Prototype for Customized Multicast Services in Software Defined Networks

[1]Shengquan Liao, [2]Xiaoyan Hong, [1]Chunming Wu, [1]Bin Wang and [3]Ming Jiang
1. College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China
2. Department of Computer Science, the University of Alabama, Tuscaloosa 35487, USA
3. College of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China
shengquan-liao@163.com, hxy@cs.ua.edu, {wuchunming, bin_wang}@zju.edu.cn and jmzju@163.com

***Abstract*[1]— Multicast, transmitting packets from one source to a group of destinations, is a popular service in Internet and now in the datacenter network. However, a unified multicast algorithm cannot satisfy the diverse performance requirements from different users. Hence customized multicast services are proposed in software defined networks (SDN) in this paper. We tackle a few associated technical challenges, and introduce a prototype with OpenFlow1.3 and RYU controller. Our contributions are: (1) we present the modules in detail and explain why they are demanded to implement multicast services in SDN, independent of current Internet-based IGMP protocols and the multicast address; (2) our prototype could sliced multicast trees in the substrate network in accordance to different embedded multicast algorithms; (3) the last hop translation is proposed in access switches to guarantee the multicast flow to be forwarded through the calculated multicast tree and be consumed in end points. Simulation experiments evaluate these performances. And two applications (namely, multicast text transfer service and streaming media application) further validate the feasibility and operability of our customized prototype for multicast services in the real world.**

Keywords—Multicast Service; OpenFlow; SDN

I. INTRODUCTION

Multicast, transmitting packets from one source to a group of destinations, is a popular communication service in Internet and now also wildly used in datacenter center networks, supporting applications from file replication to cooperative computations [1]. Furthermore, different applications usually vary a lot in performance requirements, for example, a bank's replication system particularly emphasizes the security issue, while some cooperative computations concern about the delay constraint. A unified multicast algorithm (i.e. RPF algorithm in DVMRP) can hardly satisfy all of these diverse performance requirements. Taking the RPF algorithm as an example, the algorithm can calculate a multicast tree using link delay as its cost to address the delay constraint, but RPF can not directly used for security guarantee.

The increasing deployment of software defined networking technology (SDN) brings new opportunities in providing customized multicast services to satisfy the diversified requirements through the network programmability it enables [2]. A customized multicast service should allow users to load different multicast routing algorithms for optimizing their own service performances. For instance, the bank's multicast replication system can use a separate secure-enhanced multicast algorithm, while the cooperative computation uses RPF with delay constraint. In this case, two separate multicast protocols are needed. On the other hand, with SDN, we envision that a customized multicast service can support multiple performance requirements from different users by hosting different multicast protocols.

Usually, the multicast service requires the coordination of a set of distributed protocols, e.g. IGMP, DVMRP, and PIM. Typically, the hosts join or leave a multicast group, which is identified using a multicast address, by sending IGMP messages to its directly attached multicast router. Meanwhile, the router listens to IGMP messages, and periodically sends out queries to discover which groups are active or inactive on a particular subnet. Furthermore, DVMRP or PIM runs more complicated processes on routers (i.e. exchange multicast routing table, modify the multicast routing entries, etc.) to construct a source-based tree or group-based tree for sending multicast packets.

In the SDN paradigm, the controller is responsible for managing multicast groups, and calculating effective multicast trees; while the data plane performs packet forwarding according to the multicast trees. So the data plan keeps as simple as possible. The OpenFlow specification 1.3 supports multicast by issuing group commands to the substrate network to construct multicast trees. The multicast data packets can be forward to multiple ports by a single match in Flow Table.

Some works have been proposed for multicast applications using the SDN. However, their uses of IGMP overshadow the performance of the solutions for the reason that extra overhead, such as flow entries, is produced in OF-switches in order to maintain paths for the proposed protocols. MultiFlow [3] has shown that a direct Dijstra's algorithm can converge quicker than DVMPR in the periods of multicast tree construction, and hence the latency could be greatly decreased. But its IGMP Query packets need to be broadcasted in the substrate network so that clients interested could join. Hence, the switches should slice resources (i.e. TCAM, CPU or queue resources) to deal with these queries or reply packets. The scheme of fast rerouting controller [4] maintains a redundant multicast tree for efficient retransmissions in case of failures. The multicast addresses are replaced by the Ethernet address. It still uses IGMP packets to probe whether a group member is active. Finally, RYU [5] method to IGMP is letting a switch to operate as a querier for acknowledging the multicast memberships, and transfer the membership information to the multicast server via a server port.

