Provider-Side VoD Content Delivery Using OpenFlow Multicast

Bernard Niyonteze, Jae Yoon Chung, Jian Li,

Jae-Hyoung Yoo, James Won-Ki Hong

Pohang University of Science and Technology

{bernard, dejavu, gunine, styoo, jwkhong}@postech.ac.kr

요 약

IP based multicast is considered as the most efficient way to deliver data to multiple receivers at the same time with preserving high throughput, and hence, most of the IPTV system exploits IP based multicast protocol to deliver content especially between Super head end often called root node and Regional Head end called local nodes. With this protocol, the data transmission paths are determined by multicast tree which is constructed and maintained by each network router in distributed manner. Since each network router has no holistic network view and it is not aware of the entire network status, even though some local nodes encounter a drastic packet loss due to the network congestion, the multicast tree will still be maintained, and this in turn causes waste of network resources. OpenFlow is one of the enabling technologies to manage and control a network in a centralized and programmable manner by considering the current network status. Therefore, in this paper we take advantage of OpenFlow and propose a resource efficient "Provider-Side VoD Content Delivery method using OpenFlow multicast feature in IPTV network.

Keywords: IPTV, OpenFlow, Content Delivery, IP multicast

1. Introduction

Internet Protocol Television IPTV) [1] is defined as multimedia services such television/video/audio/text/graphics/data delivered over IP based network managed to provide the required level of Quality of Service(QoS) and Quality of Experience, security , interactivity and reliability. Television (TV) channels and Video on Demand (VoD) are delivered to the television sets through a broadband connection instead of being delivered using the conventional cable or broadcast formats. The video streams are encoded into series of internet protocol packets and then carried through the public internet means which can be received by anyone who have a subscription for the service.

IPTV mainly supports two services including live TV and the stored videos called VoD service. VoD is a system which allows users to select and watch video content when they choose rather than having to watch at specific broadcast time [28]. VoD service is becoming a dominant service in the telecommunication market because it offers great convenience regarding the choice of the content items and independent viewing time.

In this paper we mainly focused on the network resource efficient in VOD content delivery on provider side. This means VoD content delivery between root node and local nodes. Service provider to deliver VOD content from root node to local nodes uses IP multicast protocol, which delivers content simultaneously to multiple local nodes to save network resources. To deliver content from root node to local nodes, IP multicast builds tree, however the problem is that if some of the local nodes cannot receive VoD packets because of network link congestion, IP-multicast keeps on sending VoD packets to the unavailable nodes. As results these packets are dropped on these links. Therefore, there is a waste of network resources. This is because IP-multicast does not update multicast tree even though some links are suffering from performance problem. To solve this problem we need a solution to adjust multicast tree topology dynamically according to link utilization and apply specific actions on the switch nodes not to forward multicast traffic instead of dropping packets. We propose multicast-based VoD Delivery service using Software Defined Networking. With SDN we efficiently deliver content from root node to local nodes, and we can dynamically adjust multicast tree while monitoring link utilizations.

SDN is an emerging paradigm in networking research and industry areas to deal with the appearing demands for flexible and agile network control [2, 3, 4]. SDN is defined as an approach to networking in which control plane is decoupled from data plane and tasks related to control plane are offloaded to separate piece of software called controller. SDN consolidates the control plane, so that a single software control program controls multiple data plane elements (i.e. routers, switches) via a well-defined API. OpenFlow [5] is a prominent example of such an API. An OpenFlow Switch consists of one or more flow tables and group tables, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller. The controller manages the switch via the OpenFlow protocol. Using this protocol, the controller can add, update, and delete flow entries, both reactively and proactively.

SDN is claimed to have a more flexibility than the legacy networks and provides abilities to speed up innovation [2]. SDN brings several advantages compared to traditional network. SDN, as programmable approach, promises to bring new level of flexibility, management and control to networking. Using SDN technologies it is possible to define, to change and to modify networking logic and network policy on regular basis, all from a central location, and propagated throughout the network. In addition, network logic at the controller level is truly programmable, meaning that the new services can be defined, innovations can be introduced and the network can be customized to meet application requirements. To realize our proposed idea we exploited group table feature provided by OpenFlow to implement multicast scheme, and implemented a resource efficient VoD content delivery solution on top of SDN controller.

2. Background and Related work

Multicast is the delivery of information to multiple destinations simultaneously using the most efficient strategy to deliver the message over each link of the network only once and only creates copies when the link to destinations split [6]. Multicast over a network allows sender to distribute data to all interested parties while minimizing the use of network resources. It is proved to be a useful and cost effective alternative to traditional methods of sending large multimedia files across network.

