Computer Communications 70 (2015) 1–14

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Solving multicast problem in cloud networks using overlay routing

Jessie Hui Wang [a,b,*], Jeffrey Cai [c], Jerry Lu [c], Kevin Yin [c], Jiahai Yang [a,b]

[a] Tsinghua National Laboratory of Information Science and Technology, China
[b] Network Research Center, Tsinghua University, China
[c] Cisco Systems, China

ARTICLE INFO

Article history:
Received 25 August 2014
Revised 28 May 2015
Accepted 30 May 2015
Available online 5 June 2015

Keywords:
Cloud
Multicast
Overlay routing

ABSTRACT

Currently, multicast in cloud networks without the support of underlay IP multicast relies on one-to-all replications, which wastes networking resources and may induce bottlenecks. In this paper, we point out this issue should be solved as an overlay routing problem and the special architecture of cloud networks should be fully exploited. Then we propose a solution which includes a SDN framework and an algorithm to construct a degree-constrained overlay multicast routing tree. Our simulations show that its performance is better than current solution. Moreover, it can deal with various multicast groups and it scales well with both group size and cloud size.

1. Introduction

In recent years, cloud networks have been more and more popular. In most of cloud networks, operators interconnect a number of physical or virtual layer 2 LANs through a layer 3 network to provide a single large scale flat LAN for cloud users. Conventionally, we call the physical network which includes physical machines and interconnections among them as "underlay network", and call the virtual network which includes virtual machines, virtual switches in hypervisors, and virtual interconnection among these virtual switches as "overlay network". Fig. 1 illustrates such a data center, where hundreds or thousands of computers are connected with each other physically, and each of these computers is virtualized as a virtual subnet which consists of a virtual switch in its hypervisor and a number of virtual machines.

Most of current cloud networks exploit encapsulation-based tunnel to construct the virtual overlay network and enable the communication between virtual machines in different virtual subnets through the physical network which connects the physical machines.

In the mechanisms based on encapsulation-based tunnel, tunnel endpoints, which might be virtual switches in hypervisors of physical machines, should learn and maintain a mapping between the underlay network and the overlay network, *i.e.*, the mapping between overlay addresses of virtual machines and underlay addresses of the physical machines where their corresponding virtual switches are located on. Based on these mappings, tunnel endpoints can enable communications between any virtual machines through encapsulation and de-encapsulation. As a result, a large virtual flat LAN was formed, wherein virtual machines are able to move from one virtual LAN to other virtual LANs smoothly and flexibly, and all network resources can be allocated dynamically to improve resource utilization efficiency.

This encapsulation and mapping method works well most of the time, but problems may occur in some cases. First, in some cloud networks, tunnel endpoints are only able to learn mappings of partial virtual machines. It is possible that the tunnel endpoint receives a packet but does not know the related mapping, *i.e.,* its corresponding underlay destination address. Then the virtual switch must broadcast the packet with unknown destination to all tunnel points in the cloud network. Although the broadcast scope can be reduced if the tenant ID and VxLAN ID are considered, it still generates many broadcast packets in the overlay network. Besides these broadcast packets generated by unicast packets with unknown destinations, cloud users may run some applications with multicast requirements, such as publish-subscribe services, web cache updates, or system monitoring *etc.*, and these applications may generate lots of multicast packets.

In case that the underlay network is multicast-enabled, these overlay broadcast/multicast packets can be mapped to underlay multicast groups and then the communication can be completed successfully. However, cloud operators usually would like to use low-end switches which may not support multicast functions, and enabling multicast in underlay IP networks may incur a lot of complexity on cloud networks.

Therefore, we have to solve the multicast problem in cloud networks without the support of underlay IP multicast. Current solution simply exploits multiple unicast connections to complete a multicast or broadcast. Logically the solution works as follows. After the virtual switch receives a packet with a unknown destination from

* Corresponding author. Tel.: +86 10 62603212.
E-mail address: hwang@cernet.edu.cn (J.H. Wang).

http://dx.doi.org/10.1016/j.comcom.2015.05.016
0140-3664/

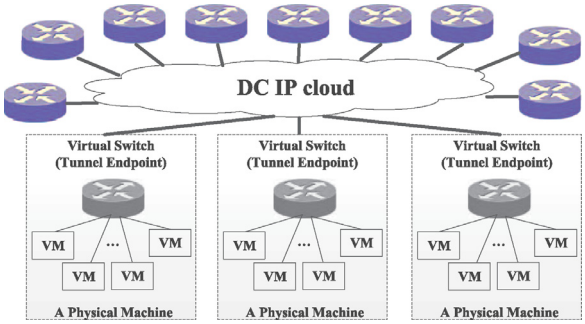

Fig. 1. Connect layer 2 virtual LANs through a core network.

its virtual machines, it sends request to the mapping server which can map the overlay broadcast or multicast to a list of underlay addresses. Then the switch can set up tunnels to each address in this list, then replicate, encapsulate and send out the packet to every destination.

This one-to-all replication method is a feasible and simple solution when the multicast group is small. But if the multicast group is with a lot of members, it would incur a lot of replication overhead to the source tunnel endpoint, and some receiving nodes would experience a very long delay because of the bottleneck for replicating. The other problem is that the outgoing links of the source node might be occupied by these replications, which may affect the performance of traffic flows of other applications on these links.

In this paper, we try to solve the multicast problem in cloud networks by proposing a degree-constrained overlay multicast scheme. We argue that the multicast problem should be solved at overlay instead of underlay, and we should take advantage of features of cloud networks to improve the efficiency of overlay multicast.

In fact, current one-to-all replication model can be viewed as a simple overlay multicast solution for cloud networks. In this simple model, the multicast distribution tree is a one-hop tree, *i.e.*, star topology. In this paper, we plan to design an efficient multicast distribution tree to replace this star distribution topology, and make sure that there is no single bottleneck for replication and traffic transmission. Furthermore, the distribution tree should be able to reduce the traffic overhead incurred on the cloud network, and control the maximum and average delay experienced by nodes in the multicast group.

The main contributions of this paper are listed as follows:

- We point out that the multicast issue in cloud networks should be solved at overlay (application) layer, and propose an overlay multicast routing framework.
- We point out that multicast problem is easier in cloud networks than in the Internet, and the construction of multicast routing tree should rely on special architecture of cloud networks instead of complex real-time measurement.
- We design an algorithm to construct a degree constrained overlay multicast routing tree. Our simulations demonstrate that our algorithm works better than current one-to-all replication model. We also show that it can deal with various multicast groups, and it scales well with group size and cloud size.

The remainder of this paper is organized as follows. In Section 2, we summarize related works on overlay multicast in the Internet, interconnection architectures and virtualization technologies of cloud networks, and recently proposed multicast solutions for cloud networks. In Section 3, we introduce some necessary preliminaries and propose the multicast routing issue in cloud networks. In Section 4, we propose our SDN framework to solve the multicast problem. Section 5 focuses on the algorithm to construct the routing tree. Our simulations and performance evaluation are shown in Section 6. Section 7 concludes the paper.

2. Related work

During the early stage of the Internet, whether the multicast service should be added to IP layer or implemented by end hosts at application layer had been discussed for a long time. In 1989, Deering argues that multicast should be implemented at the IP layer in [1] and IP multicast became the first significant feature that has been added to the IP layer. However, despite most routers today implement IP multicast, IP multicast has several drawbacks that have so far prevented the service from being widely deployed. On the other hand, numerous application layer multicast algorithms and protocols have been proposed [2,3].

Therefore, Chu *et. al.* [4] revisit the issue of whether multicast related functionality should be implemented at the IP layer or at the end systems (application layer) . The authors believe that End System Multicast has the potential to address most problems associated with IP multicast. Since all packets are transmitted as unicast packets, deployment may be accelerated. They conducted a detailed evaluation of the End System Multicast architecture, and concluded that End System Multicast is a promising architecture for enabling small and medium sized group communication applications on the Internet.

In 2014, Coras *et. al.* also tries to solve the multicast problem in the Internet by using overlay routing. The focus of this work is on inter-domain multicast issues, and intra-domain multicast is still solved by IP layer as in current Internet. In the proposed scheme, some routers should be upgraded to be overlay routing nodes, which is slightly different from the end system multicast where servers are involving routing. But both schemes are consistent with the concept of overlay routing.

The reason why End System Multicast is not suitable for large sized multicast group is that the system has to probe the underlay network to make sure that the overlay or application layer can exploit underlay links efficiently. As the group size increases, the probe overhead becomes unbearable, and the efficiency lost because of moving the function from underlay to overlay also increases.

The situation in cloud networks is different from the Internet. Internet is operated by many corporations and is heterogeneous in nature. A cloud network is always operated by one corporation and can be designed carefully, so that it is possible for the overlay to exploit underlay links efficiently based on prior knowledge without any probing or measurement. In this paper, we apply the idea of end system multicast to the multicast problem in cloud networks and take advantage of features of cloud networks to avoid the measurement overhead. The proposed scheme is especially efficient when the multicast group size is larger.

Numerous problems in conventional data centers have driven researchers to propose and design various data center architectures to solve these issues [5,6]. Data centers can be categorized mainly in two classes, the switch-centric and the server-centric [7]. The representatives of switch centric are Fat-Tree [8], VL2 [9] and Portland [10], while the representatives of server centric are BCube [11], DCell [12] and Ficonn [13]. In switch centric, switches are the dominant components for interconnection and routing whereas in server-centric, servers with multiple Network Interface Cards (NIC) exist and take part in routing and packet forwarding decisions. Currently, most cloud networks are switch-centric architecture, which is more like the Internet; while the server-centric architecture is closer to the idea of overlay routing. In this paper, we take Fat-Tree as an example of cloud network architecture and design the multicast distribution tree construction algorithm based on it. FAT-TREE leverages largely commodity Ethernet switches to support the full aggregate bandwidth of clusters consisting of tens of thousands of elements. The authors argue that appropriately architected and interconnected commodity switches may deliver more performance at less cost than available from todays higher-end solutions.

