Architecting Multimedia Conferencing Service using SDN

Bo Yan[*†], Rittwik Jana[†], Shu Shi[†], Yang Xu[*], Bo Han[†],
Vijay Gopalakrishnan[†], Lusheng Ji[†], H. Jonathan Chao[*]

[*]New York University
Tandon School of Engieering
Brooklyn, NY 11201

[†]AT&T Labs-Research
1 AT&T way
Bedminster, NJ 07921

ABSTRACT

Modern Web-enabled multimedia conferencing systems relays the source video flows to all call participants through media conferencing servers. This solution works well for small conference groups. However, as the conference size increases, the outbound bandwidth of the conferencing server becomes the bottleneck, which limits the scalability of the system and eventually degrades conference QoS. In this paper, we re-architect the multimedia conferencing service using Software-Defined Networks (SDN) that provides enhanced scalability and service orchestration. The main idea of our approach is to decouple the data plane and the control plane of the conferencing service. We advertise the group-based traffic forwarding capability of commodity OpenFlow switches as OpenFlow Selective Forwarding as a Service (OFSFaaS). The conferencing server now only handles control traffic, and dynamically establish media channels using OpenFlow switches through OFSFaaS. Preliminary prototype evaluations show that OFSFaaS eliminates the outbound bottleneck at conferencing servers and scales efficiently. OFSFaaS can be subscribed to by a wide range of multimedia services.

Keywords

Multimedia Conferencing; SDN; OpenFlow

1. INTRODUCTION

A key motivation for software-defined-networking (SDN) is that it enables *network programmers* to efficiently manage the network that would otherwise be difficult to realize using existing control plane mechanisms [12, 15]. *Application developers*, on the other hand, do not typically have the expertise (e.g. network topology, routing, table management) to design applications that use the various programmable features offered by SDN. Traditionally, application developers worry about the logic of the application they are developing and expect the network to efficiently transfer the packets to/from the applications or endpoints. This sharp separation between the application layer and the network layer is becoming fuzzy with the prominence of network programmability [4]. In this paper, we investigate how to advertise advanced data plane capabilities as a service, which can facilitate smart applications such as multimedia conferencing with the objective of maximizing scale and efficiency.

In a multimedia conferencing system, forwarding the video stream of every conference participant to all other participants is a core function of a conferencing server. Most modern conferencing systems have signaling channels and media channels as depicted in Figure 1. The signaling channel coordinates the conference by exchanging control messages between the clients and the signaling servers, which initiates or terminates sessions, notifies errors, and exchanges codecs and media types. The media channel is negotiated apriori as part of session initiation through the signaling channel for a client to send and receive multimedia contents.[1] Today conferencing servers are hosted in the clouds, typically having an outbound bandwidth of 62.5-250Mbps [6]. The software solution is to selectively forward a subset of streams (e.g., only the streams of active speakers) to all participants. As the number of clients grows, the server can quickly become a bottleneck due to insufficient outbound bandwidth.

Such a challenge can be alleviated by leveraging the SDN capabilities that modern networks can provide. Recent programmable data planes such as OpenFlow, DPDK and P4 support group-based traffic forwarding [8, 19]. By programming rules and groups, media channels can be deployed on-the-fly through network devices without passing through the application server. Since commodity OpenFlow switches support up to 1.44Tbps aggregate bandwidth [5, 2], the bandwidth bottleneck of the conferencing server is relieved. This

[1]Refer to Section 2 for a brief discussion of challenges of various conferencing systems.