Y. Yu et Al [7] proposed a multicast mechanism based on OpenFlow. They separate data and control plane by shifting the multicast management to remote centralized controller. The demonstration shows that the proposed solution compared with traditional multicast yield a good performance, remove burden of multicast routers and increases the router’s packet forward speed. This work is similar to our work in a way that it studied multicast using OpenFlow. However, our work is fundamentally different in that it considers VoD content delivery between root node and local nodes whereas their work studies the feasibility of multicast in OpenFlow considering end users.

N. Khaing et Al. [8] proposed the extension of IP multicast service model to support multicast addressing and filtering for large-scale multicast applications. Data are filtered by middleware before passing it to the application, and in addressing, data are routed only to those have expressed their interest. However, in this work the main objective is scalability.

D. Chang et al. [9] proposed an SDN-based content delivery framework, which support name-based routing and caching. They describe how the proposed method support flow management with contents names where the contents are mapped to IP addresses by a controller and the flow matching is performed on a switch with the mapped IP address. They also explain how the SDN enhance the efficiency of content delivery. By using SDN structures and properties the route of content delivery is dynamically selected or modified based on the network information and also show how cached contents are managed in centralized or decentralized manner.

D.Agrawal et al. [10] presented an IPTV distribution model. Their objective is to develop a general framework for planning an IPTV service deployment. They look at ways to maximize the numbers of subscribers that can be added to a given infrastructure in different scenarios. However, in the above research works, authors mainly focused on scalability of the system to increase the number of customers who can access the content from the edge servers.

3. Content Delivery System Architecture

Architecture of the proposed content delivery service consists of three main components including IPTV network, SND Controller and Content Delivery Manager (Figure 1)

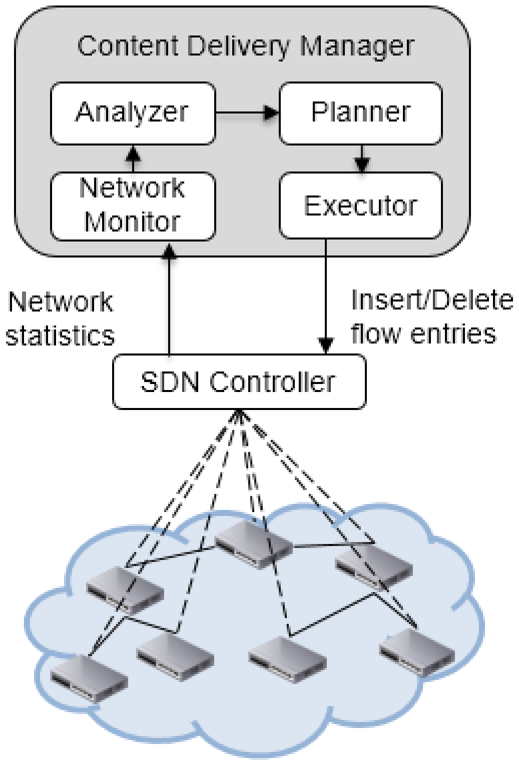

Figure 1.Content Delivery System Architecture

3.1 Content Delivery Manager

Content Delivery Manager which is a fundamental component of our system consists of four modules including Network Monitor, Analyzer, Planner and Executor.

Network Monitor: this module periodically collects links utilization from network.

Analyzer: It is responsible to build multicast tree and to compare the previous tree and current.

Planner: is responsible to check if there is a change is multicast tree and to apply modification either to add or to delete flow entries in switches.

Executor: It is responsible to send update message to network switches to change flow entries and group entries.

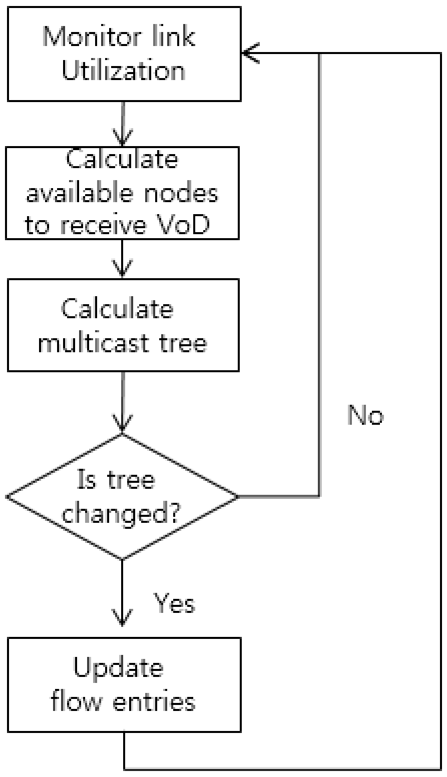

Figure 2.Control Loop Management

We implemented the control loop (Figure 2) to test the proposed method. The control loop starts with Network Monitor which is responsible to monitor link utilization through SDN controller, then analyzer calculates available local nodes which can receive VoD contents and calculates a multicast tree. Planner checks if there is a change in multicast tree. Thereafter executor updates flow entries to add or prune link in multicast forwarding tree.

