# Optimal Multicast in Virtualized Datacenter Networks with Software Switches

Rui Zhu*, Di Niu*, Baochun Li† and Zongpeng Li‡
*Department of Electrical and Computer Engineering, University of Alberta
†Department of Electrical and Computer Engineering, University of Toronto
‡Department of Computer Science, University of Calgary

*Abstract*—**Virtualized datacenter networks have been deployed in production platforms, e.g., Amazon VPC and VMware's NVP, to offer the flexibility of network management to enterprise-level clients. A common characteristic of these platforms is that they adopt software switches, such as Open vSwitch (OvS), instead of hardware switches to transfer data between VMs. Although group communication is common in enterprise applications, the unique characteristics of software switches have posed new challenges to the design of multicast protocols. How logical multicast can be optimally performed with software switches is still not well understood. In this paper, we observe that unlike hardware switches, the per-stream output rate in a software switch critically depends on the packet processing overhead of flow cloning. We study the optimal OvS multicast topology with or without the help of additional dedicated software switches called service nodes, and formulate the throughput maximization as a new class of *degree-supervised* combinatorial graph problems due to the presence of flow cloning costs. We propose a linear-time optimal solution that translates into simple forwarding rules installed at each software switch. Through emulation-based OvS profiling and extensive simulation results, we demonstrate that our proposed logical multicast solutions can significantly improve session throughput with the ability to handle load balancing and latency issues, as compared to the state-of-the-art in the literature.**

## I. Introduction

Designed to improve performance isolation and management flexibility, server virtualization technologies, such as VMware and Xen, allow multiple virtual machines (VMs) to be hosted by the same physical machine. In the same vein, virtualized datacenter networks [1] have recently been proposed to create logical virtual networks, each for a tenant, with independent service models and addressing architectures. They have already been deployed in production systems targeting enterprise clients, such as Amazon's Virtual Private Cloud [2] and VMware's NVP [3].

A common characteristic of these production systems is that data transfers between VMs rely on virtualized software switches — such as Open vSwitch (OvS) [4] — installed in host hypervisors, rather than hardware switches. The use of Open vSwitch is not a surprise. First, it is an OpenFlow-compliant multi-layer switch designed to enable easy network protocol customization. Moreover, as the number of tenants scales up, it does not suffer from flow table size constraints in traditional hardware switches[1]. Such constraints have been the main obstacle to the adoption of software-defined networking (SDN) with hardware switches in multi-tenant datacenters [5], [6]. As such, software switches are also able to support more complex and fine-grained rules and actions.

[1]Due to hardware space limitations, only a limited number of rules, e.g., 1000, can be supported in OpenFlow-enabled hardware switches.

In current virtualized datacenter networks, the communication between a pair of VMs is handled by tunneling [3] through software switches in the host hypervisors of both source and destination VMs. With the prevalence of modern data analytics using Apache Hadoop and Spark, *multicast* is an important communication pattern in enterprise applications, as large volumes of data are transferred from a mapper to multiple reducers. Yet, how multicast topologies are to be optimally constructed in such virtualized datacenter networks remains to be an open challenge.

In this paper, we focus on multicast communication between VMs in virtualized datacenter networks with software switches, such as Open vSwitch. In this context, the use of software switches has posed new and interesting challenges to the design of multicast solutions. *First,* the per-stream forwarding throughput of a software switch is mainly limited by *flow cloning*, when packets in a flow are forwarded onto multiple output ports [7] — a key operation in multicast sessions. *Second,* running on commodity x86 architectures, the performance of software switches is also limited by their CPU and I/O resources when incoming packets are processed based on the installed forwarding rules [8], [9]. In fact, we have profiled the performance characteristics of software switches, using Open vSwitch as an example, by transmitting real packets in the Mininet emulation testbed. Our measurement results suggest that the per-stream output packet rate of an Open vSwitch critically depends on both the input packet rate and its output degree in a numerically characterizable model. *Third,* as compared to the switching cost, link bandwidth is not a bottleneck in today's datacenters featuring 40 Gigabit Ethernet.

Taking into account these new characteristics of software switches, we aim to construct optimal multicast topologies in a virtualized datacenter network of software switches in order to maximize session throughput. In particular, we consider two types of multicast topologies: (1) small multicast sessions, consisting of VMs interconnected by hypervisor-to-hypervisor tunnels [9]; and (2) larger multicast sessions, using additional *service nodes* — dedicated hosts running software switches — to help relay traffic [3].

In both scenarios, the total throughput of a multicast session is critically determined by the output degree distribution among the nodes, i.e., the topology in which the nodes are interconnected. We formulate the throughput maximization problem with or without the help of service nodes as a new class of combinatorial *degree-supervised* graph problem. In a small session without service nodes, Based on the profiled performance model of an Open vSwitch, we directly give the *optimal* multicast topology connecting the source and all destination host hypervisors. More importantly, in a larger session, we analytically characterize the class of multicast trees with the optimal throughput and propose a *linear-time* algorithm to build such a tree, leading to *closed-form* SDN multicast forwarding rules that can be computed and installed on each participating switch in constant time. Aside from throughput optimality, our solution also minimizes the worst-case latency in the session among all the trees of optimal throughput. Moreover, our solution can further lead to balanced loads among service nodes when multiple sessions are using the service.

Driven by our profiling results, we demonstrate through extensive simulations that, as compared to state-of-the-art multicast solutions and several straightforward optimization heuristics, our proposed algorithms can improve session throughput by a substantial margin, yet maintaining balanced loads across service nodes.

## II. Related Work

In virtualization-dominated IT services, datacenter network virtualization has emerged as a promising solution to improve network manageability and performance isolation, and has been extensively studied recently [10]–[12]. For example, NetLord [10] aims to meet the scalability of tenants using shared infrastructures in datacenters. PortLand [11] uses a hierarchical Pseudo MAC (PMAC) addressing of VMs for L2 routing with a variation of FatTree topologies. An emerging trend in network virtualization is to use software switches, such as Open vSwitches (OvS), to transfer data. For example, in Amazon's Virtual Private Cloud [2], an OvS is installed inside each host hypervisor to forward packets from its associated VMs.

Group communication occurs frequently in many VM-based applications running in datacenters. However, existing multicast routing protocols are no longer well suited for multicast in virtualized datacenter networks with software switches. On one hand, traditional IP multicast protocols in ISP networks like PIM [13] and CBT [14] are not designed to detect network topology or end-to-end reachability. Without a global view, they are unable to optimize session throughput or balance switching loads in datacenters.