CAN'16, December 12 2016, Irvine, CA, USA

DOI: http://dx.doi.org/10.1145/3010079.3012016

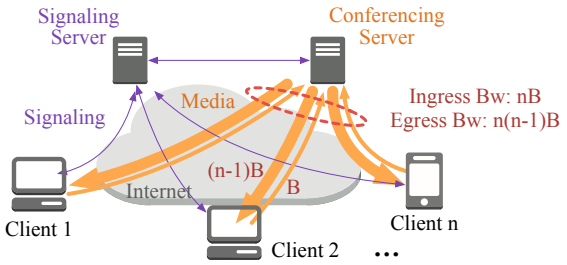


Figure 1: Conventional media bridge conferencing

motivates us to advertise SDN group-based forwarding as a service, which can be subscribed by multimedia conferencing system. However, to develop such a service faces significant challenges:

- Providing APIs oblivious to network details. Existing network programming platforms such as OpenFlow protocol [19] and Pyretic [15] require an explicit specification of data plane policies with the knowledge of network details (e.g. network topology, physical port of a switch to forward traffic), which is beyond the expertise of most application developers. Therefore, the service needs to provide APIs oblivious to these network details, and automatically compile rules and groups in switches to control media flows.
- Zero disturbance to non-conferencing traffic. As an application level service, it is unaware of the existing traffic in the network. Therefore, the rules and groups need to be carefully crafted, so that the forwarding behavior of non-conferencing traffic is not affected.
- Efficient use of switch memory. Commodity OpenFlow switches today support 4k-125k rules and 10k groups [2]. Since switch table memory resource is shared with other applications, the number of rules and groups in the switch used to implement conferencing needs to be minimized, so that more application instances can be composed using SDN.

To address these problems, we design the *OpenFlow Selective Forwarding as a Service (OFSFaaS)* to advertise the group-based forwarding capability of OpenFlow switches. We modify a multimedia conferencing system to subscribe to OFSFaaS, downshift the stream forwarding function to OpenFlow switches, and establish media channels on-the-fly. Our contributions can be summarized as follows,

- We propose architecting multimedia conferencing service with SDN. By offloading media channel deployments from the conferencing server to the OpenFlow switches, we improve scalability of the system.
- We design OFSFaaS with APIs to advertise the group-based forwarding capability of OpenFlow switches to applications without requiring detailed network information. OFSFaaS implements media channels with zero disturbance to existing network traffic, while efficiently using switch table memory.
- We implement an OFSFaaS-enabled conferencing prototype using Open vSwitch, Ryu OpenFlow controller, and Jitsi Video Bridge [3, 11]. The evaluation results show that OFSFaaS helps scale multimedia conferencing with more clients supported, and improves the conferencing QoS significantly.

The rest of this paper is organized as follows: Section 2 introduces the background and our approach of multimedia conferencing. Section 3 presents the OFSFaaS design. Section 4 shows our conferencing prototype and preliminary results. In Section 5 we generalize OFSFaaS towards support for comprehensive network and multimedia application scenarios. Finally, we conclude our work in Section 6.

2. BACKGROUND AND APPROACH

2.1 Multimedia Conferencing Techniques

IP multicast: Traditionally, multimedia conferencing has been suggested using IP multicast [10]. However, IP multicast faces configuration challenges which prohibit a widespread commercial deployment to support clients universally [9]. Recently, a series of studies such as [17, 18] investigates on implementing IP multicast with SDN. However, the lack of the adoption of IP multicast in today's multimedia applications limits their applicability.

P2P: Skype popularized conferencing via an overlay peer-to-peer (P2P) network [7]. However, with all the processing being pushed to the clients, it is difficult to deliver high-quality conferencing services to a large number of clients. Each client creates multiple media streams to its peers, which can congest the client's uplink when the number of its peers is large. Depending on the measured bandwidth between peers, the video may need to be encoded differently on each media channel, which burdens the CPUs at client devices.

Media Bridging: To achieve better scalability, enterprises employ a media bridge architecture (Figure 1), where the conferencing server deployed within the enterprise network or a cloud acts as a 'bridge' for media channels to be relayed through [1, 11]. While this strategy simplifies the complexity of clients, the aggregate bandwidth usage scales by the square of the group size, which can also congest the outbound bandwidth of the conferencing server.

2.2 Proposed Approach: Multimedia Conferencing using SDN

OpenFlow v1.1 or above provides the capability to relay traffic using multiple flow tables and a group table in an OpenFlow switch [19]. The OpenFlow controller can dynamically program rules in the flow tables and groups in the group table. Each rule consists of matching fields to match against packets, and an action to forward traffic. We set the action of each rule to forward traffic to a particular group with multiple actions for different clients.