To implement IPTV Content Delivery using OpenFlow multicast we exploited OpenFlow group table features appeared in OpenFlow specification 1.1version [11]. Group table is used to specify actions to a group of flows. Group table enables executions of multiple actions from single group for multiple flow entries from different tables. Group table have different group types including “all” which is used to implement broadcast and multicast. Packets of this group are processed by all actions buckets. The actions of each bucket are applied to the packet consecutively. For example, a group table entry with type all contains two actions buckets. The first bucket consists of the action “forward to Port 1”. The second bucket consists of the actions “forward to Port 2” then the switch sends the packet to both port 1 and port 2.

3.2 Algorithm to calculate multicast tree

The algorithm resides inside Planner component in the proposed control loop, and takes four arguments as input, and those are as follow:
Current Multicast Group Entries (*current_ge***)**: denotes a set of group entries which is stored in OpenFlow switches before rebuilding the multicast tree. Based on these group entries inside switches, the multicast tree is constructed. Each group entry specifies which ports (*out_port*) are used to forward the outgoing flows, and which ports (*in_port*) are used to accept the incoming flows.
Available Local Nodes (*available_nodes***)**: stores a list of local nodes whose available bandwidth exceeds the pre-defined threshold so that guarantee the multicast traffic to be received without any loss.
Sender Node (*sender_node***)**: denotes the node resides in head end IPTV network. This is the only sender in the

multicast tree, there is only outgoing traffic and incoming traffic from sender node.
Topology (*topology***)**: denotes a network topology which contains a set of switches and links between switches. Topology information can be retrieved from the topology discovery application which runs on top of OpenFlow controller.

Algorithm: Algorithm to build OpenFlow based Multicast

```
Input: Current Multicast Group Entries: current_ge
       Available Local Nodes: available _nodes
       Sender Node: sender_nodes
       Topology: topology

1   Initialize new_ge:
     /* Build a new group table instance                                  */

2    for   switch ∈ topology do
3       for port ∈ GetallPorts(switch) do
              /* insert the Uplink flows                                   */
4             descendantNodes = GetDescendants(switch, port);
5             if sender_node ∈ descendantNodes then
6                  AddEntryToMCTree(new_ge, switch,in_port=port);
              /* insert the downlink flows                                 */
7             else if available_nodes ∩ descendantNodes ≠ ∅ then
8                  AddEntryToMCTree (new_ge,switch,out_port=port);

    /* different part of current and new group entries                    */
9   Δ ← diff(new_ge ,current _ge )
   /* update group tables of switches                                     */

10  UpdateTo groupTable(Δ);
11  current_ge ← new_ge)
```

The algorithm is mainly comprised of two parts which are 1) calculating the multipath tree by building a new group table instance, 2) comparing the new multicast tree with current multicast tree to obtain the diff part only. By using this diff part we can minimize the number of switches whose group table should be updated to reflect multicast tree changes.

The multicast tree calculation can be further divided into two parts. Since each switch has to deal with both uplink and downlink flow, therefore, we need to specify two directional flows respectively. The sender node should be treated differently, and at least one port of switches on the path between the sender node and core switch should be occupied for transmitting all incoming traffic. Switches which are not on the path between sender node and core switch, should only need to deal with the outgoing traffic. By considering this, we design the multicast tree calculation logic as shown in pseudo code line 2-line 8.

At first the algorithm traverses the entire topology to obtain all switches, and for each switch it tries to obtain all ports (line2-3). Once a switch and a port in the switch have been specified, the algorithm tries to get all descendant nodes in the sub-tree which is directly connected to the port. And then, we search through all the descendant nodes to find out whether they contain the sender node. If they subsume the sender node, we will add a group entry which is comprised of a pair of switch id and in_port number to match over incoming flows (line5-6). If the descendant nodes contain at least one available node, then we will add a group entry which is comprised of a pair of switch id and output port number to match over the outgoing flows (line 7-8). By going through the above procedures, we can build a new multicast tree (group entries) in accordance with the currently available nodes. Since by using the OpenFlow we can partially change the multicast tree by updating the corresponding switches' group table, thus we try to fetch the different part between new and current multicast group entries, and we call this delta (line 9). Finally, we update the corresponding switches' group table by issuing the group table update message through invoking UpdateToGroupTable method with delta as input parameter.