The above architectures solve the problem of how the physical network is constructed by low-end servers and low-end switches. Generally, cloud operators should virtualize this physical network, and provide a single large LAN to cloud users [14]. OTV [15] [16] and VxLAN [17] are two popular networking virtualization technologies. Both of them exploit encapsulation-based tunnels, although they are designed for different scenarios. With OTV, broadcast traffic among endpoints is reduced, but endpoints have to maintain a lot of mapping entries, therefore OTV is suitable for connecting several geographical distributed sites. On the other hand, with VxLAN endpoints do not need to maintain a lot of mappings, but they need to broadcast a lot of packets among endpoints, therefore VxLAN works well in a single large-sized site and is not suitable for connecting sites across the Internet. Our scheme in this paper can work with any virtualization technologies using encapsulation-based tunnels.

Recently, multicast in cloud networks has drawn attention of researchers. In [18], the authors propose a stateless source routing scheme, which is called as Code-Oriented eXplicit multicast (COXcast), to ensure that unified unicast and multicast packets can be delivered in the data center networks. The source node constructs the unicast path and multicast tree by encoding the corresponding output port bitmap of each intermediate node, and packets can be self-routed to multiple receivers without requiring header modification. In addition, intermediate switches/routers on the path/tree can be stateless. The authors argue that COXcast can simplify the deployment and management of a large number of medium-scale multicast groups, especially when applied to large-scale DCNs. Unfortunately, this scheme requires that all physical networking devices should support source routing, and would incur cost on all data packets due to the protocol header for source routing.

The authors of [19] also noticed the multicast problem in cloud networks. They focus on the scalability of multicast, and regard the forwarding table capacity of a single switch as the most important bottleneck. In their architecture, a network controller carefully partitions the multicast address space and assigns the partitions across switches in datacenters' multi-rooted tree networks. Local multicast addresses can be aggregated to further increase the number of groups in each pod. Therefore switches can cooperatively support a much larger number of multicast groups across the entire datacenter. Like [18], this scheme also requires the modification of all networking devices. Our work is similar as this work in terms of that both proposals depend on the structural properties of multi-rooted tree topologies, *e.g.* Fat-Tree. However, our proposal is an overlay solution, which is easier to be deployed since no modification is required on networking devices and all upgrades are implemented on end hosts.

The authors of [20] focused on the problem of survivable provisioning for multicast service oriented virtual network requests in cloud-based data centers. The authors proposed an efficient algorithm to map the overlay virtual multicast network provisioning request to proper underlay infrastructure, *i.e.,* the virtual machine placement problem. In the algorithm, the multicast virtual network is viewed as a tree-like topology with a source node at the root and destinationnodes at the leaves of the tree. Our paper tries to improve the efficiency of traffic delivery after the virtual network is provisioned, and it can work with the algorithm in [20] to deal with two different aspects of multicast virtual networks.

In [21], the authors also use SDN controller to solve the multicast problem in cloud networks. They propose to determine elephant and mice groups according to their traffic amounts and treats them separately to reduce the cost of multicast routing calculation. The solution proposed in [22] uses node-based bloom filter to encode the tree and design a provably loop-free Bloom Filter forwarding scheme based on the feature. It exploits the feature of future data center networks, which is close to this paper.

3. Preliminaries and problem statement

In the control plane of a cloud network shown as Fig. 1, there are two kinds of solutions. One is to deploy a encapsulation mapping distribution system to distribute and maintain mapping information, so that each edge device can retrieve the information of the corresponding underlay node for every overlay destination. OTV and LISP-MS solve the problem in this way. This is called as "explicit announcement", since each node should send explicit announcement to the mapping distribution system. The other solution is to depend self-learning, which is the same as Ethernet LAN. The edge device caches the mapping information it has learned. When it receives packets with unknown destination, it will broadcast to all interfaces. This is called "Self Leaning". Currently, some researchers argue that control plane solution and data plane solution should be decoupled [23] and can be proposed independently [24]. For example, NVGRE only gives data plane solution, and it states that it can work with any control plane solutions.

Due to the page limitation, we would not introduce how cloud networks deal with unicast packets in details. In the following subsections, we will focus on multicast packets in cloud networks.

3.1. Multicast packets in cloud networks

In cloud networks, the multicast packets may be generated because of the following reasons:

- A VM sends out multicast packets because of the requirement of applications of cloud users, *i.e.*, the broadcast packets are generated by overlay applications. We call them "overlay data multicast". This multicast is limited within the overlay multicast group, which is defined by the overlay application.
- A VM sends out broadcast packets because of operations of standard protocols, *e.g.* ARP. We call them "overlay control broadcast". This kind of broadcast is generally limited within an overlay VLAN, which is defined by cloud users.
- A tunnel endpoint sends out broadcast/multicast packets because it does not know the mapping information of a destination node. We call them "tunnel control broadcast". This kind of broadcast can be flooded to all edge devices in cloud networks, if no extra mechanism is deployed. The broadcast can be limited within the VLAN of the original packet, and VLAN is defined by cloud users.

Please note that one overlay VLAN can be assigned with a multicast group ID, and then overlay VLANs can be viewed the same as overlay multicast groups. In the following sections, we will not study each kind of multicast packets individually, and use "overlay multicast group" to include all of the above scenario requirements.

3.2. Current multicast solutions and challenges

Let us first review how cloud networks implement multicast without underlay IP multicast support currently. The implementation details might be slightly different in various cloud networks, but the conceptual model is the same.

Let IP_o^i denote the overlay address of a virtual machine v_i. Let us define the virtual switch which is located at the same physical machine as v_i as its "responsible switch", since all packets of the virtual machine have to be transferred by this virtual switch. We denote the underlay IP address of this physical machine as IP_j. We say that IP_j is the underlay *associated address* of IP_o^i.

Virtual switches, which serve as tunnel endpoints, should know the mapping (IP_o^i, IP_j) to encapsulate and deencapsulate unicast data packets. When one virtual switch receives a multicast data packet, it retrieves the multicast destination address, say IP_{om}^i, sends request to the group management server. Obviously, the overlay multicast destination should be mapped to a list of underlay IP addresses (IP_1,

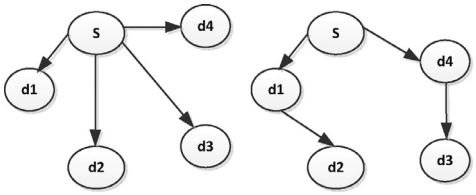

Fig. 2. Overlay multicast distribution tree.

$IP_2, \ldots, IP_n$), wherein IP_i is the underaly associated address of one virtual machine in the multicast group. With the one-to-all model, the source tunnel endpoint then sends out one copy of the data packet to each physical machine in the address list.

In OTV, the tasks of group management server are done by Adjacency Server (AS). Although it only maintains one list which includes all tunnel endpoints, the AS theoretically implements all necessary tasks, such as detection of node join and node leave, membership maintenance, receiving and responding request from tunnel endpoints.

The disadvantage of current solution is caused by the one-to-all model. We know that IP layer multicast exploits a tree structure to improve its efficiency, while one-to-all model is in fact a star topology. Therefore, the source node becomes a bottleneck, because of replication and heavy utilization of its outgoing links. It also incurs a lot of unnecessary load on cloud networks, which will be demonstrated in Section 6.

Therefore, we propose that we must design a more efficient topology to distribute multicast packets. Let us take a multicast group with four destination nodes as an example. In Fig. 2, the left part is using one-to-all replication model, while the right part shows a two-hop single-root tree. Although here we only show two distribution trees, we can find that there are more than ten possible distribution trees even for this small multicast group with only four destination nodes. Which one among these dozens of distribution trees is best for the multicast group and the cloud network? It is not easy to answer directly, especially when the multicast group is larger. This question is the key to improve the multicast efficiency of cloud networks. In Section 5, we will propose several metrics to evaluate the performance of a distribution tree and also design a heuristic algorithm to help cloud networks to construct better distribution trees.

4. Overlay multicast routing framework

In this paper, we study the problem stated in Section 3. Theoretically, it is constructing an routing overlay based on the physical network. Please note this routing overlay is different from the virtual overlay based on the same physical network. The virtual network is to provide cloud services for cloud users using virtualization techniques, while the routing overlay is to distributing multicast packets generated by the virtual overlay.

The construction of routing overlay is not a new problem in the Internet. Generally said, the efficiency of overlay routing depends on how much it knows about underlay information. For example, P2P application layer routing bring in a lot of problems because it only knows very limited information of underlay networks. The best way to learn about underlay networks is to conduct real time measurement about delay, available bandwidth and other information. However, it will bring in unbearable cost.

4.1. Taking advantage of features of fat-tree cloud networks

One hop in overlay network may involves different number of hops in underlay links. In other words, the distance of VMs are different. The key problem is to learn the underlay information as much as possible to guide the construction of overlay topology. We propose to take advantage of one significant feature of cloud networks. A cloud network is always designed and operated by a same company and can have rigid topology structure. In order to reduce the unbearable cost of measuring underlay information, we then propose the special addressing organization can help end nodes learn necessary underlay information without conducting measurement.

In this paper, we construct the routing overlay based on Fat-Tree architecture, whose design is shown in Fig. 3 [8]. Note in this Figure, only physical machines are shown, while Fig. 1 emphasizes virtual machines on these physical machines. Our framework can be applied on any architectures with rigid topology structure. In Fat-Tree cloud networks, we can easily infer the "distance" between two physical machines from their IP addresses. Therefore, the overlay can utilize this underlay information without measurement. Due to page limitation, we do not introduce Fat-Tree in details and only introduce two important properties.

According to the addressing rule of Fat-Tree, the IP address of one machine is 10.*x*.*y*.*z*, where *x* is the index of the pod where the machine is located, *y* is the index of the switch in pod_x, and *z* is the index of the host connecting with $switch_y$ in pod_x.