[1] The correspondent author is Prof. Chunming Wu.

However, it is not a trivial task to implement IGMP in the relatively simple switch hardware in SDN [2]. The aforementioned early implementation in RYU ports IGMP to OF-switches, and the switches emulate the function of multicast routers. Overhead, thus, is a big issue. For example, the switches would load the forwarding entries to maintain upstream and downstream paths for all of the registered hosts and the multicast control packets in TCAM (Ternary Content Addressable Memory) to accommodate the periodic IGMP query messages. It is well known that the TCAM resource is extremely limited in OF-switches. Without enough available TCAM, in-coming packets may fail when looking up TCAM. The hit ratio and hence the forwarding efficiency will thus be reduced.

The next problem is the multicast address. In OpenFlow1.3, the multicast addresses are treated the same as the IP address, i.e., they can be matched in the flow table and no multicast semantic is attached. An action for a flow table entry can be replicating and forwarding. Hence, we believe that we can use a traditional IP address for multicasting the associated data flows. This would reduce the complexity in handling multicast address advertisement and dealing with the associated topological challenges. This, in turn, can further optimize multicast routing tree (i.e. aggregating FIB entries), and improve the scalability of multicast groups in datacenter network [6].

Thus, we develop a prototype for customized multicast services in SDN with OpenFlow1.3 on the top of RYU controller. The platform consists of mechanisms for handling membership, mechanisms for instantiating corresponding multicast routing algorithms in substrate network. Moreover, the platform has its own way of handling multicast forwarding, and performance requirements.

Our contributions are as follows:

(1) We present the modules in detail to implement multicast services in SDN, independent of current Internet-based IGMP protocols and the multicast address. To be more important, we have explained why our proposed modules are demanded in our prototype.

(2) To deal with different requirements in multicast services, an interface for different multicast tree construction algorithms is provided, and it could sliced a multicast tree in accordance to the embedded algorithms' outputs.

(3) The last hop translation is proposed in senders' and receivers' access switches to guarantee the multicast flow to be forwarded through calculated multicast trees and be consumed by end points. Furthermore, two practical applications (namely, multicast text transfer services and streaming media applications) are designed to verify its feasibility and operability in real world.

Our simulation experiments have validated that the flow control mechanism in receiver proxy could work well and our prototype can customize multicast services in accordance to the plug-in algorithms. Besides, the running of multicast text transfer and streaming media applications in our testbeds has confirmed us the feasibility and operability of the last hop translation.

The remainder of this paper is organized as follows: Section II describes the related work; the system architecture is presented in Section III; Section IV gives simulation experiments to evaluate the system performance; it is followed by the two applications to verify the feasibility and operability of our platform in Section V. Finally, we conclude our paper in Section VI.

II. Related Work

In this section, we conclude works that involves multicast services with OpenFlow. The recent published work (SDM [7]) has presented a software-defined multicast for streaming videos on a generic network layer. Although it has the similar architecture with our platform in SDNs, it does not present the technical challenges (i.e. how to deal with ARP requests) in detail and why their architecture is feasible and operable. Furthermore, there is no module in their prototype for customized performance optimization. Except for the SDM, other works (i.e. OFM [8], CastFlow [9], XVLAN [10] and Fast Rerouting Controller) all resolve to implement functions of multicasts in SDNs or Overlay Networks. OFM and CastFlow both have implemented multicasts in SDNs from a clean-state perspective. They do not clearly show what is utilized to label multicast groups and how the group members receive the multicast flow. As the same as SDM, they both do not design mechanisms to enhance performances. XVLAN is proposed to manage IP groups. Although it can act the functions of IGMP protocols, but it demands a dedicated external server to resolve addresses in edge switches. Besides, issues of the multicast tree construction and performance optimization have not been attached. Contrary to XVLAN, Fast Rerouting Controller focuses on programming two multicast trees for efficient retransmissions, but directly uses the IGMP protocol to manage group members. As has been mentioned before, the decentralized IGMP protocol would bring extra loads in switches.