On the other hand, by exploiting the global view and centralized controllers, SDN-based multicast routing schemes have been proposed for datacenter networks to reduce forwarding states [15] or to minimize the link cost of multicast trees [16]. However, these solutions rely on OpenFlow-enabled hardware switches, which are either not always available in an existing datacenter or even if available, can only support a limited number of rules due to hardware space limitations. In contrast, for OvS-based multicast in virtualized datacenter networks, the number of forwarding rules or link cost is not a critical issue, since link bandwidth is not a bottleneck in modern datacenter networks with 40-Gigabit Ethernet [17] and the propagation delay between nodes in datacenters is also negligible [18].

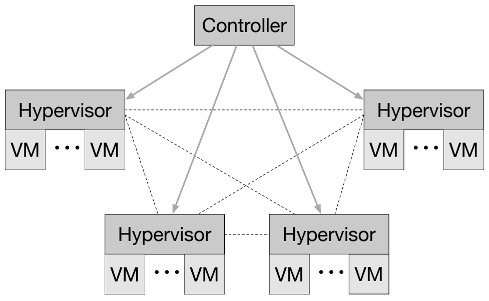


Fig. 1. A complete graph of host-hypervisors (each running an OvS) over point-to-point IP tunnels.

Measurement studies [7] have shown that in software switches like OvS, sending the same flow on multiple output ports will lead to a reduced throughput per stream. Specifically, [7] has reported that the packet rate per stream drops by 30% when a flow is sent out twice, and another 25% when it is copied one more time. Since the outgoing bandwidth of a datacenter host is abundant, such performance degradation is mainly attributed to the CPU/memory overhead of packet processing and flow cloning in software switches. Our measurements using Open vSwitches have confirmed these facts. In addition, we build a performance model to characterize the input-output relationship within a software switch, based on which an optimal multicast topology can be constructed.

## III. Background and Existing Solutions

In this paper, we focus on a specific network virtualization architecture, adopted by Amazon's Virtual Private Cloud (VPC) [2], [9] and VMware's NVP [3], that uses software switches (such as OvSes) to handle data transfers between VMs. In such an architecture, as illustrated in Fig. 1, each physical machine has a hypervisor that hosts multiple VMs. An OvS is installed inside each host hypervisor and will serve as the virtual tunnel endpoint (VTEP) [3] in the communication between VMs: the source VM sends packets to its own host hypervisor, which then forwards them to the host hypervisor to which the destination VM is attached.

A tenant of such a virtualized datacenter network can form a logical multicast session across its VMs, by configuring the forwarding rules in the corresponding OvSes via a centralized SDN controller. Under such an architecture, there are currently two kinds of logical multicast solutions: 1) point-to-point logical multicast, and 2) logical multicast via a service node.

**Point-to-point logical multicast.** A basic approach to OvS-based logical multicast [9] is to let the hypervisor of the source VM send the same data stream to all destination hypervisors at the same time in a star topology, based on point-to-point

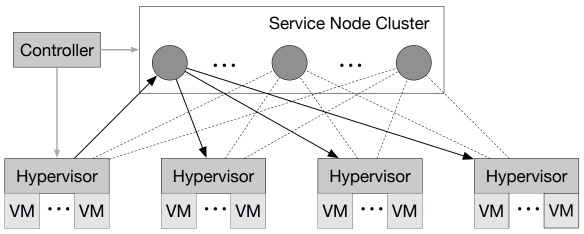


Fig. 2. Logical multicast via the help of service nodes.

tunnels, as shown in Fig. 1. However, as the number of receivers increases, such a simple star topology will suffer from serious throughput losses due to the overly high *flow cloning* cost at the source OvS.

**Optimal logical multicast via service nodes.** Another solution proposed by VMware's NVP [3] mainly targets larger sessions and uses *service nodes* as helpers to relay traffic, thus reducing the flow cloning workload at host hypervisors while improving reliability. In the implementation of NVP [3], service nodes are additional physical forwarding servers (x86-based hosts running OvSes). As shown in Fig. 2, the source hypervisor first tunnels the packet stream to *one* of the service nodes, which then forwards packets to all destination hypervisors. By using a service node cluster, it is further guaranteed that the failure of any single service node will not disrupt the logical multicast traffic.

However, using a single service node for each session may still be suboptimal, since all the forwarding burden is put on the single service node. To construct a topology with the optimal throughput for each multicast session, the operator must decide 1) how many service nodes are to be used, 2) which service nodes are to be selected for forwarding, and 3) how to connect all the nodes, including the source, the selected service nodes and destination nodes.

## IV. OvS Performance Profiling

In this section, we benchmark the performance of software switches, using OvS as an example, as a function of the rate of the input flow and the number of output streams which the input flow is cloned into. We use Mininet v2.3 [19] network emulation platform, which supports OpenFlow v1.3, to inject real packets into an OvS and measure the data rate at each output port. Starting from this version, OpenFlow has supported *Group Table* rules, which allow us to write a multicast rule to an action list for output to multiple ports. Based on all the measurements data, we will build a model to characterize the input-output relationship at an OvS.

First, we evaluate the effect of *flow cloning*, when an incoming stream needs to be replicated onto multiple output ports. In this experiment, we use a single OvS to connect to at most 15 Mininet hosts, and let one of them be the source host to multicast packets to the others. We use the `Iperf` utility as a load generator to send UDP packets at a certain rate from the source to others via the OvS. The input rate of the OvS is measured and ranges from 0.2 Gbps to 0.8 Gbps. The number of output streams ranges from 1 to 14.

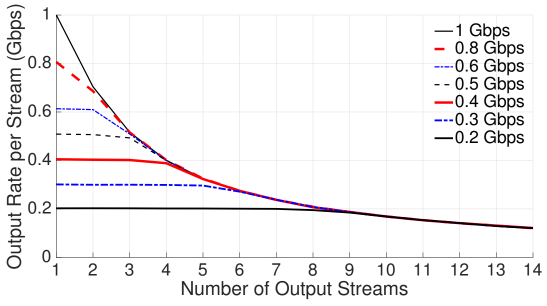


Fig. 3. The relationship between output rate per stream and the number of output streams under different fixed input rates.

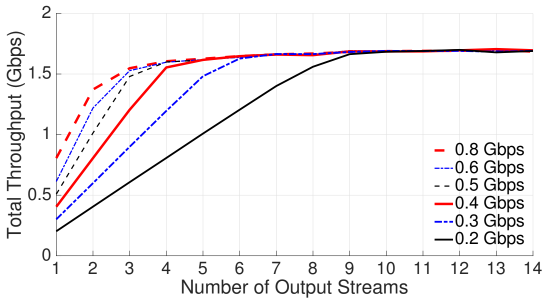


Fig. 4. The total output rate is an increasing concave function of the number of output streams under each fixed input rate.