Assume there are two physical machines, with IP address $10.x_1.y_1.z_1$ and $10.x_2.y_2.z_2$. If $x_1 \neq x_2$, two machines are located in different pods, and their distance is 6, *i.e.*, there are three switches (core switch, aggregate switch, and edge switch) between them. We call it as "inter-pod" communication. If $x_1 = x_2$ and $y_1 \neq y_2$, two machines are connecting with different edge switches in a same pod, and their distance is 4, *i.e.*, there are two switches between them. We call it as "inter-switch" communication. If $x_1 = x_2$, $y_1 = y_2$ and $z_1 \neq$

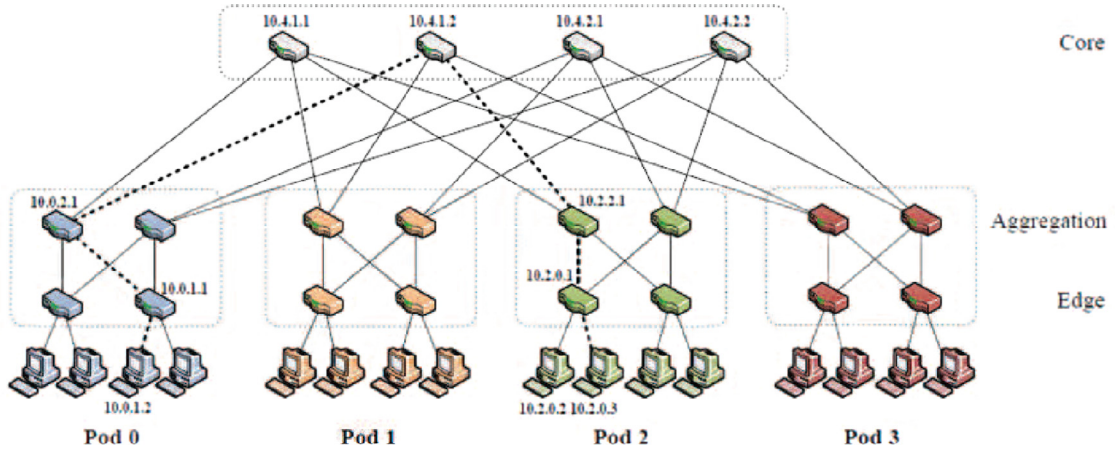

Fig. 3. Addressing and architecture of FAT-TREE cloud networks [8].

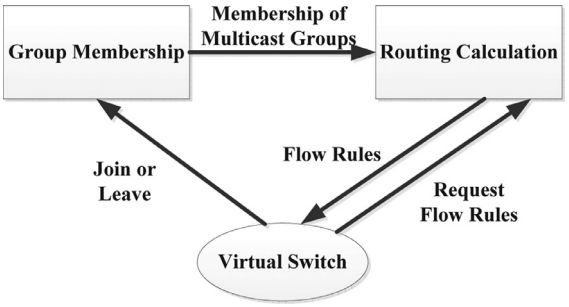

Fig. 4. Framework: software defined overlay.

z_2, they are connecting with a same edge switch, and their distance is 2, *i.e.*, only one edge switch is between them. We call it "local" communication.

Fat-Tree has a very good potential at spreading traffic on multiple links. Assume switches of the cloud network is k-port. There are $k^2/4$ possible paths between any two pods, and there are $k/2$ possible paths between any two edge switches in a same pod.

4.2. Software defined overlay

Our proposed framework is a software defined overlay. As a overlay, the cloud network does not require any modification on any physical/hardware switches. The overlay nodes are virtual/software switches in hypervisors of physical servers. They are implemented by software and are easy to enable new functions.

We exploit SDN architecture to implement and control the overlay. There should be two central modules: group membership management module and multicast routing calculation module. The two modules can be in a same central server and work closely with each other. The framework is shown in Fig. 4.

The group membership management module should able to detect events of node join and node leave. It maintains a list of IP addresses of members for each multicast group. When the membership of a group changes, it notifies the routing calculation module of a updated list, and the routing module calculates a new multicast routing tree.

The routing calculation module then translates the routing tree into flow rules of each overlay node. The flow rules are then sent down to virtual switches, which will take actions on multicast packets according to these flow rules. We can see that the routing calculation module is in fact a SDN controller.

5. Construction of multicast overlay distribution tree

In previous sections, we have described our basic idea and framework to solve the multicast problem in cloud networks. In this section, we would focus on the algorithm to construct the multicast overlay distribution tree.

5.1. Notations

For clear expression, we list some notations in Table 1. Based on these notations, we define metrics to measure the performance of a routing tree.

5.2. Local performance: delay

We use *delay* to each destination node to measure the user experience. The delay may be caused by two reasons: *transmission delay* and *replication delay*. In order to avoid measurement overhead, we simply assume each underlay (physical) link is with a transmission delay d_t, thus the delay between two overlay nodes is linear with their distance, assuming there is no replication needed. Let us assume that the

Table 1
Notations.

N	The number of destination nodes in the multicast group
M_i	The ith node ($i \in [1, N]$), the source node is defined as M_0
R_i^j	The jth overlay node on the path from the source to M_i
L_i	The length of overlay path to M_i
L	The depth of the tree, *i.e.*, $L = max(L_i)$
R_i	The overlay path to M_i, and $R_i = (R_i^1, \ldots, R_i^{L_i})$
d_i	The fan-out degree of M_i
$RI(i,j)$	The replication index from M_i to M_j, and $RI(i,j) \in [1, d_i]$
$Dis(i,j)$	The distance, *i.e.*, the number of underlay hops, from M_i to M_j

delay incurred by one replication action is d_r. So the delay to transmit a packet from the source node to a destination node M_i is

$$D_i = \sum_{j=0}^{L_i-1} \left(RI(R_i^j, R_i^{j+1}) \times d_r + Dis\left(R_i^j, R_i^{j+1}\right) \times d_t \right) \quad (1)$$

Let us define D_i^r and D_i^t as follows:

$$D_i^r = \sum_{j=0}^{L-1} RI\left(R_i^j, R_i^{j+1}\right)$$

and

$$D_i^t = \sum_{j=0}^{L-1} Dis(R_i^j, R_i^{j+1}) \quad (2)$$

Then we can rewrite the Eq. (1) as follows:

$$D_i = D_i^r \times d_r + D_i^t \times d_t \quad (3)$$

We can see that D_i^r is the number of replication actions during the whole period of transmitting a packet from the source node to the destination node, and D_i^t is the total number of underlay links on the overlay path.

Replication actions consume CPU and networking resource of the node. It is reasonable to assume that cloud operators would like to be fair to all nodes and set a maximum replication degree d, *i.e.*, $d_i < d$ for all $i \in [0, N]$. Then the minimum overlay path length can be calculated as

$$min(L) = \lceil \log_d(N(d-1) + d) - 1 \rceil. \quad (4)$$

It shows there is a tradeoff between d and L. The theoretic maximum upper bound of D_i^r is $d*L$. A simple numerical study can reveal that even when the group size is 2^{30}, the best d is 3, if we only want to minimize the longest replication delay.

In terms of minimizing transmission delay, according to the Eq. (2), we should try to minimize L as well as $Dis(R_i^j, R_i^{j+1})$.

In summary, we should keep these guidelines in mind:

Guidelines: *In order to minimize the maximum delay for nodes in the group to receive the packet, we should*

1. *Give a fixed d, we should minimize the depth of the routing tree, and the minimum depth can be calculated as Eq. (4).*
2. *Increasing d can help reduce maximum transmission delay, but it may not be helpful for minimizing maximum replication delay.*
3. *Local communications have a higher priority than inter-switch communications which are preferred to inter-pod communications. If not necessary, inter-pod overlay path should not be used.*

5.3. Networking load and load distribution

Cloud operators also would like to use its networking resource more efficiently. We define three global metrics to evaluate the utilization efficiency of a multicast tree.

The first metric is *networking load*, *i.e.*, the amount of traffic load incurred on the cloud network to transmit a multicast packet to all

destinations in the multicast group. We assume transmitting one packet on one underlay link consumes one unit of networking load. Then, the networking load is equal to the total number of underlay links the packet traverses to reach all destinations. Therefore, we have

$$T = \sum_{i=1}^{N} Dis\left(R_i^{L_i-1}, R_i^{L_i}\right). \tag{5}$$

Cloud operators would also like to see that the traffic load T is distributed evenly on more links. For example, it is obviously better that the traffic from pod_i to pod_j is distributed evenly on all usable $k^2/4$ paths, instead of on a single path. In this paper, we bring in the concept of information entropy to evaluate the degree of load balancing. The *entropy* is defined as

$$E = \sum_{i=1}^{G} \frac{T_i}{\sum_G T_i} \log_2 \frac{\sum_G T_i}{T_i}, \tag{6}$$

wherein G is the number of underlay links whose load is larger than zero, and T_i is the traffic load on the ith link. A larger E means that traffic flows are distributed on more links, or more evenly although the number of links does not increase. Cloud operators prefer a larger E.

In Fat-Tree cloud networks, underlay links can be classified into six groups: from machines to edge switches (local uplinks), from edge switches to aggregate switches (switch uplinks), from aggregate switches to core switches (pod uplinks), and corresponding local (switch/pod) downlinks. Links in a same category might be interchangeable, while links in different categories cannot be interchanged using traffic engineering techniques. Therefore, in this paper we also study the entropy of each category, as well as all links in total.

The third metric is *multicast capacity*, which is the maximum traffic rate from the source node to all destinations this cloud network can support. Mathematically, it is

$$C = \min_G \frac{C_i}{T_i} \tag{7}$$

5.4. Constrained overlay multicast strategy

The operator of one cloud network can set the maximum fan-out degree d according to the performance of physical machines in the cloud network. If the physical machine can replicate and send out packets very quickly, operators do not worry about replication delay and d can be set to a larger value to reduce transmission delay.

Our goal is to construct a multicast tree under the constraint of d. We call the problem *constrained overlay multicast tree construction algorithm*.

Since there are several metrics involved, and the possible solution space is very large, it is difficult to formulate the routing construction as an optimization problem and solve the problem symbolically. Therefore, in this paper, we will propose a *greedy heuristic* algorithm and then evaluate the algorithm by simulations. The heuristics we proposed are as follows:

1. *Minimize the depth of the multicast tree.*

Given a value of d, the minimum depth of a tree is determined by Eq. (4). In most cases, a shorter overlay path tends to have a smaller delay than a longer overlay path, so each node would like to be added to the routing tree as early as possible. We are also aware that it is not absolutely true, since a overlay hop may involve different number of underlay hops. The heuristics will be evaluated by simulations.

2. *If not necessary, a node will not send packet to nodes in other pods or other switches.*

This is based on the Guideline 2, because inter-pod communications would incur a longer delay and more traffic load than local communications. But how to determine whether it is necessary to use inter-pod or inter-switch communications? The third heuristic answers this question.