III. The System Architecture of Prototype

The proposed multicast service supports source-based trees with address translations at the last hop. The architecture has four components and it builds on top of two default APIs of RYU, namely MPLS and Discovery, as shown in the red rectangle in Fig.1.

Overall (see Fig.1), **discovery** API offers the topology of underlying infrastructure to the controller. The multicast groups are maintained by **Register.** The controller calculates a multicast tree using **Plug-in Algorithm**. Since the multicast flows and unicast flows need to be distinguished in the substrate network (i.e. Class A and D addresses in IP protocol) and the multicast address is dropped in our prototypewe assign an unique **MPLS** labels for each multicast group. This method is largely inspired by the widespread utilization of MPLS to identify different flows in the traffic engineering. The **Forwarding Translation** is designed to deal with the address issue in the last hop. In addition, the **Receiver Proxy** acts the function of ARP proxy because sender or receivers would not

sent out UDP packets unless they receive some ARP replies in TCP/IP protocol stacks. In addition, it also controls the throughput in the multicast tree due to the bandwidth constraint from bottleneck links. With the coordination of all these modules, our prototype could provide customized multicast services. Also, it can be seen from Fig.1 that each switch is connected to the controller with a control channel (i.e. SSH link) that bridges the two layers. Except for the OpenFlow1.3 commands, the bridge also deliveries all the packets exchanged between the control and data planes.

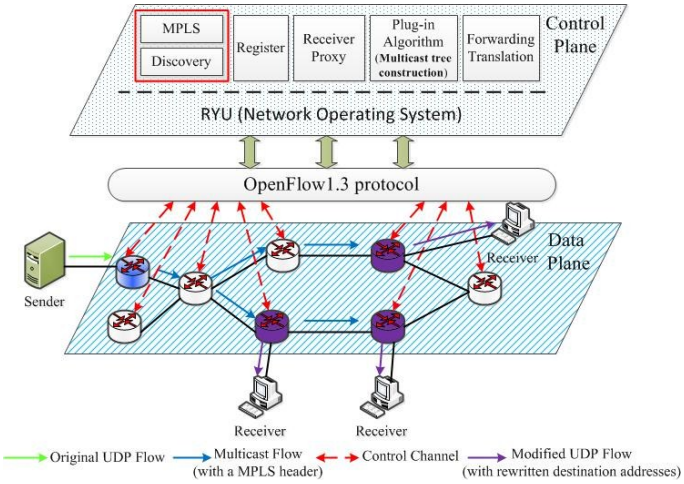


Fig. 1 System Architecture of Our Prototype

The major components are introduced as follows:

Register is responsible to manage multicast group members. When receivers join or leave the multicast group, this event will trigger the update of corresponding group members and the reconfiguration of the multicast tree. Whenever a sender is registered, this module will generate a unique **MPLS** label (named with its IP and port) which names the same as the ID of processes in Internet to guarantee its uniqueness. If the sender logs off, all its correspondent multicast trees and MPLS labels would be cancelled from the physical network infrastructure.

Receiver Proxy acts the functions of ARP proxy so that the receivers or senders can sent out their UDP packets. Also, the module controls the throughput in multicast tree due to the bandwidth constraints from the bottleneck links. The flow control can be implemented by adding Meter-Mod and Queue-Mod commands in the corresponding action set.

Plug-in Algorithm provides interfaces for users to load self-defined multicast tree construction algorithms (i.e. KMB [11] and Shortest Path Tree: SPT) on an up-to-date topology supplied by **Discovery** API. With outputs of this embedded algorithm, the controller generates flow tables, and sends them to switches. Hence, the substrate network could customize multicast tree in accordance to users' algorithms.