As shown in Fig. 3, for each fixed input rate, the output rate per stream decreases as the number of output streams increases, confirming the result shown in another measurement study [7]. We also find that when the input rate is below 0.8 Gbps, the output rate per stream does not decrease for small multicast sessions. Yet it starts to drop when the number of output streams exceeds a certain value. When the input rate is 0.8 Gbps or larger, the output rate per stream starts to decrease even when there are two output streams.

Then, we show the relationship between the total output rate and the number of output streams in Fig. 4. We can see that given any fixed input rate, as the number of output streams increases, the total output throughput increases, yet its marginal increase rate decreases. Thus, we can conclude that the total output rate is an increasing concave function of the number of output streams under a fixed input rate.

For comparison, we also check the output rate for a unicast session from a single source host to a single destination host connected via a line network of OvSes, and plot the output rate of the session as the number of OvSes between the source and the destination increases. As we can observe in Fig. 5, the output rate does not drop as more OvSes are used. This is different from the multicasting scenario in Fig. 3, where the output rate will drop significantly as the number of output streams increases. Thus, we can conclude that store-and-forward does not affect the throughput of the OvS, no matter how many hops are involved. However, flow cloning, i.e., forwarding the same input flow onto multiple output ports, will affect throughput.

Finally, we investigate the relationship between the input rate and the total output rate. In Fig. 6, we can see that the

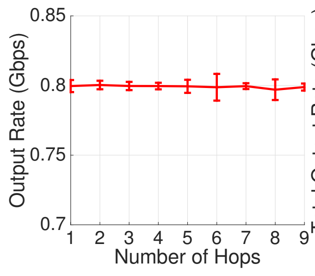


Fig. 5. Output Rate of unicast sessions.

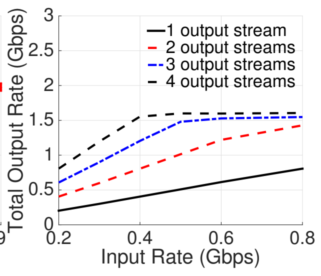


Fig. 6. Input Rate vs. Total Output Rate

output rate of an OvS increases as the input rate increases. We can also see that the output rate stops increasing after the input rate exceeds a certain value, and having more output streams will make such an increasing trend stop earlier. Furthermore, the final cap that the total throughput reached still increases as the number of output streams increases.

These observations reveal that we can build a model for the total output throughput of an OvS as a function of both the number of output streams and the input rate. Formally speaking, for an OvS with $d$ output streams and an input rate of $r$ Gbps, where $d$ is a nonnegative integer, denote its output rate per stream by $f(d, r)$ and denote its total throughput by $U(d, r) := d \cdot f(d, r)$. From Fig. 3, we can see that the output rate per stream can be expressed as

$$f(d, r) = \min(g(d), r), \quad d = 1, 2, \dots, \ r \le 0.8.$$

i.e., the output rate per stream is the minimum between the input rate $r$ and another curve $g(d)$, where $g(d)$ corresponds to the curve in Fig. 3 with an input rate of 0.8 Gbps. Clearly, $g(d)$ is a decreasing function of the out degree $d$. Therefore, we have the total output throughput given by $U(d, r) = \min(dg(d), dr)$.

From Fig. 6, we can observe that $U(d, r)$ is a *concave* function of $r$ for each fixed $d$, and in Fig. 4 we can further see that the curve with a 0.8 Gbps input rate, i.e., $dg(d)$, is also *concave* in terms of the (integer) out degree $d$. Note that 0.8 Gbps is the maximum packet rate generated by `Iperf` under the default packet length setting. We can increase packet length and get a higher input rate in `Iperf`. In this case, $f(d, r)$ will be always decreasing in $d$ like in $g(d)$ but is larger than $g(d)$ when $d = 1$.

We will utilize these properties to derive the optimal solutions to our logical multicast problems in Sec. V-A and Sec. VI-B.

## V. Problem Formulation

In a multicast among VMs, the source VM is sending packets via the OvS in its own host hypervisor to the corresponding OvSes in the host hypervisors of the receiving VMs. Each receiving VM then acquires data from its own host hypervisor. Therefore, we only need to consider the multicast among the participating host hypervisors (or *hypervisors* for brevity).

### A. Small Sessions without Service Nodes

We first consider a small OvS-based multicast session involving only a few VMs, without the help of service nodes, where the hosts of all the involved hypervisors are most likely co-located within a same server rack or connected by the same Tier-2 switch or edge switch. Therefore, we can assume that the propagation delay between nodes is negligible [20] and the virtual links between them have abundant bandwidth in 40 Gigabit datacenters [12], [18]. As a result, unlike traditional networking, the main bottleneck is not the network bandwidth, but lies in the processing capability of the software switches in the hypervisors.

Let $G = (V, E)$ denote the logical overlay network of all participating hypervisors $V = \{h_1, \dots, h_n\}$, where $h_1$ is the source hypervisor and $h_2, \dots, h_n$ are destination hypervisors, and we denote the source rate of this multicast session by $r$. $E$ is the set of all logical point-to-point tunnels interconnecting the hypervisors $V$. We call each logical tunnel a "link."

Without any service node, the optimal logical multicast from $h_1$ to all the other nodes in $G$ should maximize the total receiving rate of all the receivers $h_2, \dots, h_n$. Since the total receiving rate is equal to the total sending rate plus the source rate $r$, we can equivalently maximize the total sending rate of participating hypervisors $h1, \dots, h_n$. Formally speaking, the problem is to form a spanning tree $T$ connecting all nodes in $G$ such that the total session throughput is maximized, i.e.,

$$\underset{T}{\text{maximize}} \quad \sum_{v \in T} U\left(d_T^o(v), r_T(v)\right) \tag{1}$$

$$\text{subject to} \quad T \text{ is a spanning tree of } G, \tag{2}$$

where $U(d_T^o(v), r_T(v))$ represents the output rate of hypervisor $v$, which depends on the number of its output streams, or its out degree $d_T^o(v)$ in the formed spanning tree $T$, as well as on the input rate denoted by $r_T(v)$ that node $v$ receives from its parent in the tree $T$.