3. *Inter-pod or inter-switch communications would be used, if the depth of the multicast tree would be larger with only local communications.*

In order to understand this heuristic, let us first see an extreme case. Initially, all pods, except the one which the source node is located in, do not have any node that received the packet. Therefore, inter-pod and inter-switch communications are necessary. Otherwise, they will never receive the packet.

Now let us consider the following scenario. After one or more inter-pod and inter-switch communications, some nodes in a pod have received the packet. Let us consider a pod r. Assume that the number of nodes in this pod is n_r. These nodes can be at three status:

- Done nodes: the node has been in the multicast tree and has used up its replication degree, *i.e.*, has had parent and children in the tree. Let us define this status as *done* and the number of this kind of node is n_r^d.
- Sending nodes: the node has been in the multicast tree and is finding proper child nodes. Let us define this status as *sending* and the number of these nodes is n_r^s.
- Waiting nodes: the node has not been added to the multicast tree, *i.e.*, it does not have any parent node and is waiting for other nodes to select it as a child node. Let us define this status as *waiting*, and the number is n_r^w.

Obviously we have $n_r = n_r^d + n_r^s + n_r^w$.

We have known that the minimum depth of the tree is L according to Eq. (4). Let L_j denote the level of a done node or a sending node M_j in the tree. And the root of the tree, *i.e.*, the source node is at level 0.

Under the condition that the depth of the multicast tree should not exceed L, the maximum number of nodes that a sending node M_j can cover in its subtree is

$$\frac{d^{l-l_j+1}-1}{d-1} - 1.$$

Therefore, the number of destination nodes who can receive the packet from n_r^s sending nodes without any inter-pod communication is

$$n_r^c = \sum_{j=1}^{n_r^s} \left(\frac{d^{l-l_j+1}-1}{d-1} - 1 \right). \tag{8}$$

If $n_r^w > n_r^c$, it means this pod needs help from other pods. Otherwise, it means local sending nodes have been able to cover all waiting nodes in the pod, and this pod may help other pods. We still need to know how many child nodes can be selected from other pods by these sending nodes.

Our greedy algorithm is a bread-first search algorithm. So we can assume at the moment that sending nodes are finding child nodes, these sending nodes are located at a same level l_r. Their child nodes are at the level $l_r + 1$. The number of nodes one child node can cover is

$$\frac{d^{l-l_r}-1}{d-1} - 1.$$

Therefore, to make sure that at the next level this pod still does not need help from other pods, at least the sending nodes should select n_r^l child nodes from the local pod, and n_r^l is

$$n_r^l = \left\lceil \frac{n_r^w}{\frac{d^{l-l_r}-1}{d-1} - 1} \right\rceil. \tag{9}$$

So at most the n_r^s sending nodes can select n_r^e child nodes from other pods if other pods need help, and we have $n_r^e = n_r^s \times d - n_r^l$. Please note if other pods do not need help, the sending node will

select child nodes locally, *i.e.*, local preference. If all local nodes have been added to the tree, the sending node just gives up selecting more child nodes, *i.e.*, no inter-pod if not necessary, which means its fan-out degree is less than d.

Now we need to determine which pod should be selected first if inter-pod communications are necessary.

4. *When more than one pods need incoming links, the pod who needs most incoming links are selected first.*

If $n_r^e < 0$, it means the pod r needs help from other pods, and it needs $-n_r^e$ incoming links. When one pod finds it can help other pods, it will select to help the pod with the largest $-n_r^e$.

In the above heuristics, we mainly consider inter-pod communications. Inter-switch communications should be dealt similarly.

Based on these heuristics, we design an algorithm to construct the constrained overlay multicast tree.

Constrained multicast tree construction algorithm:
The SDN controller (routing calculation module) executes this algorithm to construct a multicast routing tree.

```
1.  initialize the source node as a sending node at level 0,
    and other nodes as waiting nodes;
2.  l_c = 0;
3.  termination_flag = FALSE;
4.  while (termination_flag == FALSE){
5.    calculate n^s and n^w of the cloud network;
6.    calculate the expected level l using n^s and n^w;
7.    calculate n_r^s, n_r^w, n_r^l, and n_r^e for all r;
8.    for each sending node m at level l_c {
9.      let i be its pod index and j be its switch index;
10.     if n_j^e <= 0, d_l = d;
11.     if n_j^e > 0, d_l = ⌈n_j^l / n_j^s⌉;
12.     let d_j be the number of waiting nodes in j;
13.     m selects min(d_l, d_j) random nodes in j as children;
14.     update related variables;
15.     d_e = d − min(d_l, d_j);
16.     if n_i^e <= 0 {
17.       while d_e > 0 {
18.         let j̄ = argmax_r(−n_r^e) for all switches in pod i;
19.         m selects a random node in pod i switch j̄;
20.         update related variables;
21.         d_e = d_e − 1;
22.       }
23.     }
24.     else {
25.       let ī = argmax_r(−n_r^e) for all pods;
26.       while d_e > 0 and n_ī^e < 0 {
27.         let j̄ = argmax_r(−n_r^e) for all switches in pod ī;
28.         m selects a random node in pod ī switch j̄;
29.         update related variables;
30.         d_e = d_e − 1;
31.         let ī = argmax_r(−n_r^e) for all pods;
32.       }
33.     }
34.     update related variables;
35.   }
36.   l_c = l_c + 1;
37.   if there is no waiting nodes
38.     termination_flag = TRUE;
39. }
```

In the algorithm, steps from 10 to 14 are selecting local child nodes; steps from 16 to 23 are selecting children from nodes that are connecting with different edge switches in the same pod; while steps 25 to 32 are selecting children from nodes in other pods.

6. Performance evaluation

In this section, we would conduct simulations to evaluate the performance of the algorithm proposed in this paper. The performance of a multicast group might be affected by the distribution of destination nodes in the cloud network. For example, a multicast group with all nodes in one pod may have different performance from a multicast group whose nodes are distributed evenly in all pods.

Therefore, in this section, we first propose a simple model to generate different multicast groups. Then, based on this model and performance metrics defined in Section 5, we compare our algorithm with current one-to-all replication model. We also evaluate the scalability property of our algorithm from various perspectives.

6.1. Multicast group generation model

One cloud application prefer to use nodes in one pod instead of nodes in different pods, because nodes in pod can communication with each other with short delay. Therefore, it is reasonable to assume that the distribution of nodes of a group in pods of a cloud network obeys Zipf's law, which is a discrete power law probability distribution. Mathematically, we have

$$p_i = \frac{1}{i^s \sum_{n=1}^{k} n^{-s}} \quad i \in [1, k]; \tag{10}$$

wherein p_i is the probability that a node is in pod i, s is the parameter of Zipf distribution which controls the extent of heavy-tail. Note that k is a parameter of the cloud network, and it is in fact the number of pods.

However, in one cloud network, the maximum number of nodes in one pod is $k^2/4$. It is possible that $N \times p_i$ exceeds the threshold. In case that all nodes in pod i have been destination nodes of the multicast group, we will run the Zipf distribution algorithm again and again, until a usable pod is found.

We assume that nodes are distributed evenly in one pod, *i.e.*, the expected number of nodes connecting with different switches in one pod is equal. The algorithm to generate a multicast group is described as follows.

Algorithm to generate a multicast group:
We run this algorithm to generate different multicast groups with N nodes for simulations

```
1.  for x = 1 : N {
2.    run Zipf distribution to find a pod i;
3.    while (pod i has no free node) {
4.      run Zipf distribution to find a pod i;
5.    }
6.    set node x to be in pod i;
7.    run uniform distribution to find a switch j;
8.    while ( (i, j) has no free node) {
9.      j = (j mod k) + 1;
10.   }
11.   set node x to be in pod i switch j;
12. }
```

Fig. 5 shows several multicast groups generated with different parameters. Roughly speaking, the left graph illustrates dense groups, where a lot of nodes in the cloud take part in the multicast group; while the right part illustrates sparse groups. $s = 0$ produces groups

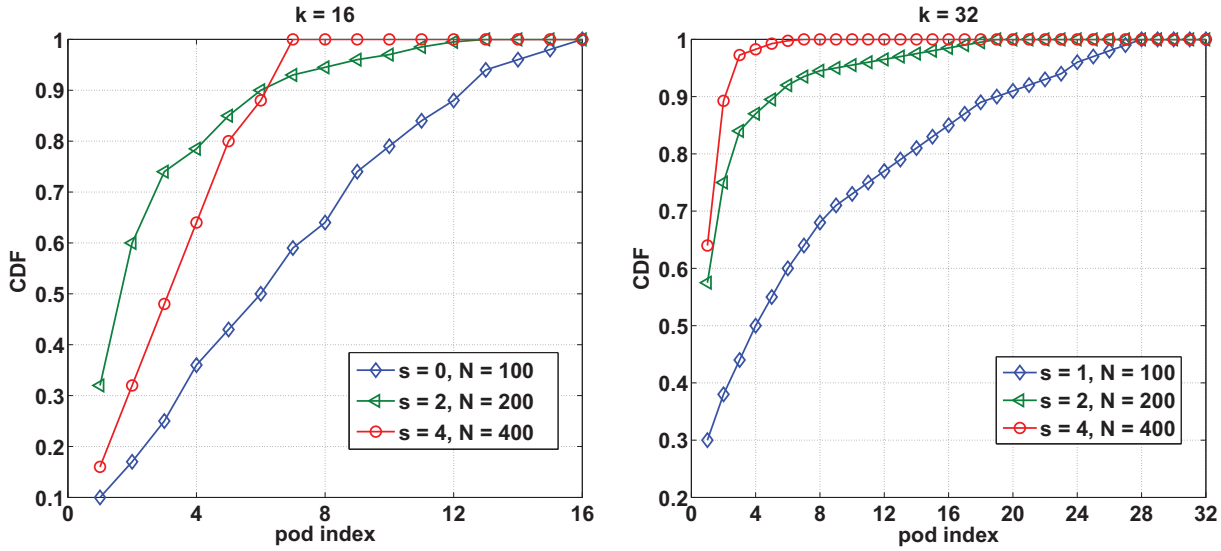

Fig. 5. Examples of multicast group with different parameters.