Both the functional support and recent performance improvements of programmable switches motivate us to use the group-based forwarding capability to enable a high-performance multimedia conferencing service.[2] We developed OpenFlow Selective Forwarding as a Service (OFSFaaS) to advertise

[2]Recent switch products can handle 12K flow entry updates per second [2], which provides real-time update capability.

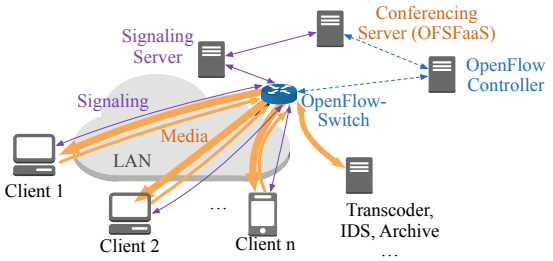


Figure 2: Multimedia Conferencing using SDN

this capability. The application developers can now discover and leverage OFSFaaS, and redesign the conferencing application. Figure 2 presents the architecture of our proposed system. The signaling channels are established using conventional L2/L3 networking through the signaling server. Different from existing architectures, the conferencing server subscribes to OFSFaaS. OFSFaaS connects to the OpenFlow controller and creates the media channels between clients and OpenFlow switches.

Our design benefits multimedia conferencing in two ways,

- From the performance perspective, the bandwidth bottleneck at the conferencing server is relieved. Only the signaling traffic, which is tiny even for coordinating large conference group sizes, is processed by the conferencing server. Since the OpenFlow switch has high data plane throughput, decoupling the media traffic from the conferencing server in this manner at the switch can deliver a high QoS.
- From the management perspective, the conferencing application can be orchestrated with other services. For instance, traffic can now be dynamically chained through other media functions such as media transcoding, intrusion detection, and data archiving.

As a proof of concept, in this work we consider conferencing service deployment in a LAN environment. The gateway of the conferencing server and the signaling server is an OpenFlow-enabled switch, which supports hybrid OpenFlow operations and conventional L2 switching and L3 routing features [19]. In this setup, all the conferencing traffic traverses through the OpenFlow switch. The entire design is made transparent to the clients, and the upgrades are only applied at the service provider edge, which simplifies service deployments. We also use unencrypted streams in our implementation for the sake of simplicity. We will extend our work to support encryption, wide area deployment and richer application scenarios as discussed in Section 5.

3. OFSFaaS DESIGN

3.1 OFSFaaS-enabled Conferencing

OFSFaaS is an application level service collocating with the conferencing application (Figure 3). The signaling server establishes signaling channels with the conferencing program and clients using standard XMPP or SIP protocols [11]. To set up media channels, OFSFaaS learns the negotiated sockets of the media channel from the northbound multimedia application and determines the physical port attached to each client from the southbound OpenFlow controller.

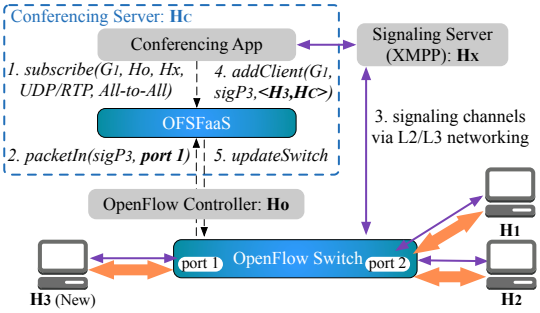


Figure 3: OFSFaaS-enabled conferencing workflow

The multimedia application such as conferencing system can leverage the northbound API of OFSFaaS to deploy media channels without detailed knowledge of the network.