Forwarding Translation modifies flows in the last hop switches (i.e. blue and purple switches in Fig.1), and hence receivers could consume multicast flows. Initially, the original UDP flow from the sender would be encapsulated with a MPLS header in its access switch (i.e. the blue switch). Then the flow with a unique **MPLS** header would be forwarded to receivers through the calculated multicast paths. However, these packets would be dropped in the network adapter because of incorrect "Multicast Address" (MPLS). Hence the Forwarding Translation also works in receivers' access switches (i.e. purple switches) to reform these packets with new destination addresses (i.e. receivers' own IP or MAC address) so that group members can receive the multicast flow as normal one. Although these works seems resource-consuming, it just adds two actions in the action set mapped to the related multicast groups. Most importantly, this module does not increase the flow entries in OF-switches and do not need the coordination of end points.

Within our prototype, if a sender registers its role with a message "(Sender: IP: port)" encapsulated in UDP packet, the Register module would generate a correspondent MPLS label with the registered IP and port of the sender. Then, the controller broadcasts the MPLS to all of the hosts through control channels. After that, whenever receivers choose to join the multicast group by sending UDP message "(Join: IP: MPLS)", the controller calls plug-in multicast construction algorithm to calculate paths for multicast services, and then ports these calculated paths to OF-switches. When the sender receives the signal that the building tree has been finished, it would send multicast packets. Processed by the Receiver Proxy and Forwarding Translation, the adjusted flow will be forwarded through the instantiated multicast tree as the blue paths in Fig.1, and received by registered group members. In addition, the receiver's leave messages "(Leave: IP: MPLS)" would trigger the reconfiguration of the corresponding multicast tree. The whole procedure can be seen in Fig.2.

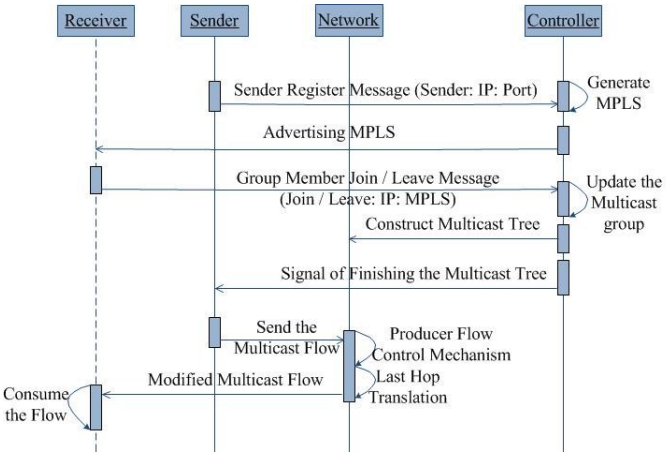


Fig.2 The sequence diagraph of our prototype

IV. PERFORMANCE EVALUATION

In this section, we first evaluate the efficiency of the flow control mechanism in Receiver Proxy, and then validate that our prototype can customize multicast services in accordance to the embedded plug-in algorithms.

A. Simulation Environments

We use Mininet to generate the backbone of *Internet2* [12] (Fig.3) as our simulation topology. In the topology, each switch is connected with two hosts numbering *2*dpid-1* and *2*dpid*, respectively. The *dpid* is the *ID* of OF-switches in the

substrate network, and hosts are not shown in Fig.3 due to RYU GUI tools.

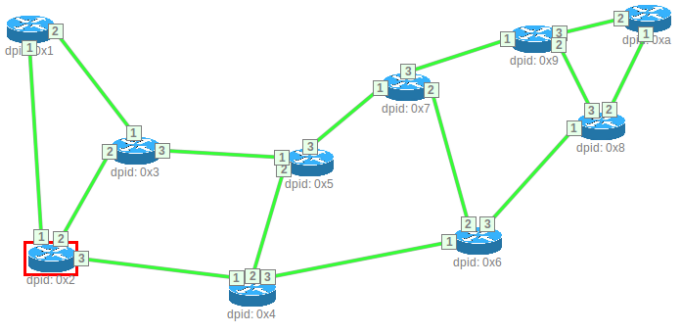


Fig.3 Simulation topology

To test our flow control mechanism in Receiver Proxy, we have defined a multicast group where h4 is the sender and {h1, h6, h7, h13, h20} is the receiver set. Then we load an original traffic in h4 with different thresholds, and observe the adjusted flows in *WireShark*.