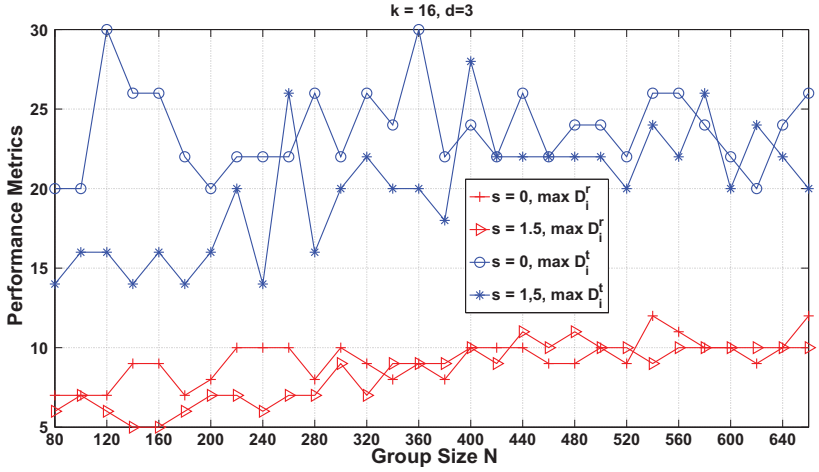

Fig. 6. Constrained algorithm: $\max D_i^r$ and $\max D_i^t$ as N increases.

whose nodes are distributed evenly. As s increases, the tail becomes longer and longer. Note that the line with circle markers in the left subgraph is roughly linear when $x <= 5$. It is because all nodes in the top five pods have been in the multicast group as s becomes larger.

6.2. One-to-all replication and constrained multicast tree

We consider a cloud network with $k = 16$, so it at most can have $k^3/4 = 1024$ nodes. In this cloud network, we generate multicast groups with different sizes (N from 80 to 660) and different s ($s = 0$ or $s = 1.5$). Then we run one-to-all algorithm and our algorithm ($d = 3$) for each multicast group and compare their performance.

1. Delay

Since the values of d_r and d_t are determined by the performance parameters of cloud networks, we cannot compare them directly. Therefore, here we would study the transmission delay and replication delay separately. With one-to-all routing algorithm, we can easily have

$$\max D_i^r = N \quad \max D_i^t = 6.$$

Fig. 6 shows the running result of the constrained overlay multicast routing algorithm. Here we do not present the average delay, which shows a same property with the maximum delay.

Therefore, we have the following observations:

Observation 1: *With the one-to-all routing, the maximum replication delay increases linearly with group size; while with the constrained multicast routing, the maximum replication delay does not change a lot as the group size increases.*

Observation 2: *With the one-to-all routing, the maximum transmission delay does not change with group size; while with the constrained multicast routing, the maximum transmission delay is also relatively stable, although it is a little longer than one-to-all algorithm.*

2. Average networking load incurred by one node

When the multicast group has more and more nodes, it is natural that the networking load may increase. Therefore, we plot T/N instead of T in Fig. 7.

Observation 3: *With the one-to-all routing, when the multicast group becomes larger, the average networking load incurred by one node also increases slowly; while with the constrained multicast routing, the average networking load incurred by one node decreases, which shows a good property of scalability.*

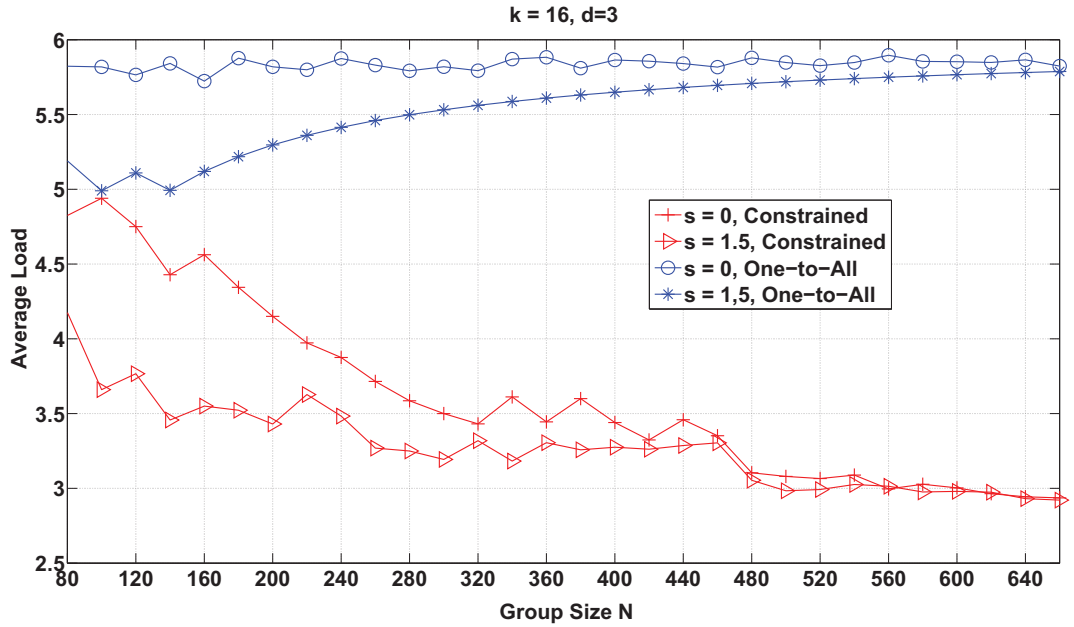

Fig. 7. $\frac{T}{N}$ as N increases.

3. Entropy of up-links and down-links

We study the load distribution problem under a favourable assumption, *i.e.*, each switch in the cloud network is using simple round robin multi-path packet forwarding. In other words, we assume traffic flows between two pods would be splitted evenly on all possible $k^2/4$ pod uplinks and pod downlinks, and we use similar assumptions for other links. After we calculate traffic load on each link, we can have the entropy of each category of links from Eq. (6), and plot them in Figs. 8 and 9.

With the one-to-all routing, the entropy of local uplinks is always 0, because only the source node incurs traffic load on local uplinks, and all these traffic flows must use the outgoing link of the source node. Similarly, only $k/2$ switch uplinks can be used and only $k^2/4$ pod links can be used. Therefore, the entropy of switch uplinks is always 3, and the entropy of pod uplinks 6. The entropies of these link types are much larger with the constrained multicast routing, which demonstrates a good property of load balancing.

Now let us look at pod and switch downlinks. Although the entropies under the constrained multicast routing is still larger than one-to-all routing, the differences between them become smaller. It is because of the good multi-path capability of Fat-Tree.

There is no difference on the entropy of local downlinks between two routing algorithms. This is because that the traffic load on local downlinks is determined by the distribution of destination nodes, and routing algorithms cannot help.

We summarize our observation as follows.

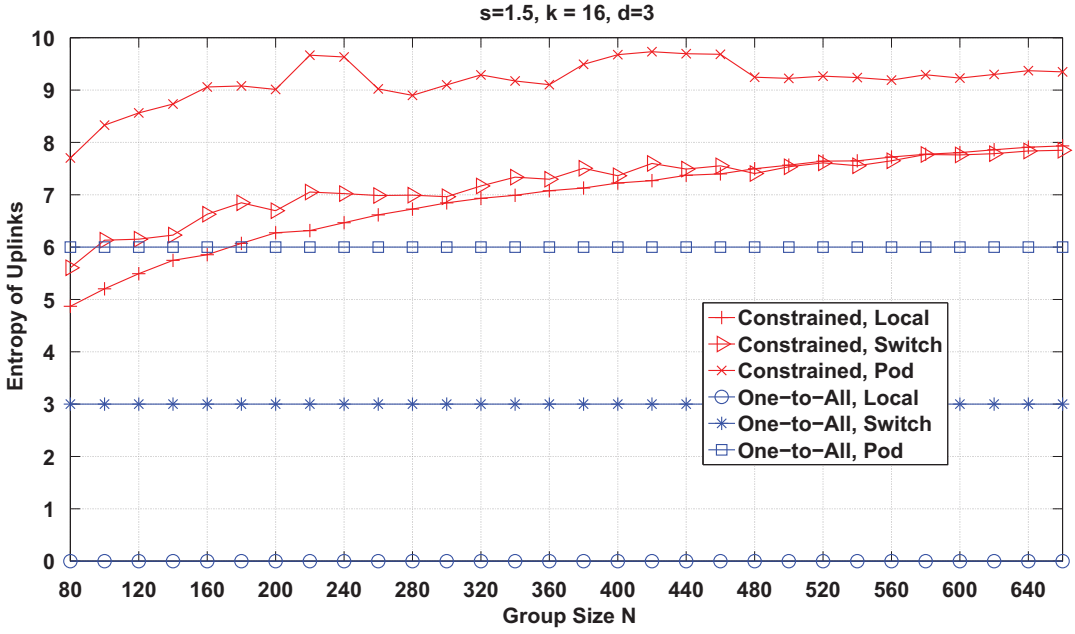

Fig. 8. Entropy of each category of uplinks.

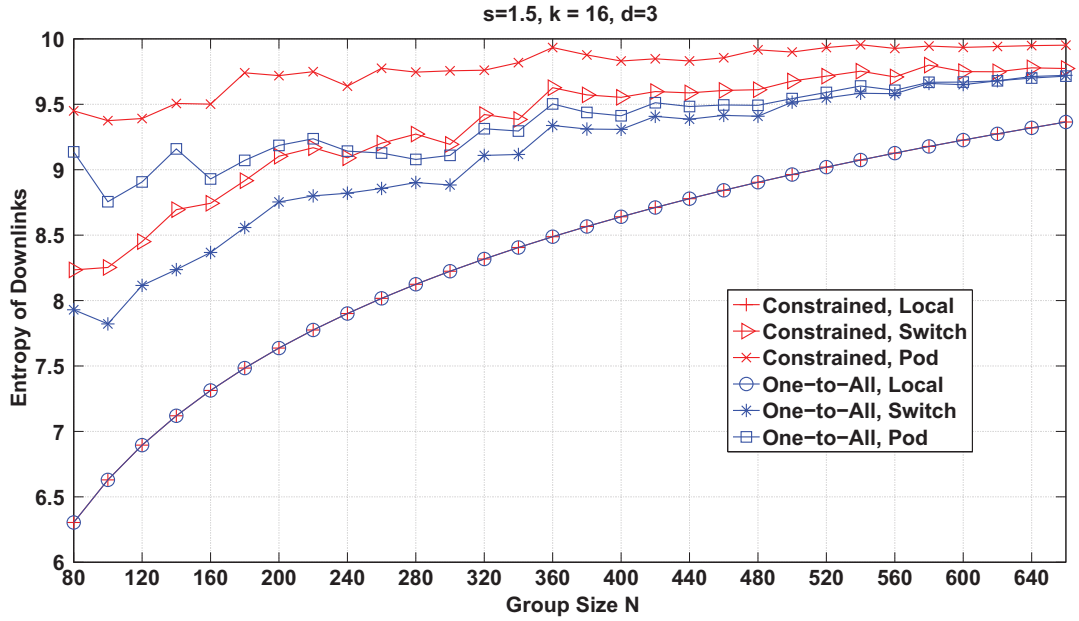

Fig. 9. Entropy of each category of downlinks.