Clearly, the combinatorial problem above is different from the well-known min-cost spanning tree problem or Steiner Tree [21] which aims to minimize the total edge cost of $T$. Here the challenge is that the input $r_T(v_1)$ of a node $v_1$ depends on its parent $v_0$'s output per stream, which depends on the out degree of the parent $v_0$ as well as its input rate $r_T(v_0)$, in a recursive fashion. Hence, we call this type of problem a *Degree Supervised Graph* problem. In Sec. VI-A, we will derive a closed-form solution to (1), which directly translates the forwarding rules to be installed on the OvSes of corresponding hypervisors.

### B. Larger Sessions with the Help of Service Nodes

For a larger multicast session involving many host hypervisors, it is helpful to use service nodes, which are additional physical forwarding servers (x86-based hosts running OvSes), to relieve the traffic forwarding burden on the host hypervisors. When a cluster of dedicated service nodes is introduced, there is no point for each host hypervisor to help forward packets. Therefore, in this case, we require the source hypervisor to send packets to only one service node, which may further forward data to other service nodes for traffic relaying, as illustrated in Fig. 7. Each receiving hypervisor only needs to connect to one of the service nodes to receive data.

For a larger session with the help of service nodes (SNs), there are more factors to be considered. *First*, we need to maximize a session's throughput by jointly deciding 1) which and how many service nodes to employ, and 2) the connection topology among service nodes and receiving hypervisors, i.e., to which hypervisor or other service nodes should each service node forward its received packet stream. *Second*, when choosing service nodes for each session, we also need to consider the loads that are already imposed on each service node and route traffic away from already heavily loaded service nodes. *Third*, since in a larger session, hosts may be from different server racks spanning several aggregation switches or even core switches, we wish to reduce the number of hops from the source to any receiver hypervisor in order to confine the propagation delays.

Formally speaking, for a particular session, denote the set of service nodes by $F = \{s_1, \ldots, s_m\}$, in addition to the set of all source and receiving hypervisors $V = \{h_1, \ldots, h_n\}$, where $h_1$ is still the source hypervisor. The source rate of the multicast session is still $r$.

We use a binary integer $y_i \in \{0, 1\}$ to indicate whether service node $s_i$ is employed, and use $x_{i,j} \in \{0, 1\}$ to indicate whether hypervisor $h_j$ will be a child of $s_i$ (receiving packets from $s_i$). Let $F_s$ denote all the service nodes employed by the session. Clearly, the topology connecting all the employed service nodes should be a spanning tree in the complete graph $G_s$ formed by all the employed service nodes $F_s$. We denote this spanning tree by $T$.

In this case, since the receiving hypervisors $h_2, \ldots, h_n$ are not forwarding packets, the total session throughput is the summation of sending rates of all service nodes. Assume each service node $s_i$ has a weight $w_i$, representing the existing workload on $s_i$. Then, we aim to maximize the total session throughput penalized by the total weight of all the selected service nodes $F_s$, where the penalty is used to avoid selecting already heavily loaded service nodes. That is, we need to jointly determine the selection of service nodes $\{y_i\}$, the connection topology (spanning tree) of the selected service nodes $T$, and $\{x_{i,j}\}$, i.e., from which service node each hypervisor should receive packets, by solving the following problem:

$$\max_{\boldsymbol{x},\boldsymbol{y},T} \sum_{s_i \in F} U\Bigg( \sum_{h_j \in V} x_{i,j} + d_T^o(s_i), r_T(s_i) \Bigg) - \lambda \sum_{s_i \in F} y_i w_i \tag{3}$$

$$\text{s.t. } x_{i,j}, y_i \in \{0, 1\}, \quad \forall s_i \in F, h_j \in V \tag{4}$$

$$x_{i,j} \le y_i, \quad \forall s_i \in F, h_j \in V \tag{5}$$

$$\sum_{s_i \in F} x_{i,j} = 1, \quad \forall h_j \in V \tag{6}$$

$$T \text{ is a spanning tree of } G_s, \tag{7}$$

where $U(\sum_{h_j \in V} x_{i,j} + d_T^o(s_i), r_T(s_i))$ is the output rate of service node $s_i$, in which $r_T(s_i)$ denotes the input rate of $s_i$, and $\sum_{h_j \in V} x_{i,j} + d_T^o(s_i)$ gives the total number of output streams of service node $s_i$. The rationale is that $d_T^o(s_i)$ represents the out degree of $s_i$ in the spanning tree $T$ of all the employed service nodes in $G_s$. Thus, $d_T^o(s_i)$ is the number of output streams at $s_i$ going to other service nodes. In the meantime, the term $\sum_{h_j \in V} x_{i,j}$ is the number of output streams at $s_i$ going to receiving hypervisors. Therefore, $\sum_{h_j \in V} x_{i,j} + d_T^o(s_i)$ represents the total number of output streams at service node $s_i$ Finally, $\lambda > 0$ is a multiplier to balance the trade-off between throughput maximization and load balancing.

Again, (3) is a new type of *degree-supervised* combinatorial graph problem, which also involves integer decision variables. We will propose a *linear-time* algorithm to find an optimal solution to problem (3), and among all such optimal multicast trees, we find the one with the minimum height to confine propagation delays.

## VI. Optimal Solutions and Forwarding Rules

We provide efficient solutions to form optimal multicast trees for both small sessions without service nodes and larger sessions with service nodes, and directly translate them to SDN forwarding rules to be installed on each OvS.

### A. Small Sessions without Service Nodes

Due to the property of the input-output relationship of OvS profiled in Sec. IV, the optimal multicast tree of small sessions can be given in a closed form:

**Theorem 1.** *Suppose $U(d^o, r)$ is concave, increasing and positive with respect to $d^o$ with $U(0, r) = 0$, $U(1, r) = r$ for all $r > 0$, and is also increasing and positive with respect to $r > 0$. Then, the optimal solution to problem* (1) *is a path of all the nodes in $V$, i.e., $h_1 \to h_2 \to \ldots \to h_n$.*

*Proof.* We first provide an upper bound on the objective function of (1) and then show that the path from $h_1$ to $h_n$ can achieve such a bound. The upper bound can be derived by Jensen's inequality as follows. As a tree, there must be at least one leaf node in a multicast tree. For convenience, denote $d_i$ as the degree of each node $h_i$ and $r_i$ as its input stream rate. Recall that $h_1$ is the source node and node $h_n$ is a leaf node. As a source node, we have

$$U(d_T^o(h_n), r_T(h_n)) = \begin{cases} U(d_1, r_1), & i = 1 \\ U(d_i - 1, r_i), & i = 2, \ldots, n-1 \\ U(0, r_i), & i = n. \end{cases}$$