Next, a Poisson process with an average 5 requests per time unit is introduced to simulate a dynamic multicast application scenario, where each request is composed with a sender and a receiver set whose size is randomly picked from the range of 2 to 10. In the simulation, we randomly generate 2500 groups of such multicast requests, and map them into the substrate network as Fig.3. Hence, we can obtain service performances in 500 time units with three comparisons (MultiFlow, SPT and KMB). MultiFlow use a Dijstra's algorithm to program a multicast tree, and map the logical tree in substrate network with OpenFlow1.0 commands. After that, we replace the OpenFlow1.0 with OpenFlow1.3, and then rename the MultiFlow as SPT. Furthermore, KMB is a scheme which utilizes KMB to calculate multicast trees and instantiates these trees via OpenFlow1.3 commands.

B. Metrics

We have designed three metrics (network resource consumption, average delay and standard deviation of delay) to show that different multicast algorithms vary a lot in service performances. The network resource consumption is the biggest concern of ISPs, while delay and its standard deviation are important parameters for users to reflect the service performances.

1) Network resource consumption

Our resource consumption is calculated with the following equation [13]:

$$c(G^m(t)) = 0.5 * \mathrm{CPU}(n^m) + 0.5 * \mathrm{Link}(l^m)$$

Here, for each request $G^m(t)$, m = 1,…2500; the network resource is calculated as a weighted sum over the nodes (n^m) and links (l^m) of the mapped multicast tree in substrate network. If the unit of allocated resource (i.e. CPU, TCAM) in the OF-switch to maintain a flow entry is assigned to 1, $\mathrm{CPU}(n^m)$ equals to the total number of switches within the multicast tree. Likely, setting the unit of link bandwidth to carry a multicast flow as 1, we can calculate $\mathrm{Link}(l^m)$ by summing up the number of ports of OF-switches involved in the multicast tree.

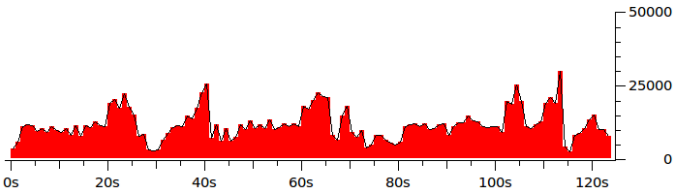
2) Delay and its Standard Deviation

The average delay is calculated as the total delays of multicast paths over the size of the receiver set. This parameter can roughly evaluate the efficiency of multicast algorithms. Standard deviation of delay is to measure the latency differences of receivers, which can show the service performance to some extent.

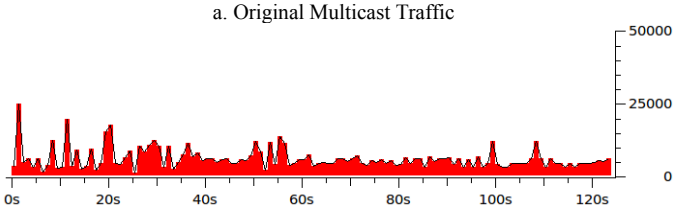
C. Simulation Results

1) Producer Flow Control Mechanism

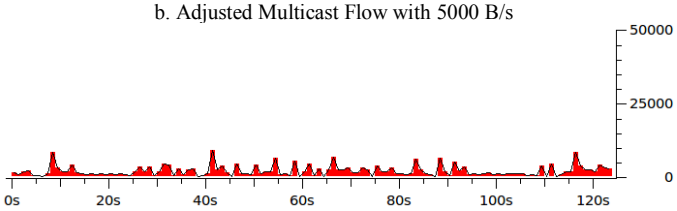
As had been mentioned before (see Sec. III), Receiver Proxy has an additional function for the flow control. To validate the efficiency of this mechanism, we load the sender h4 the original traffic given in Fig.4a, graphed to show the instantaneous throughput (Bytes/second) when time (in second) goes by. Then we apply different thresholds to control the rate of the multicast flow, and the results are shown in Fig.4b and Fig.4c. It can be clearly seen that the traffic flows in Fig.4b and Fig.4c with thresholds to be 5000 B/s and 1000 B/s respectively do not exceed our predefined values in the most interval of our time window (120s). Furthermore, Fig 4b has higher throughput than Fig 4c. That is majorly because the flow rate threshold in Fig.4b (5000 B/s) is bigger than that of Fig.4c (1000 B/s). The results suggest that our producer flow control mechanism in Receiver Proxy can effectively adjust the traffic in the multicast tree with an imported threshold.