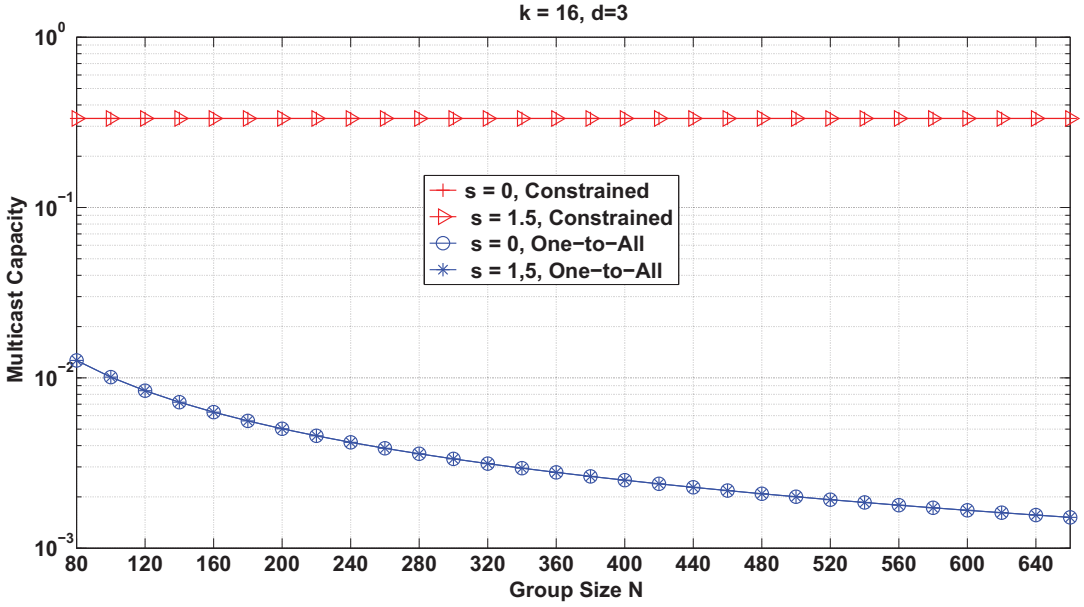

Fig. 10. Multicast capacity.

Observation 4: *The constrained multicast routing can spread traffic load on multiple links to avoid bottlenecks much better than the one-to-all routing, especially on uplinks.*

4. Multicast capacity

Fig. 10 shows the maximum traffic rate of multicast packets that the cloud network can support, assuming all links in the cloud network are with a capacity of 1.

Note that s does not affect C, so in the figure we can see only two lines.

Observation 5: *With the one-to-all routing, when the multicast group becomes larger, the multicast capacity decreases; while with the constrained multicast routing, the multicast capacity does not change, which shows a good property of scalability.*

6.3. Influences of parameters on constrained multicast routing

In the above subsection, we focus on the comparison of our constrained overlay multicast routing with current one-to-all routing algorithm. We can also see that the performance of our routing algorithm does not degrade a lot, even improve in terms of some metrics, as the multicast group becomes larger. It demonstrates that the constrained multicast routing algorithm is scalable with multicast group size N.

In this subsection, we conduct a further study on the performance of our algorithm as other parameters change.

1. Replication degree *d*

Figs. 11–13 plot the multicast performance when cloud operators set different replication degree d.

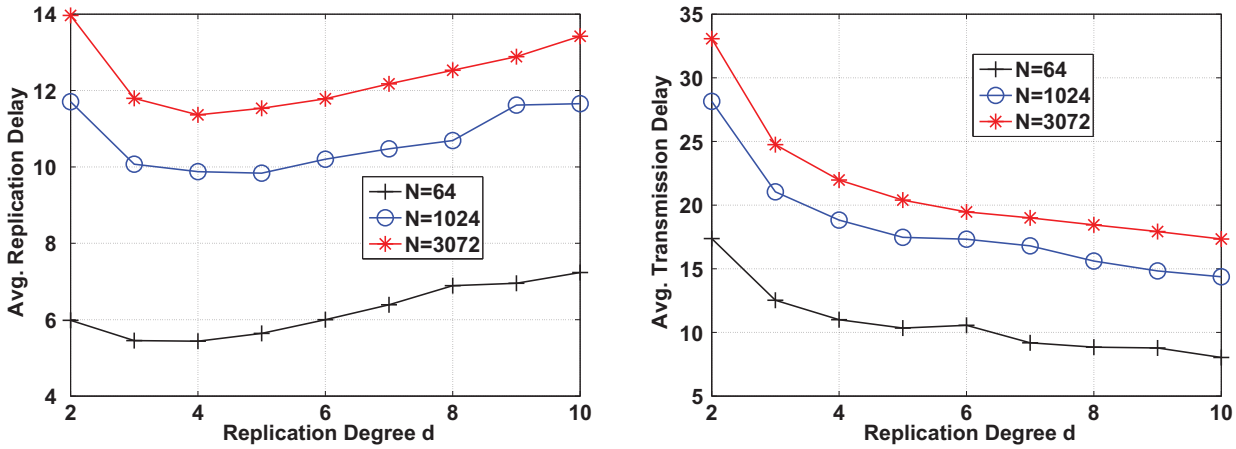

Fig. 11. Constrained multicast routing: D_i^r, D_i^t *vs. d* ($k = 32, s = 2$).

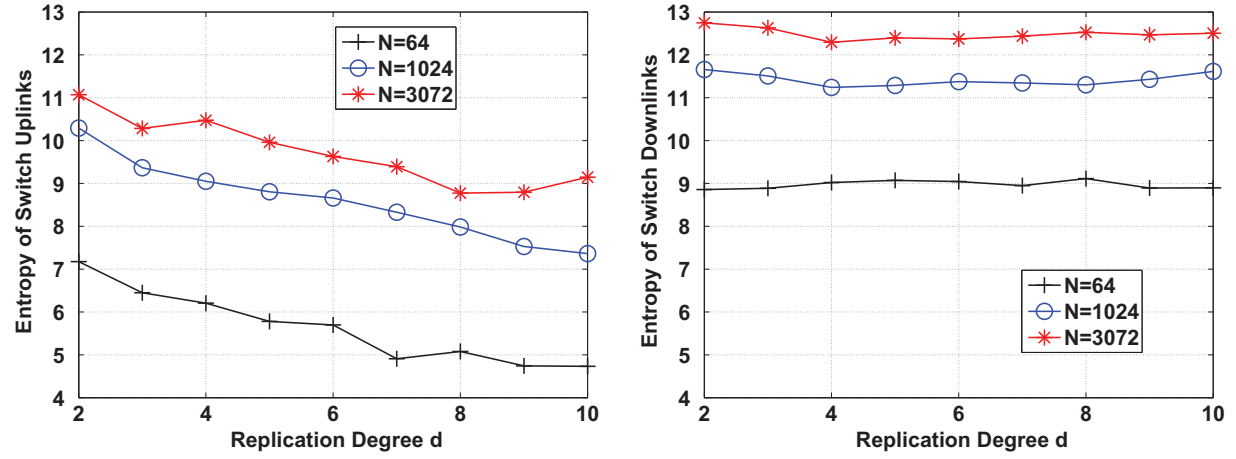

Fig. 12. Constrained multicast routing: *E vs. d* ($k = 32, s = 2$).

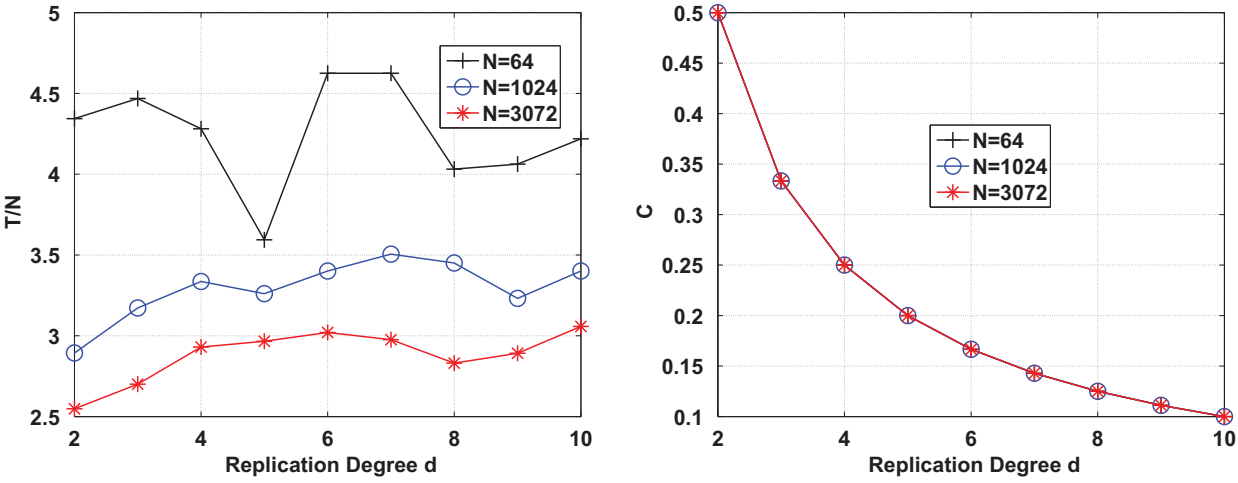

Fig. 13. Constrained multicast routing: *T*/*N*, *C vs. d* ($k = 32, s = 2$).