4. Validation

In this section we describe how we implemented and validated the proposed VoD content delivery service for IPTV using OpenFlow Multicast. First we present testbed environment in which we describe an IPTV network topology, and explain the implementation details of the proposed approach.

4.1 Testbed Construction and Implementation

The testbed environment for validating the proposed approach is described as follows: first we describe an IPTV network topology (Figure 3) used for running our experiment; second, we describe the detailed implementation.

The network topology is divided into 8 regions, switches and local nodes reside in each region. A region binds to a geographically separated area in which a certain number of subscribers use IPTV services. Since the number of subscribers would be varied from area to area, to simulate and accommodate such variation, we assigned different numbers of switches and local nodes in each region. For example, there was only one node with two switches in region 1, and two nodes with four switches in region 2 and so on. Overall, with this parameter setup, there were 80 switches, and 20 local nodes in this network topology.

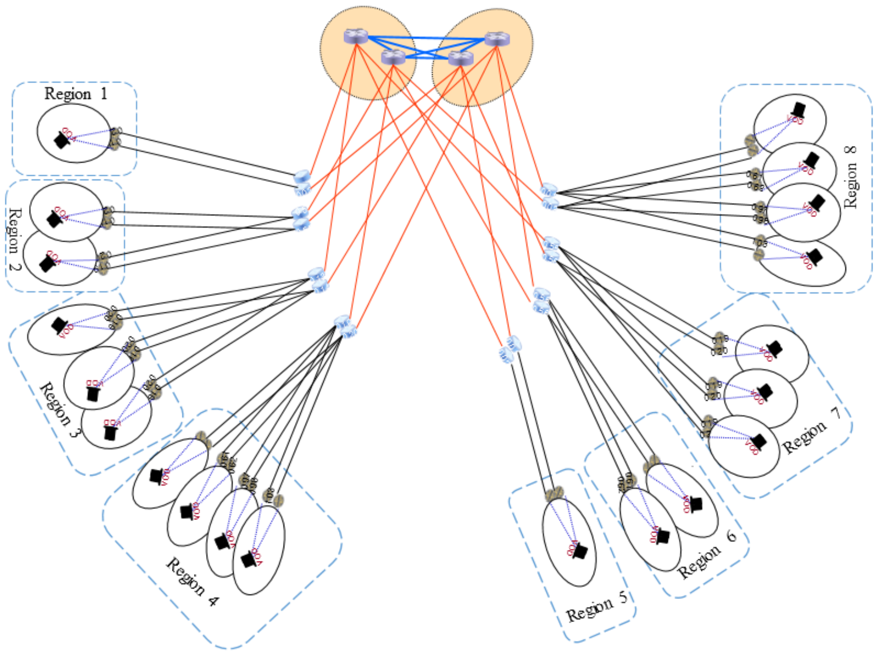

Figure 3. IPTV Network Topology used for running our experiment

To emulate the network topology, we adopted a well-known network emulator - Mininet [12], in which a set of virtual hosts and virtual switches reside. A virtual host was emulated as a single OS process, whereas, a virtual switch was realized using OpenVSwitch (OVS) [13]. Note that, the exploited OVS supported the OpenFlow protocol up to 1.3 and was compatible with the most of OpenFlow controllers. To the best of our knowledge, Ryu [14] supported almost all features of OpenFlow 1.1, therefore, we adopted Ryu as the OpenFlow controller, and implemented the proposed approach as a Ryu application. Although the resulting solution was relying on specific controller and virtual switch solution, however, since OpenFlow provides vender-agnostic interface, our solution can be easily deployed using any OpenFlow based hardware and software switches.

We call the resulting solution as Content Delivery Manager, which contains four main components as we have shown in previous section. Network Monitor periodically collects network statistics from the forwarding devices (switches) in polling manner, and forwards collected statistics to analyzer component. Analyzer analyzes the statistics, calculates available node to receive VoD content and calculates multicast tree, then sends information to the planner. In our case, we check whether there are any switches change their status either from busy to idle or from idle to busy. The planner compares the current multicast tree and the previous tree to find the difference. Since in the proposed scheme, we do not need to build the entire multicast tree, therefore only a differentiated multicast sub-tree is calculated and resulted as the output of the planner. As long as any status change occurs, planner notifies this to the executor. The executor finally converts the sub-tree into a set of OpenFlow commander and issues to the corresponding switches.