a. Original Multicast Traffic



b. Adjusted Multicast Flow with 5000 B/s



c. Adjusted Multicast Flow with 1000 B/s

Fig.4 Results of flow control

2) Performance Analysis

Fig.5 illustrates that different multicast algorithms show great differences in network resource consumption even for the same request set. Although MultiFlow shares the same

multicast tree as SPT, it makes the switches manage more MPLS labels (more entries in the flow table) due to its independent identities for each source-destination path. That contributes to more resource consumption for MultiFlow. Furthermore, KMB optimizes the multicast tree with two iterated Kruskal's algorithm, and its tree would have less nodes and links. Therefore, KMB demands least network resources than MultiFlow and SPT to support the same request set.

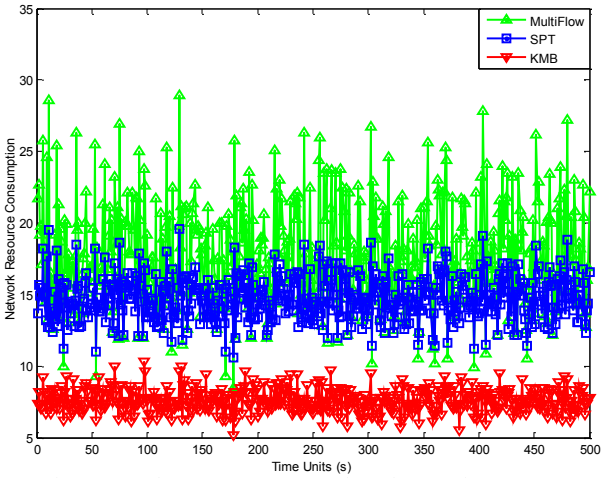


Fig.5 Network Resource Consumption of Dynamic Requests

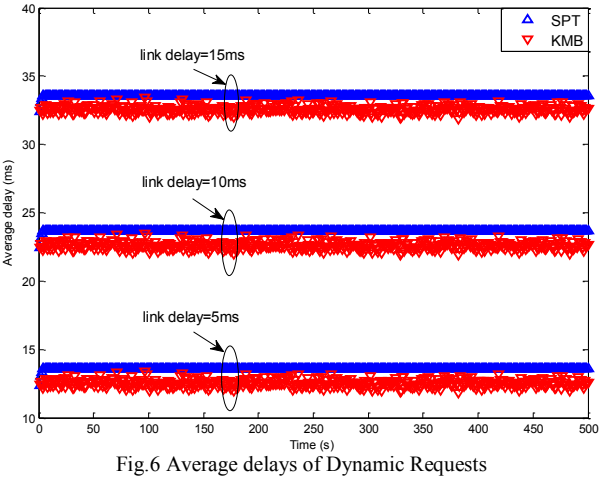


Fig.6 Average delays of Dynamic Requests

Because MultiFlow and SPT program the same multicast tree, they share the same value in average delay and standard deviation. Thus, we only compare SPT with KMB. Fig.6 illustrates that the average delay of KMB is always smaller than that of SPT in the cases for the link delay being 5ms, 10ms and 15ms, respectively. In contrast, KMB shows larger standard deviation than SPT's in Fig.7. This is because that the optimization procedure of KMB resolves to reduce the total hops in the multicast tree, but enlarge the differences of path lengths at the same time.

3) Discussion

In summary, the flow control mechanism can work well in Receiver Proxy to adjust throughputs in the multicast tree, and results of the second simulation confirm us that our prototype for customized multicast services can isolate multicast tree in the substrate network in accordance to plug-in algorithms. Furthermore, different multicast algorithms vary a lot in network resource consumption and service performances. It is unrealistic to hold the assumption that an optimal multicast algorithm can be designed to satisfy all of performance requirements from users. For example, KMB algorithm cannot always keep optimal performances in terms of average delay and its standard deviation at the same time in comparison to SPT. However, it is another way of saying that our platform is able to load different multicast algorithm for provision of self-customized multicast services, and our solution of such a customized prototype is feasible and operable.