OFSFaaS's northbound API consists of three functions:

`subscribe` (*group, OFcontroller, sigServer, mediaProto, fwdPolicy*)
`addClient` (*group, sigPacket, mediaChannel*)
`delClient` (*group, mediaChannel*)

The `subscribe` function registers a conferencing *group* to OFSFaaS. *OFcontroller* and *sigServer* inform OFSFaaS about the sockets to talk with the OpenFlow controller and the signaling server. *mediaProto* specifies the type of protocol to implement the media channels, and *fwdPolicy* defines the selective forwarding policy between clients. `addClient` or `delClient` is called whenever a client joins or leaves a conferencing *group*. *mediaChannel* contains the negotiated sockets of the media channel for the client. The new signaling packet *sigPacket* is used as a credential for OFSFaaS to match the physical port attached to the client.

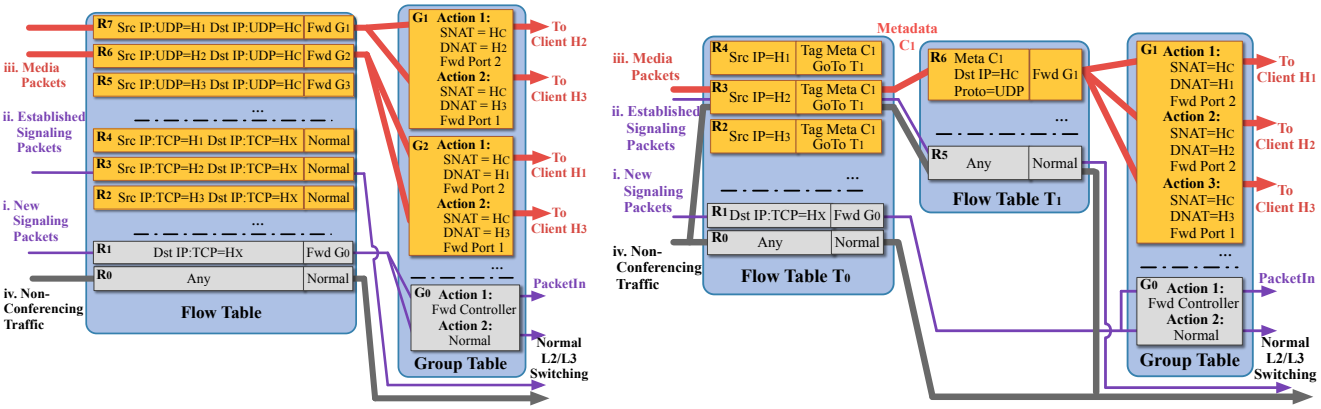
OFSFaaS's southbound API consists of two functions:

`packetIn` (*sigPacket, port*)
`updateSwitch` (`list<`*rule*`>`, *group*)

Whenever a new client joins a *group*, the new signaling packet is copied to the OpenFlow controller as a *PacketIn* message. The PacketIn message carries the physical port where the signaling packet arrives from, and reaches OFSFaaS through the `packetIn` function. The original signaling packet is sent to the signaling server through L2/L3 networking. By matching two copies of *sigPacket* received by `packetIn` and `addClient`, OFSFaaS learns the physical *port* of the switch to which the client is connected. Finally, OFSFaaS compiles necessary *rule*s and *group*s, and implements the media channels using `updateSwitch`.

Figure 3 gives an example workflow of OFSFaaS-enabled multimedia conferencing. The conferencing application first subscribes to OFSFaaS (Step 1). *All-to-All* specifies that the media traffic of each client is relayed to all the other clients.[3] Suppose clients H_1 and H_2 are already in a conference, and a new client H_3 is joining in. The new signaling packet $sigP_3$ is copied both to the controller (Step 2) and to non-OpenFlow L2/L3 switching port (Step 3). With the signaling channel established between H_3 and H_X, the conferencing application follows the original logic to negotiate the me-

[3]In addition to All-to-All, 1-to-All or K-to-All is supported to implement other multimedia applications such as WebTV or Webinar.