And for all $h_i$ we have $U(d_T^o(h_n), r(h_n)) \le U(d_T^o(h_n), r)$. Thus, the total utility is

$$\begin{aligned} \sum_{i=1}^{n} U(d_T^o(h_i), r(h_i)) &= U(d_1, r_i) + \sum_{i=2}^{n-1} U(d_i - 1, r_i) \\ &\le U(d_1) + \sum_{i=2}^{n-1} U(d_i - 1, r). \end{aligned}$$

As a tree, we have the constraint $\sum_{i=1}^{n} d_i = 2(n-1)$. Let $d'_1 := d_1$, and $d'_i := d_i - 1$ for all $i = 2, \ldots, n-1$. Thus, we have $\sum_{i=1}^{n-1} d'_i = n-1$, and the total utility is

$$\sum_{i=1}^{n} U(d_T^o(h_i), r_T(h_i)) \leq \sum_{i=1}^{n-1} U(d'_i, r) \leq (n-1)U(1, r) \quad (8)$$

by Jensen's inequality. Now we show that the path graph can achieve (8). As a path graph $h_1 \to \ldots \to h_n$, we have $U(d_T^o(h_i), r_T(h_i)) = U(1, r_T(h_i)) = r$. Therefore, both inequalities in (8) can achieve equalities. □

The assumption that $U(d^o, r)$ is concave in $d^o$ has been verified by the OvS performance profiling results that we have presented in Sec. IV. In Sec. VII, we will show that the proposed line network of hypervisors indeed achieves a higher total receiving rate than the naive star topology and binary/ternary trees. The forwarding rules in the line network $h_1 \to h_2 \to \ldots \to h_n$ is straightforward: each node $h_i$ (except $h_1$) receives from $h_{i-1}$ and each node $h_i$ (except $h_n$) sends to $h_{i+1}$. The insight from Theorem 1 is that we should make the out degrees in the network as evenly distributed as possible, an idea we will exploit again to solve the harder problem for larger sessions with service nodes.

## B. Optimal Multicast with Service Nodes

In the presence of service nodes, we notice that solving problem (3) is equivalent to selecting the best $k^*$ service nodes, and forming the optimal spanning tree among all these $k^*$ service nodes and finally connecting the receiving hypervisors $h_2, \ldots, h_n$ to these service nodes as leaf nodes in an optimal way. In the following, we first show that given a particular set of $k$ selected service nodes, the optimal multicast tree of these selected service nodes and receiving hypervisors must satisfy a certain degree distribution. Based on this fact, we can give a linear-time algorithm to solve problem (3) by searching for the best $k$ value.

First, we present two lemmas to characterize the structure of optimal solutions to (3) given a particular set of $k$ selected service nodes. In this case, without loss of generality, we abuse the notation for simplicity and denote the selected $k$ service nodes as $s_1, \ldots, s_k$, with $s_1$ being the root that receives packets from the source $h_1$. Thus, $\sum_{i \in F} y_i w_i$ is fixed. Now we will show that the optimal total throughput, i.e., the first term in (3), is maximized when the out degrees $d_T^o(s_i)$ of service nodes are as evenly distributed as possible.

Let $d_i := d_i^1 + d_i^2$ denote the out degree of $s_i$, where $d_i^1 := \sum_{h_j \in V} x_{i,j}$ is the number of receiver hypervisors connected to $s_i$, and $d_i^2 := d_T^o(s_i)$ is the number of other service nodes connected to $s_i$. Let $d_i^{(k)}$ and $r_i^{(k)}$ denote the $d_i$ and $r_i$ in an optimal multicast tree of the $k$ selected service nodes and all the hypervisors.

**Lemma 1.** *Suppose $f(d, r) = \min(g(d), r)$ and $g(\cdot)$ is a positive and decreasing function. Suppose $k$ service nodes $s_1, \ldots, s_k$ are already selected, with $s_1$ being the root. In an optimal solution to (3), the out degrees of the nodes on any path from the root $s_1$ in the formed multicast tree are non-increasing. In addition, we have $f(d_i^{(k)}, r_i^{(k)}) = f(d_1^{(k)}, r)$ and*

$$U\big(d_i^{(k)}, r_i^{(k)}\big) = d_i^{(k)} f\big(d_1^{(k)}, r\big), \text{ for all } i = 1, \ldots, k. \quad (9)$$

*Proof.* Consider a particular path $s_1, s_{i_1}, \ldots, s_{i_l}, \ldots$ of the service nodes in the formed multicast tree $T^{(k)}$. The sum of $U(\cdot, \cdot)$ along these service nodes is

$$\begin{aligned} & U(d_1, r) + U(d_{i_1}, r_{i_1}) + \ldots + U(d_{i_l}, r_{i_l}) + \ldots \\ = \;& d_1 f(d_1, r) + \ldots + d_{i_l} f(d_{i_l}, r_{i_l}) + \ldots \\ = \;& d_1 r_{i_1} + \ldots + d_{i_l} r_{i_{l+1}} + \ldots . \end{aligned}$$

Since the input rates along this path are non-increasing, according to the rearranged inequality, the sum is upper-bounded by the case when the out degrees in $T^{(k)}$ are also non-increasing, i.e., in an optimal multicast tree, we have $d_1^{(k)} \geq d_{i_1}^{(k)} \geq d_{i_l}^{(k)} \geq \ldots$. For $s_{i_1}$, we have

$$\begin{aligned} f(d_{i_1}^{(k)}, r_{i_1}^{(k)}) &= \min\Big(g\big(d_{i_1}^{(k)}\big), g\big(d_1^{(k)}\big), r\Big) \\ &= \min\Big(g\big(d_1^{(k)}\big), r\Big) = f\big(d_1^{(k)}, r\big), \end{aligned} \quad (10)$$

where the second equality is from $d_{i_1}^{(k)} \leq d_1^{(k)}$ and the decreasing property of $g(\cdot)$. Similarly, we have $f(d_{i_l}^{(k)}, r_{i_l}^{(k)}) = f(d_1^{(k)}, r)$. Since for each service node $s_i$ there is always a path from $s_1$ to $s_i$, we have $U(d_i^{(k)}, r_i^{(k)}) = d_i^{(k)} f(d_i^{(k)}, r_i^{(k)}) = d_i^{(k)} f(d_1^{(k)}, r)$ and have proved the lemma. □

The assumption that $f(d, r) = \min(g(d), r)$ and $g(\cdot)$ is a positive and decreasing function has been substantiated by the careful OvS performance profiling in Sec. IV.