Although the average transmission delay keeps decreasing as *d* increases, almost all other metrics become worse. The average replication delay becomes longer after $d > 4$, the entropy of uplinks may decrease slightly, and the average networking load caused by one node may increase. Most importantly, the multicast capacity keeps decreasing. If the replication degree is too large, a node that replicates multicast packets will incur lots of traffic on its outgoing link to the edge switch and the link become a bottleneck.

Observation 6: *One cloud operator should set d according to performance parameters of the cloud network, e.g. transmission cost and replication cost. d should not be too large, and the recommended value is from* 2 *to* 6.

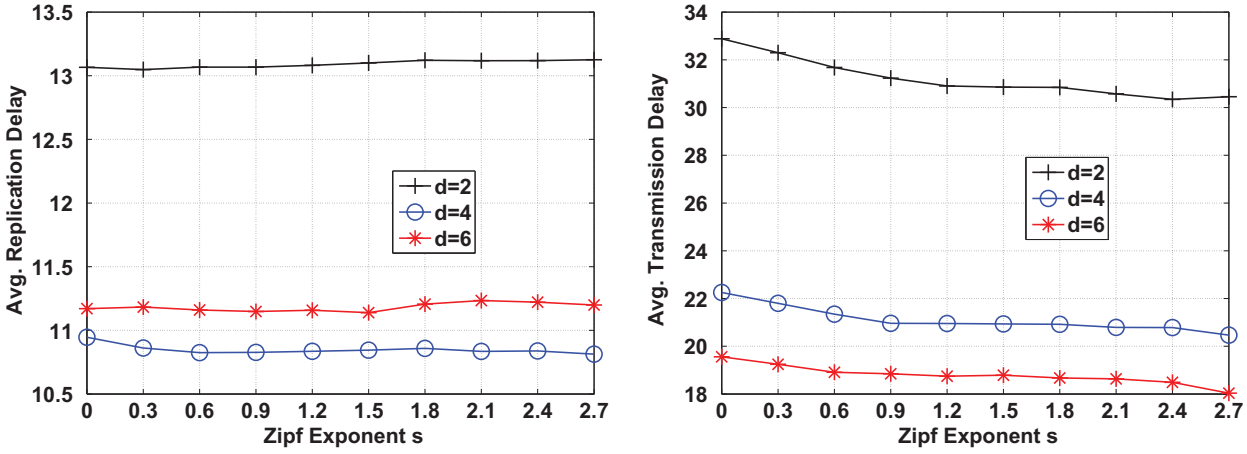

Fig. 14. Constrained multicast routing: D_i^r, D_i^t *vs. s* ($k = 32, N = 2048$).

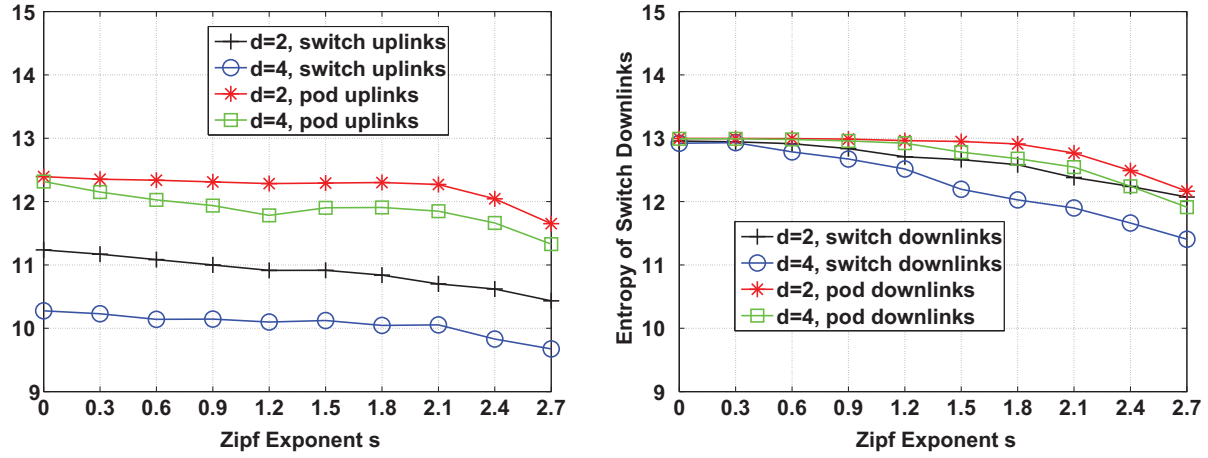

Fig. 15. Constrained multicast routing: *E vs. s* ($k = 32, N = 2048$).

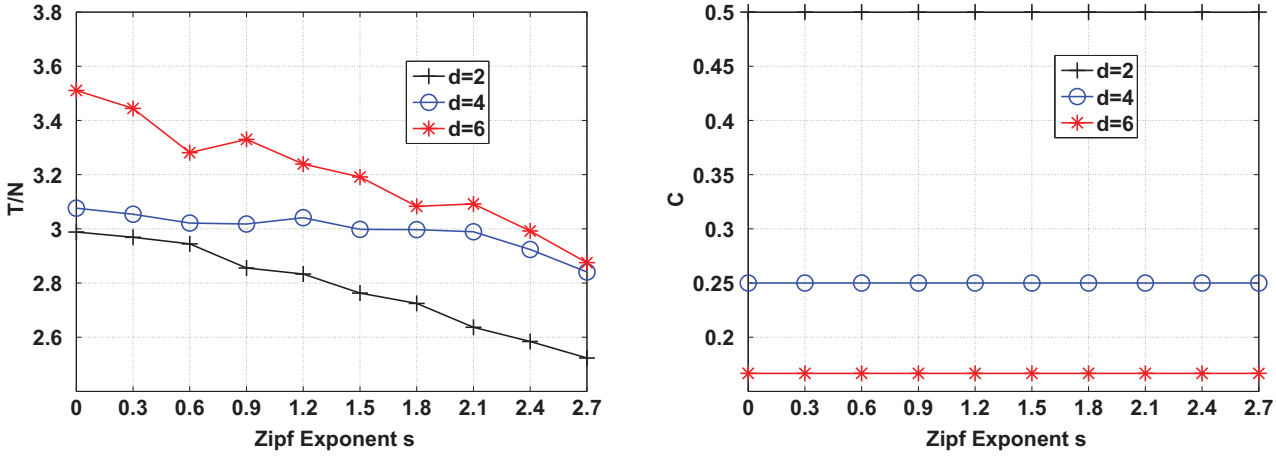

Fig. 16. Constrained multicast routing: *T*/*N*, *C vs. s* ($k = 32, N = 2048$).

2. **Zipf distribution exponent *s***

Figs. 14 –16 plot the performance for multicast groups with different nodes distribution.

In terms of the transmission delay and the average load incurred by one node, the multicast performance slightly improves when the distribution of multicast destinations are more heavy-tailed, *i.e.*, *s* increases. But the entropy of down links decrease, which reflects a worse load balance on these links. *s* does not have an obvious influence on other metrics.

Observation 7: *Roughly speaking, the performance of the constrained multicast routing is stable for multicast groups with different*

distributions of multicast destinations. When the distribution is more heavy-tailed, its transmission delay and average networking load slightly improve, but its load balance slightly degrades.

3. **Cloud size parameter *k***

Figs. 17 –19 plot the performance of our constrained multicast routing algorithm as k increases. Here we define the density of multicast group $p = N/(k^3/4)$, *i.e.*, the probability of a node in the cloud network to join the multicast group. It means that in the cases we study in these figures, the group size increases exponentially with k.

We can see that both average transmission delay and average replication delay increase with k. It is intuitively true since the group has much more nodes. Fortunately, we can see the increase rate is very slow, considering the exponentially increasing group size.

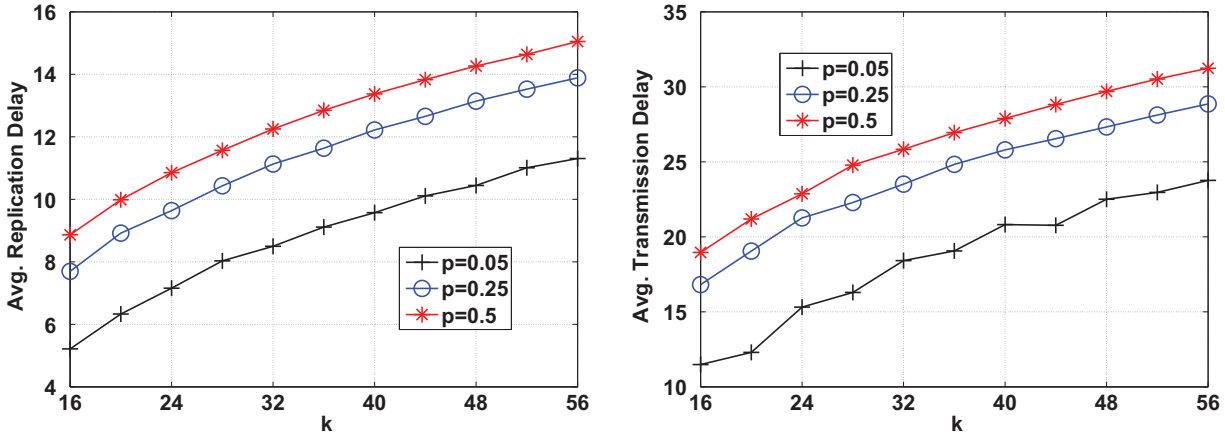

Fig. 17. Constrained multicast routing: D_i^r, D_i^t *vs.* k ($s = 2, d = 3$).