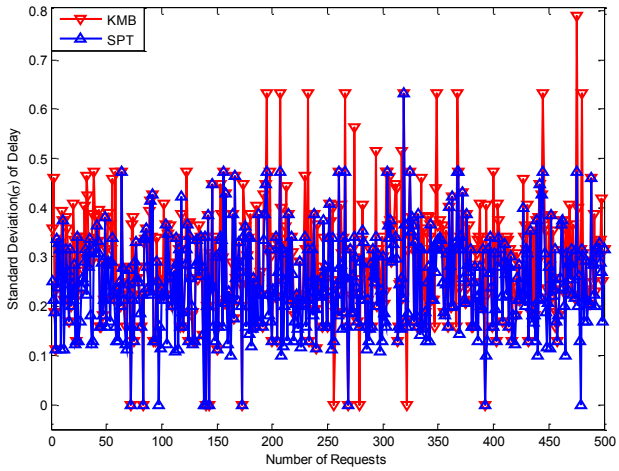


Fig.7 Standard Deviation of Dynamic Requests' Delay

V. CASE STUDY

The simulation experiments in last section has validated that our framework for customized multicast services is feasible and operable. In this section, two appropriate scenarios are designed to illustrate that our system can support practical applications: multicast text transfer and stream media services. Multicast text transfer application shows that the text message can be delivered in a multicast transmission to all its receivers through the backbone network in Fig.3. The second experiment confirms us that our last hop translation can process large number of packets, since multicast stream media could run smoothly in the edge network. These two experiments further verify the feasibility and operability of our prototype in the real world.

A. Multicast Text Transfer Application

This experiment is designed to validate that our platform can support multicast text transfer application in backbone network. For that purpose, we generate two multicast groups in the Internet2 topology (Fig.3), and map them onto the substrate network under the architecture of our prototype. The sender of the first group is h4 (IP: *10.0.0.4*), and its receiver set is {h1, h6, h13, h20}. With its sender being h8 (IP: *10.0.0.8*), the second receiver set is {h1, h10, h14, h20}. After that, we run processes to send message "GROUP#1: INFO in the 1st group!" in h4, and message "GROUP#2: INFO in the 2th group!" in h8, 15 times respectively. It can be seen in the Fig. 8 that both receiver sets of Group #1 and Group #2 have received the text message from their own multicast groups (*10.0.0.4: 40452* and *10.0.0.8: 52728*).