**Lemma 2.** *Let $D := \lceil (n+k-2)/k \rceil$. Under the same conditions of Lemma 1, we have*

$$\sum_{i=1}^{k} U\big(d_i^{(k)}, r_i^{(k)}\big) = (n+k-2) f(D, r), \quad (11)$$

*And for a given $k$, exactly $x = (n-2) \mod k$ elements of $d_1^{(k)}, \ldots, d_k^{(k)}$ must be $D$, and the remaining $k - x$ elements in the list must be $D - 1$.*

*Proof.* Since each of the $n-1$ receiving hypervisors must be served by one service node and the $k$ service nodes form a tree, we have

$$\begin{cases} d_1^1 + d_2^1 + \ldots + d_k^1 = n-1, \\ d_1^2 + \sum_{i=2}^{k} (d_i^2 + 1) = 2k-2. \end{cases} \quad (12)$$

Thus, we have

$$d_1 + \ldots + d_k = n + k - 2.$$

The same applies to $d_1^{(k)}, \ldots, d_k^{(k)}$. By Lemma 1, we have

$$\begin{aligned} \sum_{i=1}^{k} U\big(d_i^{(k)}, r_i^{(k)}\big) &= \Big(\sum_{i=1}^{k} d_i^{(k)}\Big) f(d_1^{(k)}, r) \\ &= (n+k-2) f(d_1^{(k)}, r), \end{aligned} \quad (13)$$

Therefore, $\sum_{i=1}^{k} U\big(d_i, r_i\big)$ is maximized when $f(d_1, r)$ is maximized over $d_1$. Since $f$ is non-increasing in $d_1$, the optimal $d_1^{(k)}$ should be the smallest possible $d_1$. Since $d_1 + \ldots + d_k = n + k - 2$ and by Lemma 1, $d_1$ is no smaller than $d_i$, for $i = 2, \ldots, k$, the smallest possible $d_1$ is simply $d_1^{(k)} = D = \lceil (n + k - 2)/k \rceil$. Therefore, substituting $d_1^{(k)}$ in (13), we have shown $\sum_{i=1}^{k} U(d_i^{(k)}, r_i^{(k)}) = (n+k-2) f(D, r)$.

When $d_1^{(k)} = D = \lceil (n+k-2)/k \rceil$, to maintain $\sum_{i=1}^{k} d_i^{(k)} = n+k-2$, $d_i^{(k)}$ for $i = 2, \ldots, k$ must be either $D$ or $D-1$. And there is only one unique degree distribution of $d_1^{(k)}, \ldots, d_k^{(k)}$ that can achieve this, which is described below: let $x = (n + k - 2) \bmod k = (n - 2) \mod k$; exactly $x$ elements of $d_1^{(k)}, \ldots, d_k^{(k)}$ must be $D$, and the remaining $k - x$ elements in the list must be $D - 1$. This is because $x = (n-2) \mod k$ is the unique nonnegative integer solution to

$$xD + (k - x)(D - 1) = n + k - 2 = \textstyle\sum_{i=1}^{k} d_i^{(k)}.$$

To see this, at this point, if $x$ is increased $xD + (k-x)(D-1)$ will exceed $n+k-2$ and if $x$ is decreased, $xD + (k-x)(D-1)$ will be less than $n + k - 2$. □

Based on the necessary conditions above, we can give a linear-time algorithm to solve problem (3).

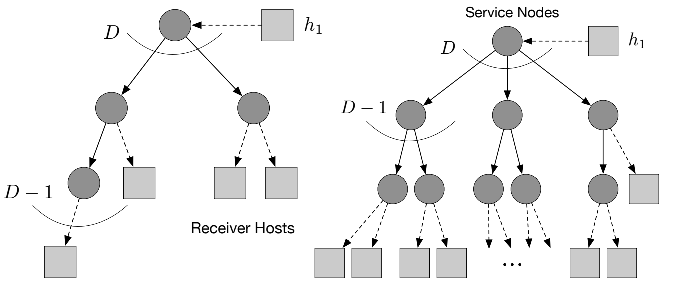
**Theorem 2.** *If the weights of service nodes $w_1 \leq \ldots \leq w_m$ are sorted in ascending order, there exists a linear time algorithm to find the optimal $k^*$ together with the optimal degree distributions $d_1^{(k^*)}, \ldots, d_{k^*}^{(k^*)}$.*

*Proof.* To find the best $k^*$, we only need to evaluate $\sum_{i=1}^{k} U\big(d_i^{(k)}, r_i^{(k)}\big) - \lambda \sum_{i=1}^{k} w_i = (n + k - 2) f(D, r) - \lambda \sum_{i=1}^{k} w_i$ for each $k \in \{1, \ldots, m\}$, which can be done in linear time. To see this, we first recursively construct a cumulative array $W_1 = w_1$ and $W_k = W_{k-1} + w_k$ for $k = 2, \ldots, m$ in linear time. Without loss of generality, we assume $m \leq n$, otherwise we can simply consider the $n$ service nodes with the least weights. Then, $k^* = \arg\max_k (n+k-2) f(D, r) - \lambda W_k$ can be found in linear time. And service nodes $1, \ldots, k^*$ (in the sorted list) will be used.

Once $k^*$ is found, the optimal degree distribution $d_1^{(k^*)}, \ldots, d_{k^*}^{(k^*)}$ can be determined from Lemma 2. The complexity of the entire procedure is $O(m)$. □

We are now ready to give a linear-time algorithm to solve problem (3) and construct the optimal multicast tree, which also has the minimum height, thus also achieving the minimum worst-case delay from the source to any receiving hypervisor. Once $k^*$ and the optimal degree distribution are found, we can form a corresponding tree by the following straightforward corollary. Under this degree distribution, we can minimize the tree height by letting nodes closer to the root have as many children as possible, as shown in the corollary:

**Corollary 3.** *Let $k^*$ be the number of service nodes found by Theorem 2. Let $D := \lceil (n + k^* - 2)/k^* \rceil$, then a* min-height multicast tree*, which is an optimal solution to problem* (3)*, is given by a $D$-ary tree as follows:*



(a) $n = 5$, $k = 4$, $D = 2$ (b) $n = 12$, $k = 9$, $D = 3$

Fig. 7. Min-height optimal multicast trees according to Corollary 3.

1) *Form an ordered list of all the nodes*
$$\mathcal{N} = \{h_1, s_1, \ldots, s_{k^*}, h_2, \ldots, h_n\}.$$
2) *Let $x := (n-2) \mod k$. From $s_2$ onward in $\mathcal{N}$, form $x$ groups of nodes, each group having $D$ nodes. Then form $k - x$ groups of nodes, each group having $D - 1$ nodes.*
3) *For $i = 1, \ldots, k$, assign group $i$ to be the children of $s_i$. Assign $s_1$ to be the only child of the source $h_1$.*

Then, we arrive at a set of simple closed-form forwarding rules that can be installed by the SDN controller into each node, described in Algorithm 1, where Lines 5-7 can always generate the desired optimal degree distribution satisfying Lemma 2. Apparently, once $k^*$ is found, it takes constant time for each service node to form its own forwarding rules. Thus, the time complexity of forwarding rule generation is also linear.