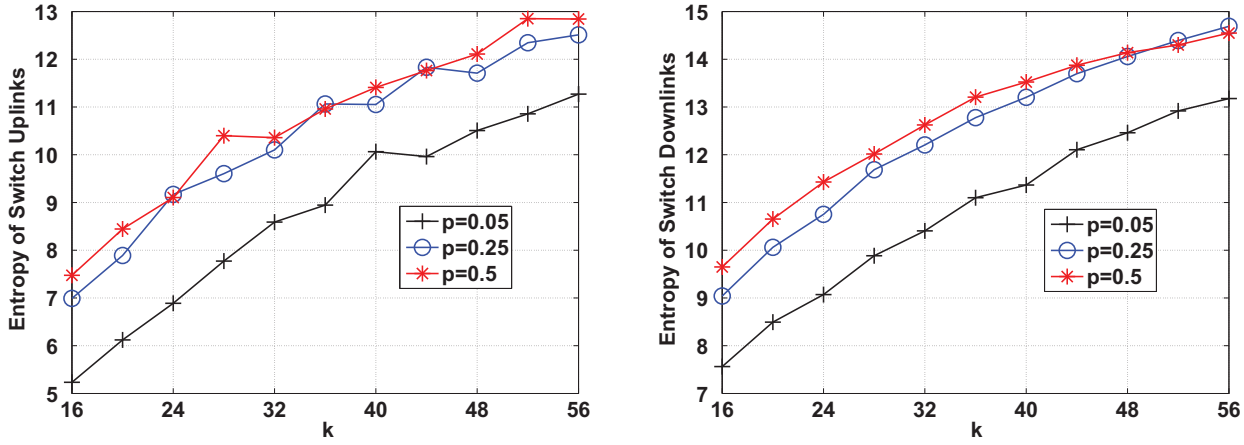

Fig. 18. Constrained multicast routing: E *vs.* k ($s = 2, d = 3$).

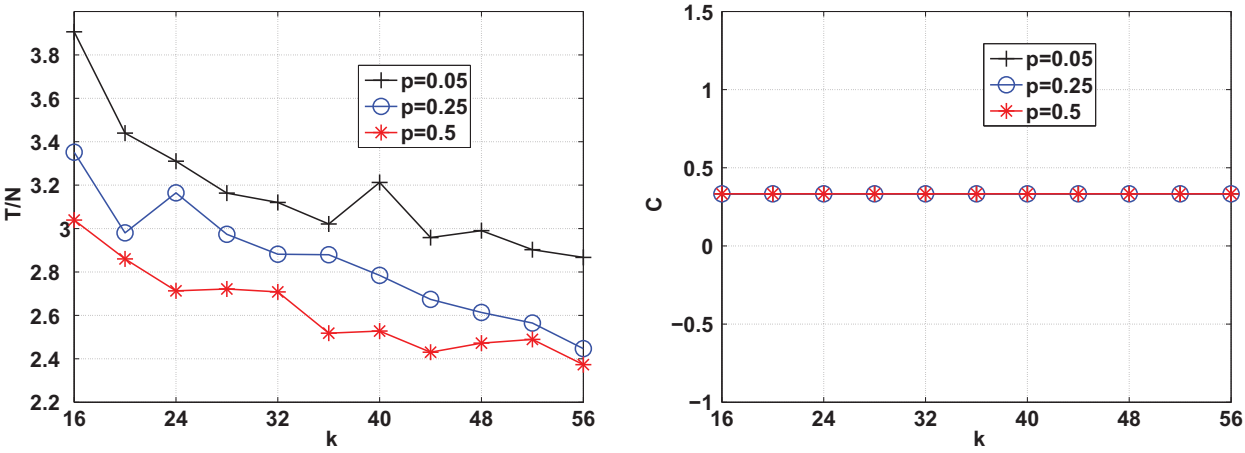

Fig. 19. Constrained multicast routing: T/N, C *vs.* k ($s = 2, d = 3$).

As k increases, the traffic flows on both uplinks and downlinks are more balanced, and the average load incurred by one node decreases. Both are good for the cloud network.

Observation 8: *As the cloud network extends, the constrained multicast routing algorithm works better in terms of load balance and the average load of one node. The delay increases, but its increase rate is much slower than the increase of group size.*

6.4. Summary of simulation results

We summarize all observations from our simulations as follows.

First, comparing to the one-to-all routing, our constrained multicast routing shows a better performance in terms of all four metrics defined in Section 5, *i.e.*, delay, networking load per node, traffic balance and multicast capacity.

Second, our constrained multicast routing shows a good property of scalability, as the multicast group size N increases or the cloud network size k extends.

Third, the performance of the constrained multicast routing is relatively stable for multicast groups with different distributions of multicast destinations.

Forth, the replication degree d (set by cloud operators) really affects the multicast performance. d should not be too large, and the recommended value is from 2 to 6.

7. Conclusion

In this paper, we solve the multicast problem as an overlay routing problem and take advantage of the special architecture of cloud networks to avoid unbearable costs to learn underlay information. Our solution is based on SDN framework which implements and controls the overlay. The SDN controller runs our algorithm to construct a degree-constrained overlay multicast routing tree. Then it translates the routing tree into flow rules of each overlay node. The flow rules are then sent down to virtual switches, which will take actions on multicast packets according to these flow rules.

Obviously, the algorithm to construct multicast trees has a significant influence on the performance of our solution. Our simulations show that its performance is better than current solution which relies on one-to-all replications. More importantly, it can deal with various multicast groups and it scales well with both group size and cloud size.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under grant no. 61202356 and 61170211, Tsinghua-Cisco Joint Research Lab under grant no. 2013300 0186, and Tsinghua University Initiative Scientific Research Program under grant no. 20121302141.

References

[1] S.E. Deering, D.R. Cheriton, Multicast routing in datagram internetworks and extended lans, ACM Trans. Comput. Syst. 8 (2) (1990) 85–110, doi:10.1145/78952.78953.

[2] M. Hosseini, D. Ahmed, S. Shirmohammadi, N. Georganas, A survey of application-layer multicast protocols, Commun. Surveys Tutorials IEEE 9 (3) (2007) 58–74, doi:10.1109/COMST.2007.4317616.

[3] P2p vs. ip multicast: Comparing approaches to iptv streaming based on tv channel popularity, Comput. Networks 55 (6) (2011) 1310–1325. http://dx.doi.org/10.1016/j.comnet.2010.12.020.

[4] Y. hua Chu, S.G. Rao, S. Seshan, H. Zhang, A Case for end-system multicast, IEEE J. Selected Areas Commun. Spec. Issue Network Support Multicast Commun. 20 (8) (2002).

[5] N. Maksic, A. Smiljanic, Improving utilization of data center networks, Commun. Mag. IEEE 51 (11) (2013) 32–38, doi:10.1109/MCOM.2013.6658649.

[6] Data center evolution: a tutorial on state of the art, issues, and challenges, Comput. Networks 53 (17) (2009) 2939–2965. Virtualized Data Centers. http://dx.doi.org/10.1016/j.comnet.2009.10.004.

[7] A. Hammadi, L. Mhamdi, Review: a survey on architectures and energy efficiency in data center networks, Comput. Commun. 40 (2014) 1–21, doi:10.1016/j.comcom.2013.11.005.

[8] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, in: SIGCOMM'08, ACM, New York, NY, USA, 2008, pp. 63–74, doi:10.1145/1402958.1402967.

[9] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, Vl2: A scalable and flexible data center network, SIGCOMM Comput. Commun. Rev. 39 (4) (2009) 51–62, doi:10.1145/1594977.1592576.

[10] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: A scalable fault-tolerant layer 2 data center network fabric, in: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, in: SIGCOMM'09, ACM, New York, NY, USA, 2009, pp. 39–50, doi:10.1145/1592568.1592575.

[11] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, Bcube: high performance, server-centric network architecture for modular data centers, in: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, in: SIGCOMM'09, ACM, New York, NY, USA, 2009, pp. 63–74, doi:10.1145/1592568.1592577.

[12] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, S. Lu, Dcell: a scalable and fault-tolerant network structure for data centers, in: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, in: SIGCOMM'08, ACM, New York, NY, USA, 2008, pp. 75–86, doi:10.1145/1402958.1402968.

[13] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, S. Lu, Ficonn: Using backup port for server interconnection in data centers, in: INFOCOM 2009, IEEE, 2009, pp. 2276–2285, doi:10.1109/INFCOM.2009.5062153.

[14] R. Jain, S. Paul, Network virtualization and software defined networking for cloud computing: a survey, Commun. Magazine, IEEE 51 (11) (2013) 24–31, doi:10.1109/MCOM.2013.6658648.

[15] United States Patent Application 20120176934, Overlay transport virtualization, http://www.freepatentsonline.com/y2012/0176934.html, 2012.

[16] draft-hasmit-otv 04, Overlay transport virtualization, http://tools.ietf.org/html/draft-hasmit-otv-04, 2013.

[17] draft-mahalingam-dutt-dcops-vxlan 09.txt, Vxlan: A framework for overlaying virtualized layer 2 networks over layer 3 networks, http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-09, 2014.

[18] W.K. Jia, A scalable multicast source routing architecture for data center networks, IEEE J. Select. Areas Commun. 32 (1) (2014) 116–123.

[19] X. Li, M.J. Freedman, Scaling ip multicast on datacenter topologies, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, in: CoNEXT'13, ACM, New York, NY, USA, 2013, pp. 61–72, doi:10.1145/2535372.2535380.

[20] D. Liao, G. Sun, V. Anand, H. Yu, Survivable provisioning for multicast service oriented virtual network requests in cloud-based data centers, Optical Switching and Networking 14 (0 (Part 3)) (2014) 260–273.SI: Optimization and Application in Converged Optical and Data Center Networks

[21] W. Cui, C. Qian, Dual-structure data center multicast using software defined networking, CoRR abs/1403.8065 (2014).

[22] D. Li, J. Yu, J. Yu, J. Wu, Exploring efficient and scalable multicast routing in future data center networks, in: INFOCOM, 2011 Proceedings IEEE, 2011, pp. 1368–1376, doi:10.1109/INFCOM.2011.5934921.

[23] B. Davie, A. Lambeth, Network virtualization, encapsulation, and stateless transport tunneling (stt), 2012, http://networkheresy.com/2012/03/04/network-virtualization-encapsulation-and-stateless-tcp-transport-stt/.

[24] draft-sridharan-virtualization-nvgre 07.txt, Nvgre: Network virtualization using generic routing encapsulation, https://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-07, 2015.