4.2 Experiment results

To evaluate the proposed approach, we exploited the testbed introduced in previous section. To simulate the video traffic, we artificially generated the UDP traffic using iPerf program with 20 MBps speed. For the sake of simplicity, we generated one video traffic, and it was initiated from node 1 which is located in region 1, and multi-casted to all residual nodes through intermediate switches. We performed the traffic measurement on node 8 located in region 4, for the purpose of validating whether the OpenFlow based multicast works properly. Moreover, to verify the proposed approach, we artificially generated and injected the cross traffic to node 8 at around 9 second with 22 MBps speed. Note that the threshold value that we chose to trigger the proposed algorithm was around 35 MBps, and the total amount of link capacity was 100 MBps.

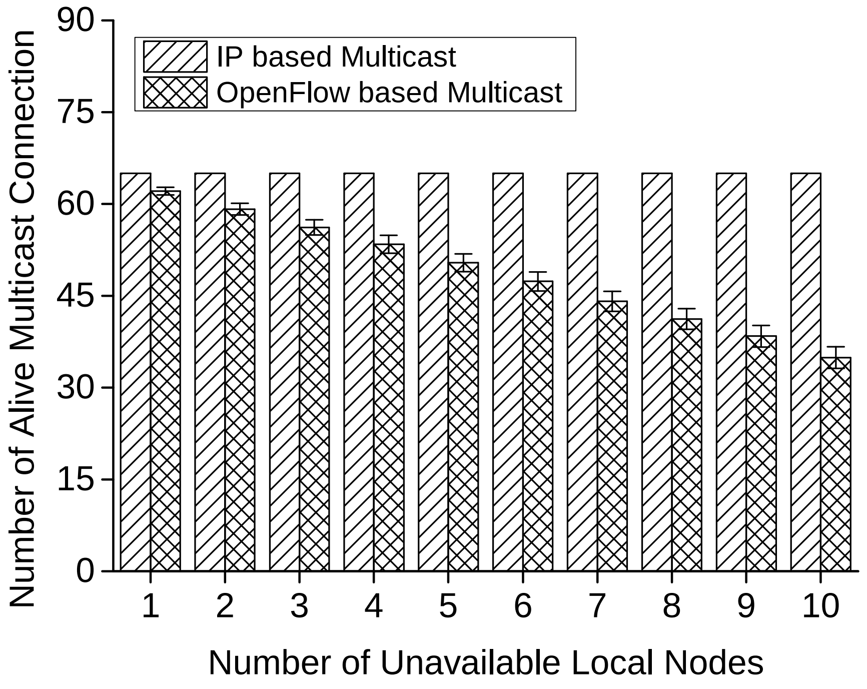

Figure 4. Comparison on the number of alive connection

In the first experiment, we compared the total number of alive multicast connections in the network topology by choosing different multicast schemes. For the comparison purpose, we chose the IP based multicast as the base line approach, and varied the number of unavailable local nodes by generating the cross traffic to randomly selected local nodes. We performed 100 tries for each number of unavailable local nodes and the experiment result has been shown in Figure 4.

From Figure 4 we can observe that, with IP based multicast, the number of alive multicast connections remained the same, no matter how many number of unavailable local nodes we have. In contrast, since OpenFlow based multicast proportionally prune the unnecessary branches from multicast tree by referring to the network condition, we can observe that the number of alive multicast connections were reduced in accordance with the number of unavailable local nodes increased. More interestingly, the standard deviation of the number of alive multicast connection increased as the number of unavailable local nodes increased. This is because with larger number of unavailable local nodes, there would be much more different ways to prune the multicast tree, and this is mainly determined by the location of unavailable local nodes.

The purpose of the second experiment is to evaluate the performance of planner and executor. The planner deals with building the multicast tree in the case of the availability status of local node changes due to the cross traffic. As we have mentioned in previous section, the time for building the multicast tree is mainly contributed by the multicast tree calculation time and the time for obtaining the differentiated part between newly calculated and existing multicast trees. We ran 20 experiments and obtained the time consumed by the planner and the executor respectively shown in Figure 5. The time consumed by the planner was further divided as calculation time and diff time in figure 5, and the time consumed by the executor is denoted as execution time. As we can see, the sum of calculation and diff time contributed the most which was around 3.8 ms, whereas executor only required the time which was less than 0.05 ms. Overall, the time for planner and executor was around 4 ms, which is fairly short time for building the multicast group tree.