**Examples:** Fig. 7 shows two examples of min-height optimal trees according to Corollary 3, given $n$ and $k$ as the input. As we can observe, the intuition is that the out degrees of service nodes should be as evenly distributed as possible, and should be either $D$ or $D - 1$. That is, given that $k$ service nodes are selected, their out degree distribution is *unique*. Then, letting nodes closer to the root have a higher degree $D$ will reduce the worst number of hops from $h_1$ to any receiver, i.e., the height of the formed tree is only $O(\log_D(n + k))$.

## VII. Performance Evaluation

Based on the OvS performance traces obtained from Mininet-based profiling in Sec. IV, we perform extensive simulations to evaluate our proposed logical multicast routing schemes.

### A. Point-to-Point Multicast without Service Nodes

First, we show how the formed multicast topology affects the total throughput. Since point-to-point multicast is only useful for small sessions, we only test up to 14 hypervisors (each equipped with an OvS). We compare our optimal algorithm (a pipelined path graph) in Sec. VI-A with some typical multicast topologies: a star graph, a binary tree, and a ternary tree in which a node can have at most four children. From Fig. 8, we can see that the path graph has the largest total throughput, since there are at most two output streams for each OvS: one to the host attached to it and the other to the next OvS. The star graph has the lowest throughput, since the workload for

**Algorithm 1** Forwarding Rules for Min-Height Optimal Multicast with Service Nodes

```
1: Input: $\{s_1, \dots, s_m\}$, $\{h_1, \dots, h_n\}$.
2: Output: The forwarding rule for each service nodes.
3: Use the $O(m)$ procedure in Theorem 2 to get the optimal $k^*$ nodes with the least weights.
4: Arrange all the hypervisors and service nodes into an ordered list $h_1, s_1, \dots, s_{k^*}, h_2, \dots, h_n$, and assign them the node indices $0, 1, \dots, n + k^* - 1$.
5: $D := \lceil (n + k^* - 2)/k^* \rceil$.
6: Each service node $j \in \{1, \dots, k^*\}$ should forward packets to the nodes with indices given by
       $(j-1)D + 2 \sim jD + 1, \quad \text{if } 1 \le j \le x$
       $(j-1)(D-1) + x + 2 \sim (j-1)(D-1) + x + D, \quad \text{else},$
   where $x := (n-2) \mod k$.
7: Each node $y \in \{1, \dots, n + k^* - 1\}$ should receive packets from its parent, which has a node index $p(y)$ given by
       $p(y) = \begin{cases} 0, & \text{if } y = 1, \\ \lfloor \frac{y-2}{D} \rfloor + 1, & \text{if } 2 \le y \le xD + 1 \\ \lfloor \frac{y-(x+2)}{D-1} \rfloor + 1, & \text{if } xD + 2 \le y \le n + k^* - 1. \end{cases}$
```

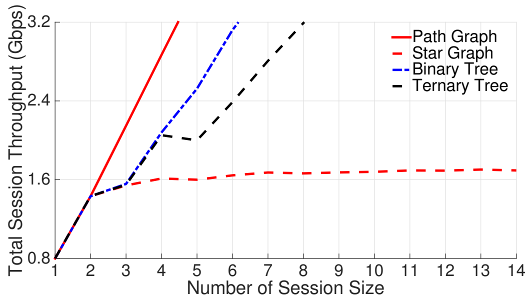


Fig. 8. Comparison of logical multicast with point-to-point tunnels.

the central switch is higher than all the others. Furthermore, the performance with binary and ternary trees is in between the optimal topology and the star topology. The simple message conveyed by Fig. 8 is that the larger the output degree of each switch, the less total throughput we will have, which conforms to our analysis.

## *B. Multicast with a Service Node Cluster*

We assume 100 service nodes (dedicated OvSes for traffic relaying) are available with some initial background flows, and simulate 100 multicast sessions that join the network consecutively. The multicast rate ranges from 0 to 100 Mbps, and the number of involved host hypervisors is uniformly distributed between 10 and 200. The service node selection and optimal multicast topology construction are performed for each session individually when it joins.

The weight of each service node $s_i$ used for load balancing in (3) is defined as $w_i = c(\omega_i) = -\alpha \log(1 - \omega_i/\Omega)$, where $\omega_i$ is the total existing workload (input rate) on $s_i$, including background flows and flows of previously joined multicast sessions; $\alpha > 0$ is a coefficient; and we set $\Omega = 32$ Gbps

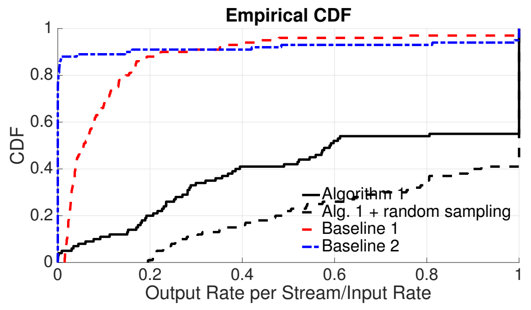


Fig. 9. CDF of the output rate per stream, normalized by the input rate in each session, which compares the throughput performance.

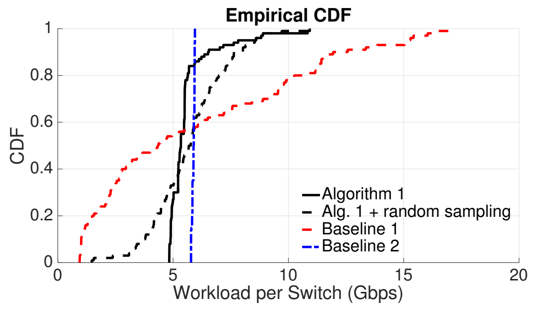


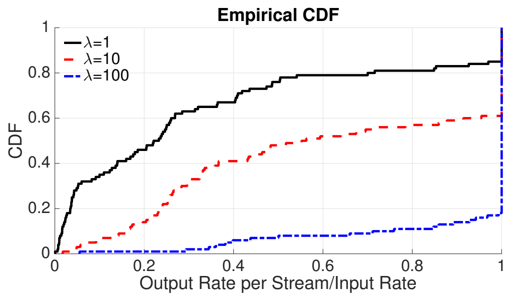
Fig. 10. CDF of the workload per service node under different schemes.

to be the highest input rate supported. The increasing convex nature of $c(\cdot)$ ensures that a higher existing workload implies a higher weight $w_i$, which increases faster when the existing workload is closer to the capacity of an OvS. This way, newly joined sessions will avoid selecting OvSes with higher weights to achieve load balancing.

