, QoS offered to tenant applications and quality of experience (QoE) provided for users of these applications.

**Network sharing and performance guarantees.** Datacenters host applications with diverse and complex traffic patterns and different performance requirements. Such applications range from user-facing ones (such as web services and online gaming) that require low latency communication to inward computation (e.g., scientific computing) that need high network throughput. To gain better understanding of the environment, studies [17, 18, 20, 45, 73] conducted measurements and concluded that available bandwidth for VMs inside the cloud platform can vary by a factor of five or more during a predefined period of time. They demonstrate that such variability ends up impacting overall application execution time (resulting in poor and unpredictable performance). Several strategies (including [49, 64, 70, 75, 83]) have been proposed to address this issue. Nonetheless, they have one or more of the following shortcomings: (i) require complex mechanisms, which, in practice, cannot be deployed; (ii) focus on network sharing among VMs (or applications) in a homogeneous infrastructure (which simplifies the problem [84]); (iii) perform static bandwidth reservations (resulting in underutilization of resources); or (iv) provide proportional sharing (no strict guarantees). In fact, there is an inherent trade-off between providing strict guarantees (desired by tenants) and enabling work-conserving sharing in the network (desired by providers to improve utilization), which may be exacerbated in a heterogenous network. We believe this challenge requires further investigation, since such high-performance networks ideally need simple and efficient mechanisms to allow fair bandwidth sharing among running applications in a heterogeneous environment.

**Address flexibility for tenants.** While network performance guarantees require quantitative performance isolation, address flexibility needs qualitative isolation [60]. Cloud DCNs, however, typically provide limited support for multi-tenancy, as they have a single address space divided among applications (according to their needs and number of VMs). Thereby, tenants have no flexibility in choosing layer-2 and layer-3 addresses for applications. Note that, ideally, tenants should be able to design their own address spaces (i.e.,
they should have similar flexibility to a private environment), since already developed applications may necessitate a specific set of addresses to correctly operate without source code modification. Some proposals in the literature [35, 55, 60] seek to address this challenge either by identifying end-hosts with two addresses or by fully virtualizing layer-2 and layer-3. Despite adding flexibility for tenants, they introduce some overhead (e.g., hypervisors need a shim layer to manage addresses, or switches must support IP-over-IP) and require resources specifically used for address translation (in the case of VL2). This is an important open challenge, as the lack of address flexibility may hinder the migration of applications to the cloud platform.

**Mechanisms for load balancing across multiple paths.** DCNs usually present path diversity (i.e., multiple paths between servers) to achieve horizontal scaling for unpredictable traffic matrices (generated from a large number of heterogeneous applications) [35]. Their topologies can present two types of multiple paths between hosts: uniform and non-uniform ones. ECMP is the standard technique used for splitting traffic across equal-cost (uniform) paths. Nonetheless, it cannot fully utilize the available capacity in these multiple paths [65]. Non-uniform multiple paths, in turn, complicate the problem, as mechanisms must take more factors into account (such as path latency and current load). There are some proposals in the literature [15, 38, 40, 43] to address this issue, but they either cannot achieve the desired response times (e.g., Hedera) [30] or are developed for specific architectures (for instance, PSSR for SecondNet). Chiesa et al. [24] have taken an initial approach towards analyzing ECMP and propose algorithms for improving its performance. Nevertheless, further investigation is required for routing traffic across both uniform and non-uniform parallel paths, considering not only tree-based topologies, but also newer proposals such as random graphs [76, 77]. This investigation should lead to novel mechanisms and protocols that better utilize available capacity in DCNs (for example, eliminating bottlenecks at level-0 links in DCell).

4.8 **Summary**

In this chapter, we have presented basic foundations of datacenter networks and relevant standards, as well as recent proposals in the literature that address limitations of current mechanisms. We began by studying network topologies in Section 4.2. First, we examined the typical topology utilized in today’s datacenters, which consists of a multi-rooted tree with path diversity. This topology is employed by providers to allow rich connectivity with reduced operational costs. One of its drawbacks, however, is the lack of full bisection bandwidth, which is the main motivation for proposing novel topologies. We used a three-class taxonomy to organize the state-of-the-art datacenter topologies: switch-oriented, hybrid switch/server and server only topologies. The distinct characteristic is the use of switches and/or servers: switches only (Fat-Tree, VL2 and OSA), switches and servers (DCell and BCube) and only servers (CamCube) to perform packet routing and forwarding in the network.

These topologies, however, usually present rigid structures, which hinders incremental network expansion (a desirable property for the ever-growing cloud datacenters). Therefore, we took a look at network expansion strategies (Legup, Rewire and Jellyfish) in Section 4.3. All of these strategies have the goal of improving bisection bandwidth to increase agility (the ability to assign any VM of any application to any server). Furthermore, the design of novel topologies and expansion strategies must consider the nature of traffic in
DCNs. In Section 4.4, we summarized recent measurement studies about traffic and discussed some proposals that deal with traffic management on top of a DCN topology.

Then, we discussed routing and addressing in Sections 4.5 and 4.6, respectively. Routing was divided in two categories: layer-3 and layer-2. While layer-3 routing typically employs ECMP and VLB to utilize the high-capacity available in DCNs through the set of multiple paths, layer-2 routing uses the spanning tree protocol. Despite the benefits, these schemes cannot efficiently take advantage of multiple paths. Consequently, we briefly examined proposals that deal with this issue (Hedera, PSSR, SPAIN and Portland). Addressing, in turn, is performed by using LAN technologies, which does not provide robust isolation and flexibility for tenants. Towards solving these issues, we examined the proposal of a new standard (VXLAN) and commercial solutions developed by Amazon (VPC) and Microsoft (Hyper-V). Furthermore, we discussed proposals in the literature that aim at separating name and locator (VL2 and Crossroads) and at allowing full address space virtualization (NetLord).

Finally, we analyzed open research challenges regarding datacenter networks: (i) the need to design more efficient DCNs with heterogeneous sets of devices, while considering optimality; (ii) strategies for incrementally expanding networks with general topologies; (iii) network schemes with strict guarantees and predictability for tenants, while allowing work-conserving sharing to increase utilization; (iv) address flexibility to make the migration of applications to the cloud easier; and (v) mechanisms for load balancing traffic across different multiple parallel paths (using all available capacity).

Having covered the operation and research challenges of intra-datacenter networks, the next three chapters inside the networking and communications part discuss the following subjects: inter-datacenter networks, an important topic related to cloud platforms composed of several datacenters (e.g., Amazon EC2); the emerging paradigm of Software-defined Networking (SDN), its practical implementation (OpenFlow) and how these can be applied to intra- and inter-datacenter networks to provide fine-grained resource management; and mobile cloud computing, which seeks to enhance capabilities of resource-constrained mobile devices using cloud resources.

Bibliography