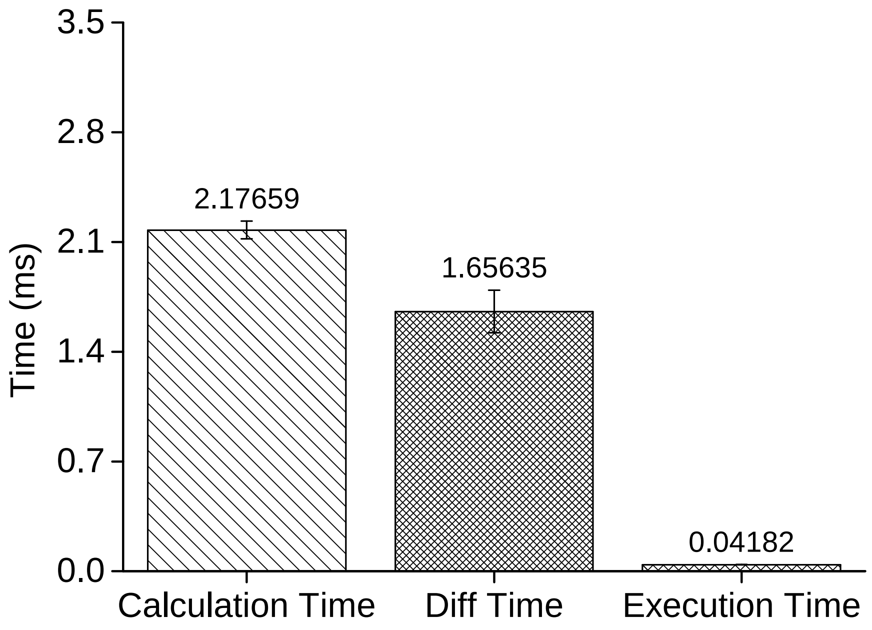

Figure 5. Performance of planner and Executor

In the following experiment, we evaluated the performance of the entire procedure on the proposed control loop. Since the current version of OpenFlow only supports network metric reporting in polling manner, therefore we implemented the network metric collecting procedure in polling manner with a predefined periodicity. Three experiments were conducted by varying the polling interval. We set 1, 2 and 3 seconds respectively for each experiment. The reason why we used different time interval value is to find out the impact of polling interval on the entire control loop procedure.

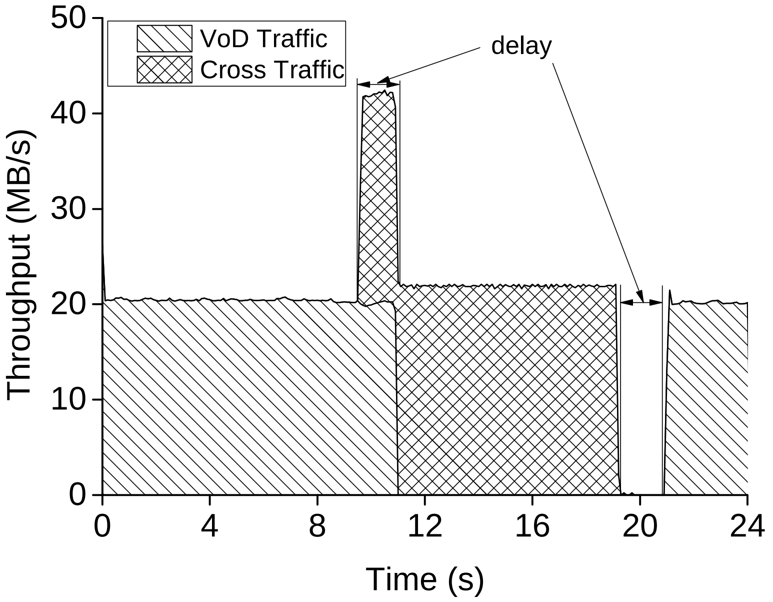

Figure 6. Traffic pattern with 1 second polling interval

The experiment results have been shown in Figure 6, Figure 7 and Figure 8. As we can observe from the figures:

1) In all cases, our solution correctly terminated the VoD traffic by forcing the affected local node to leave from the multicast tree, since the sum of VoD traffic and cross traffic exceeded the preconfigured threshold value which was 35 MBps.

2) It also shown that the VoD traffic did not terminate immediately after injecting the cross traffic. There was observable time lag after commencing or terminating the cross traffic. Such consequence was inevitable because the network metric collecting mechanism was implemented in polling manner. Therefore, even if the throughput of a certain link exceeded the threshold value, such event would not be spontaneously reported to Content Delivery Manager, instead, Network Monitor had to sense such event by polling the network metrics with a certain time interval.