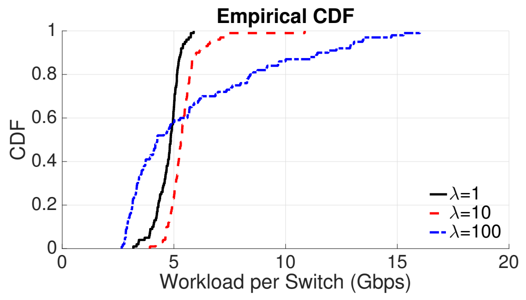
We compare Algorithm 1 with the following algorithms, including a variation of itself and two baseline algorithms:

- **Algorithm 1 with Random Sampling**: each session randomly selects 10 service nodes and use Algorithm 1 to form the optimal multicast topology with these selected service nodes;
- **Baseline 1 (NVP)**: each session finds the least loaded service node and use this single service node to multicast data to all destination hosts;
- **Baseline 2 (Binary Tree)**: each session finds the 50 least loaded service nodes, and among them selects all those with enough remaining capacity to accommodate the new session's input rate. Then, the selected service nodes are connected in a binary tree, where each service node handles two receiving hypervisors.

First, to evaluate throughput, we plot the output rate per stream normalized by the input rate for each session in Fig. 9. We can see that nearly half of the sessions controlled by Algorithm 1 can reach the full output/input ratio; the random sampling version further improves such a ratio to nearly 60%, though at the cost of degraded load balancing to be described below. In contrast, in Baseline 1 (the scheme used in NVP), a single service node may become a bottleneck for each session, leading to a throughput loss when there are many flows. Baseline 2, using binary trees as a heuristic solution, is even worse, since neither the number of used service nodes nor the



(a) CDF of the output rate per stream, normalized by the input rate in each session.



(b) CDF of the workload per service node.

Fig. 11. The impact of $\lambda$ on throughput performance and workload balancing among service nodes.

topology is optimized for each session.

Second, we show the workload per service node in Fig. 10 after all multicast sessions have been scheduled. Without surprise, as more service nodes are used in each session, the workload will be more balanced, with Baseline 2 that uses 50 service nodes per session achieving the best load balancing and Baseline 1 that uses 1 service node per session achieving the worst. Moreover, Algorithm 1 is better than its random sampling version.

In fact, with Algorithm 1, we can explicitly adjust the trade-off between the total throughput and load balancing, by tuning the parameter $\lambda$ in (3). It now becomes helpful to evaluate the impact of $\lambda$. A smaller $\lambda$ encourages Algorithm 1 to employ more service nodes per session, whereas a larger $\lambda$ discourages that. Therefore, a small $\lambda$ can achieve more balanced workloads and a large $\lambda$ can lead to higher throughput, as is confirmed in our evaluation results, shown in Fig. 11.

## VIII. Concluding Remarks

In this paper, we have proposed optimal logical multicast solutions for a virtualized datacenter network running software switches, where the bottleneck shifts from the link bandwidth to packet processing and flow cloning. Based on careful performance profiling on Open vSwitches, we formulate the logical multicast problem as Degree Supervised Graph problems, which is a new type of combinatorial optimization problems. As a highlight of this paper, we have proposed a linear-time algorithm to find the optimal multicast tree in the presence of service nodes, with joint considerations of throughput, load balancing and delay. Our solutions and algorithms can translate to simple SDN forwarding rules to be installed on each involved switch. By a wide margin, it outperforms the current industrial practices, such as NVP, in terms of both throughput and load balancing.

## References

[1] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data Center Network Virtualization: A Survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 2, pp. 909–928, 2013.

[2] "Amazon Virtual Private Cloud," http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html.

[3] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram *et al.*, "Network Virtualization in Multi-Tenant Datacenters," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[4] Open vSwitch, http://openvswitch.org.

[5] X. Li and M. J. Freedman, "Scaling IP Multicast on Datacenter Topologies," in *Proc. ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2013, pp. 61–72.

[6] R. Cohen, L. Lewin-Eytan, J. S. Naor *et al.*, "On the Effect of Forwarding Table Size on SDN Network Utilization," in *Proc. IEEE INFOCOM*, 2014, pp. 1734–1742.

[7] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance Characteristics of Virtual Switching," in *Proc. IEEE International Conference on Cloud Networking (CloudNet)*, 2014, pp. 120–125.

[8] M. Moshref, M. Yu, A. B. Sharma, and R. Govindan, "Scalable Rule Management for Data Centers." in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 157–170.

[9] R. Niranjan Mysore, G. Porter, and A. Vahdat, "FasTrak: Enabling Express Lanes in Multi-Tenant Data Centers," in *Proc. ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2013, pp. 139–150.

[10] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: a Scalable Multi-Tenant Network Architecture for Virtualized Datacenters," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, 2011, pp. 62–73.

[11] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009.

[12] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: a Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proc. ACM CoNEXT*, 2010.

[13] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "An Architecture for Wide-Area Multicast Routing," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 4, pp. 126–135, 1994.

[14] T. Ballardie, P. Francis, and J. Crowcroft, "Core Based Trees (CBT)," *ACM SIGCOMM Computer Communication Review*, vol. 23, no. 4, pp. 85–95, 1993.

[15] W.-K. Jia and L.-C. Wang, "A Unified Unicast and Multicast Routing and Forwarding Algorithm for Software-Defined Datacenter Networks," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2646–2657, 2013.

[16] A. Iyer, P. Kumar, and V. Mann, "Avalanche: Data Center Multicast Using Software Defined Networking," in *Proc. IEEE Conference on Communication Systems and Networks (COMSNETS)*, 2014, pp. 1–8.

[17] R. Sankar, "Data-Center Networking: What's Next Beyond 10 Gigabit Ethernet?" http://electronicdesign.com/communications/data-center-networking-what-s-next-beyond-10-gigabit-ethernet.

[18] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 242–253.

[19] "Mininet," http://mininet.org/.

[20] F. Hao, T. Lakshman, S. Mukherjee, and H. Song, "Enhancing Dynamic Cloud-Based Services Using Network Virtualization," in *Proc. the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, 2009, pp. 37–44.

[21] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem.* Elsevier, 1992.