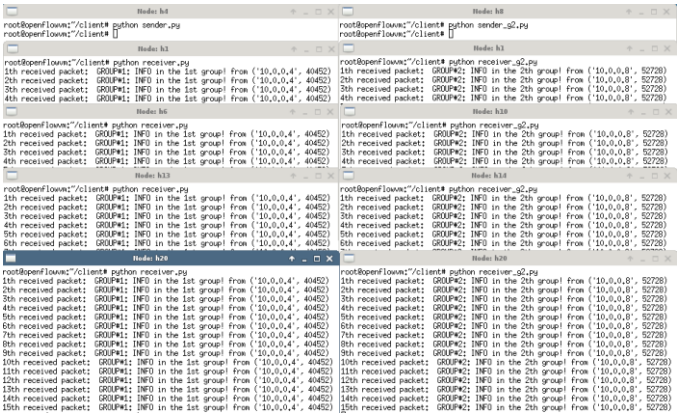


Fig. 8 Testbed running text transfer application

B. *Multicast Streaming Media Application*

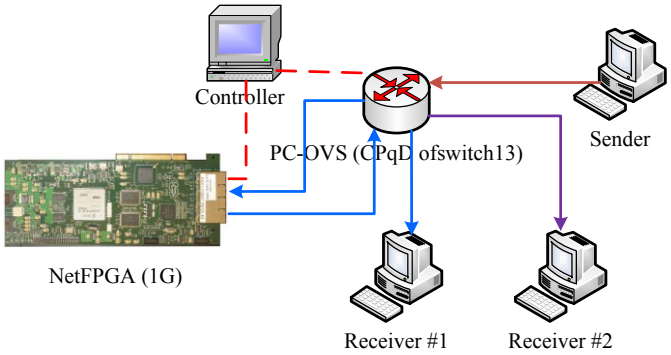


Fig. 9 Topology of streaming media application

Fig.10 Testbed running streaming media application

This experiment validates whether our last hop translation module can process large number of packets through running multicast stream media application. We build a testbed as show in Fig.9, which is the edge network of topology shown in Fig.3. The two receivers are mounted to a OF-switch (PC-OVS: CPqD ofswitch13), and a sender send a multicast flow to them through the switch. The controller has delivered the multicast paths to the OF-switch. During the multicast transimission, the switch should reform those multciast packets with **Forwarding Translation** module for the two receivers. It is well known that the packet flow rate of streaming media is big. That would bring a greate chanllenge to our **Forwarding Translation** module. At the beginning, when the NetFPGA is not accessed, the OVS (Open Virtual Switch) drop a lot of packets. That may be because our PC is not powerful, so that it cannot timely process so many packets. Hence, the NetFPGA is introduced in our testbed to undertake part of forwarding traslation (as seen the blue flow) to reduce the load of OVS. The simulation result is shown in Fig.10. The two hosts with linux can receive and play the multicast flow sent from the VLC media player in the laptop (The entire vedio recording can be availabe at [14]). It can be seen that out prototype can efficiently support multicast streaming vedio tasks.

VI. CONCLUSION

We have presented a prototype for customized multicast services in software defined network. Within plug-in algorithms, it can provide customized multicast services. Our simulation experiments and two practical applications showed its feasibility and operability in the real world. Furthermore, the platform proposed two critical modules, namely, the receiver proxy controlling the throughput on the multicast tree and the forwarding translation to deal with the multicast address issue at the edge network.

ACKNOWLEDGEMENT

This work is supported by the National Basic Research Program of China (973 Program) (2012CB315903), the Key Science and Technology Innovation Team Project of Zhejiang Province (2011R50010-05), 863 Program of China (2012AA01A507), and the National Natural Science Foundation of China (61379118).

REFERENCES

[1] M. Isard, M. Budiu, Y. Yu and etc., “Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks”, In Proceedings of ACM EuroSys’07, 2007.

[2] Greg Goth, “Software-Defined Networking Cloud Shake Up More than Packets,” *IEEE Internet Computing*, July/August, 2011.

[3] Lucas Bondan, et al., “Multiflow: Multicast Clean-slate with Anticipated Route Calculation on OpenFlow Programmable Networks,” *Journal of Applied Computing Research*, Vol. 2, No. 2, pp. 68-74, 2012.

[4] Daisuke Kotani, et al., “A design and implement of OpenFlow Controller handling IP multicast with Fast Tree Switching”, In Proceddings of IEEE/IPSJ SAINT, July, 2012.

[5] RYU [online]: https://github.com/osrg/ryu.

[6] Dan Li, Jiangwei Y., et al., “Exploring Efficient and Scalable Multicast Routing in Furture Data Center Networks,” Transaction on Networking, Vol. 20, No. 3, Jun. 2012.

[7] Julius R. et al., “Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks”, Netw Syst Manage, Jul. 2014.

[8] Yang Yu, et al., “OFM: A Novel Multicast Mechanism Based on OpenFlow”, Vol. 4, No. 9, pp. 278-286, May 2012.

[9] Cesar A. C. M. et al., “CastFlow: Clean-Slate Multicast Approach using In-Advance Path Processing in Rrogrammable Networks”, ISCC 2012.

[10] Yukihiro N. et al., “A Management Method of IP Multicast in Overlay Networks using OpenFlow”, In Proceeding of ACM HotSDN’12, 2012.

[11] L. Kou, G. Markowsky and L. Berman. “A fast algorithm for steiner trees”, *Acta Informatica*, Vol. 15, No. 2, pp.141-145, 1981.

[12] IP backbone topology of Internet2 [Online]: http://noc.net.internet2.edu/i2network/maps-documentation/maps.html.

[13] Minlan Yu, et al., “Rethink virtual network embedding: substrate support for path splitting and migration,” ACM SIGCOMM Computer Review, Vol. 38, No. 2, pp. 17-29, 2008.

[14] Video record [online]. http://youtu.be/Xz9oAgMtYl8.