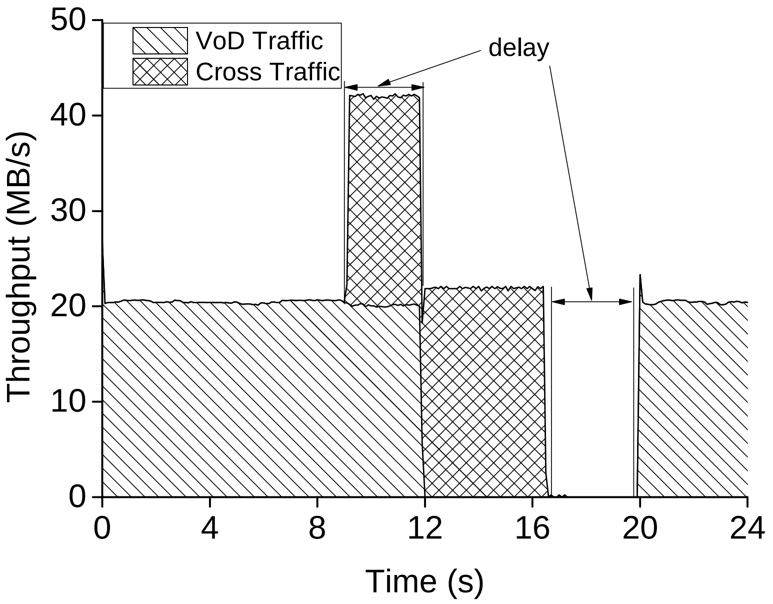

Figure 7. Traffic pattern with 2 seconds polling interval

Moreover, we also measured the lag time during leaving and joining multicast group. Unsurprisingly, as the polling interval increased, the lag time for leaving and joining the multicast group increased accordingly. The detailed results have been shown in Table 1. We also measured how much packets were transmitted during the lag time, and the results have been shown in Table 2. Although the resulting solution shown lag time in accordance with the polling interval due to the implementation issue, but we have very strong confidence on eliminating such lag time by reporting the status of local node changing event using trap mechanism rather than polling mechanism. However, implementing trap mechanism requires intensive modification on underlying switch as well as OpenFlow protocol which contradicts with the OpenFlow de facto standard.

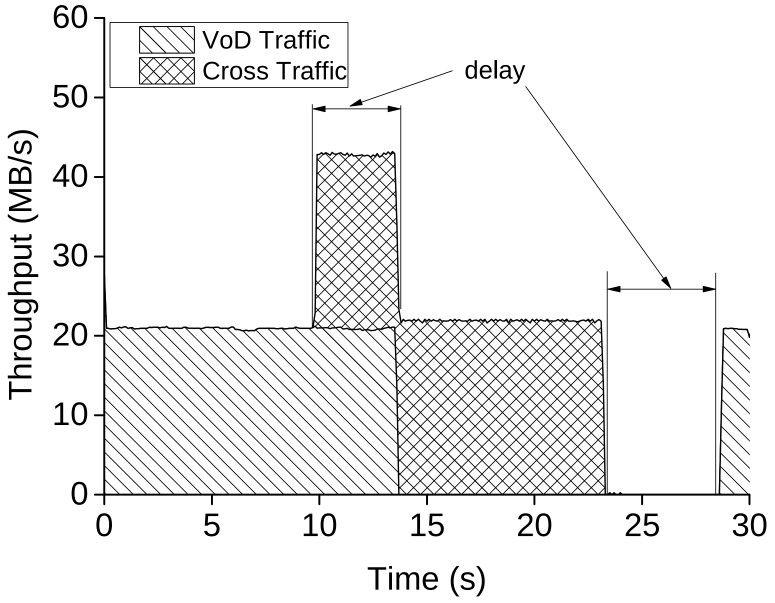

Figure 8. Traffic pattern with 3 seconds polling interval

Table 1. Detection delays

	Polling Interval Time		
	1second	2 seconds	3 seconds
Lag time during leaving multicast group	1.3	2.8	4.8
Lag time during joining multicast group	1.2	2.7	4.7

Table 2. Sum of packets transmitted during the overlap time and detection delays

	Polling Interval Time		
	1 Second	2 Seconds	3 Seconds
Detection delays(s)	1.3	2.8	4.8
Sum of packets transmitted(MB)	28057	56884	80778

5. Conclusion and Future work

In this paper we proposed provider-side VoD content delivery solution. We focused on the VoD content delivery between root node which resides in super head end and local nodes reside in regional head end in IPTV network. The proposed solution is based on multicast protocol, since the IP based multicast suffers from the network resource wastage problems, we adopted the OpenFlow to realize the multicast and further enhance it by exploiting control-loop based management mechanism. We implemented the proposed content delivery solution and named as Content Delivery Manager which consists of four components including Network Monitor, Analyzer, Planner and Executor. We exploited group features to realize multicast, and adopted Ryu SDN controller to implement the Content Delivery Manager. We performed extensive experiments using virtual network topology and the experiment results shown that OpenFlow based multicast can save network resources compared to IP based multicast.

It is noticed that some local notes may not receive all delivered content because of link congestion. P2P is important aspect we will consider to extend our future work. As some local nodes may not receive all delivered content because of link congestion, to deliver missing part of content we will study how to implement P2P using OpenFlow technology, in this case a local node with full content can serve neighboring nodes with the missing VoD content.

6. Reference

[1] T. Plevyak and V. Salim, "Next Generation Telecommunications Networks, Services, and Management." JohnWiley & Sons, 2011.
[2] "Software Defined Networking" https://www.opennetworking.org/sdn-resources/sdn-definition
[3] W.Braun and M Menth "Software Defined Networking Using OpenFlow: Protocols, Applications and Architecture Design Choices", Future Internet, 12 May 2014
[4] N. Feamster, J. Rexford and E. Zegura "The Road to SDN", ACM, December 30, 2013.
[5] N. McKeown, T. Anderson, H. Balakrishnan, G.Parulkar, L.Petterson, J.Rexford, S.Shenker and J.Turner, "OpenFlow: Enabling Innovation in Campus Networks" ACM SIGCOMM April, 2008
[6] A. Nagata, Y.Tsukiji and M. Tsuru, "Delivering a File by Multipath-Multicast on OpenFlow Networks. INCoS, 2013 5th International Conference.
[7] Y. Yu, Q. Zhen, L. Xin, C. Shanzhi," OFM: A novel Multicast Mechanism Based on OpenFlow", Advances in information Sciences and Service Sciences, May, 2012
[8] N. New Khaing, T. Phyu and T. Thu Naing ,"IP multicast Content Delivery System for Large Scale Applications" Information and Telecommunication technologies, 2005. APSITT 2005 Proceeding 6th Asia Pacific Symposium
[9] D. Chang, M. Kwak, N.Choi, T.Kwon and Y.Choi "C-flow: An efficient content delivery framework with OpenFlow", ICOIN, 2014
[10] D. Agrawal, M.S. Beigi, C.Bisdikian, L. Kang-Won, "Planning and Managing the IPTV Service Deployment. In proceeding of 10 the IFIP/IEEE International Symposium on Integrated Network management IM '2007, Munich, Germany May 2007
[11] "OpenFlow Switch Specification Version 1.1.0." Open Network Foundation, Feb.28, 2011
[12] "Mininet: An instant Virtual Network on your Laptop (or other PC)." http://mininet.org/
[13] Nicira Networks, "Open vSwitch: Production Quality, multilayer Open Virtual Switch", http://openvswitch.org
[14] "Build SDN Agilely: RYU Controller", Component-based software defined networking framework." ttps://github.com/osrg/ryu

Bernard Niyonteze

2005~2009: Bachelor of Science in Computer Science, Kigali Institute of Education, Rwanda
2012~2014: Master of Science in Computer Science and Engineering, Pohang University of Science and Technology, South Korea

Jian Li

2007 연변과학기술대학교, 전자통신공학과 학사
2012 포항공과대학교, 컴퓨터공학과 석사
2012 ~ 현재 포항공과대학교, 정보전자융합공학부 박사과정

Jae Yoon Chung

2005. 3 ~ 2009. 2: B.S. in Computer Science and Engineering, Pohang University of Science and Engineering, South Korea.

2009.3 ~ present: Ph.D. Candidate, Dept. of Computer Science and Engineering, Pohang University of Science and Engineering, South Korea.

Jae-Hyoung Yoo

1983 연세대학교 전자공학과 학사
1985 연세대학교 전자공학과 석사
1999 연세대학교 컴퓨터공학과 박사
1986 ~ 2012: KT 네트워크 연구소
2012 ~ 2013: KAIST 전기 및 전자공학과 연구부교수
2013 ~ 현재: POSTECH 컴퓨터공학과 연구부교수

James Won-Ki Hong

1983 Univ. of Western Ontario, BSc in Computer Science
1985 Univ. of Western Ontario, MS in Computer Science
1985 ~ 1986 Univ. of Western Ontario, Lecturer
1986 ~ 1991 Univ. of Waterloo, PhD in Computer Science
1991 ~ 1992 Univ. of Waterloo, Post-Doc Fellow
1992 ~ 1995 Univ. of Western Ontario, 연구교수
1995 ~ 현재 포항공과대학교, 컴퓨터공학과 교수
2007 ~ 2011 포항공과대학교, 정보통신대학원장
2007 ~ 2010 포항공과대학교, 정보통신연구소 연구소장
2008 ~ 2010 포항공과대학교, 컴퓨터공학과 주임교수
2008 ~ 2012 포항공과대학교, 정보전자융합공학부장
2012 ~ 2014 KT 종합기술원, 원장
2008 ~ 현재 포항공과대학교, 컴퓨터공학과 교수