(a) Strawman implementation

(b) Resource efficient implementation

Figure 4: Table management implementations for OFSFaaS-enabled multimedia conferencing service. Grey entries are statically deployed. Yellow entries are updated on demand.

dia channel between H_3 and H_C, and notifies OFSFaaS to `addClient` (Step 4). By matching $sigP_3$ from `packetIn` and `addClient`, OFSFaaS learns that H_3 resides at port 1 of the switch. Finally, OFSFaaS is able to generate the corresponding rules and groups to deploy the channels (Step 5). The switch table management conducted by `updateSwitch` needs to satisfy the following data plane policies:

Policy i. New signaling packets are forwarded to both the OpenFlow controller and the L2/L3 switching port.

Policy ii. Established signaling packets are forwarded to normal L2/L3 switching port.

Policy iii. The media traffic is forwarded to all the clients in the same group.

Policy iv. Non-conferencing traffic remains intact.

3.2 Switch Table Management

In this section, we present the switch table management that implements the above data plane policies. We consider two design goals: First, zero disturbance to the forwarding behavior of non-conferencing traffic in the network. OFSFaaS is not aware of the non-conferencing traffic or the table configuration of the OpenFlow switch for other services. To ensure that other services are undisturbed, a correct implementation of Policy iv is critical.[4] Second, minimized table usage and update frequency. Note that the table memory and the control channel bandwidth are limited for OpenFlow switches, especially considering OFSFaaS is sharing the resource with other services. By minimizing the resource consumed, the system can sustain more application instances, such as conferences, at the same time.

3.2.1 A Strawman Implementation

Figure 4a presents a strawman implementation using a single flow table and a group table. The main idea is to configure exact rules for each policy and forward the media traffic to designated groups. We use the example in Figure 3 to explain the implementation. The rule with a higher index has a higher priority.

Rules R_0, R_1 and group G_0 are statically installed to implement Policy i and Policy iv. The default rule R_0 forwards all non-conferencing traffic using conventional L2/L3 switching by specifying the action as 'NORMAL'. Rule R_1 matches every new XMPP signaling packet and forwards it to group G_0. G_0 directs it to both the controller and the L2/L3 switching port. The new XMPP packet sent to the controller is forwarded to OFSFaaS by the `packetIn` function. Meanwhile, the other copy forwarded through L2/L3 port sets up the signaling channel.

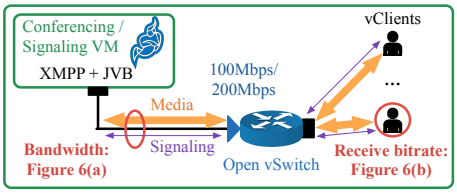
The rest of the rules and groups are deployed dynamically for Policy ii and Policy iii. After a signaling channel is established, a flow entry is installed to avoid further XMPP messages to be sent to the controller. In the example, R_2, R_3, R_4 matches the 5-tuple of the established XMPP packets, and direct them to L2/L3 network to fulfill Policy ii. To deploy the media channel negotiated through signaling, each client has a rule installed to classify the 5-tuple of the media channel, and to direct the packets to the designated group. Each group applies all the actions which forwards the traffic to the corresponding destinations (Policy iii). Here the ports to forward are previously learned from the new signaling packets. In the example, the media traffic from client H_1 matches rule R_7 and gets forwarded to clients H_2 and H_3 through port 2 and port 1, respectively. To achieve transparency to clients, SNAT and DNAT are applied to set the source as the conferencing server H_C and the destination to each client.

By composing exact entries, the strawman implementation enforces the data plane policies. However, it costs a total of $2N$ flow entries and N group entries for N clients, which is inefficient in table usage. Every client arrival or departure triggers two rule updates and multiple group updates, which can throttle the control channel of the switch.

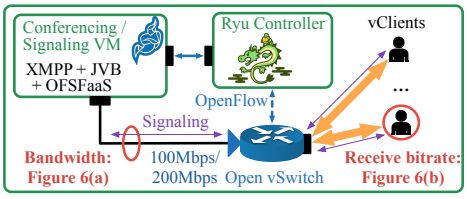
3.2.2 A Resource Efficient Implementation

We propose an alternative implementation with minimized table usage and update frequency in Figure 4b. The idea is to leverage the *flow table pipeline* and *metadata* to decouple the classification of different fields. OpenFlow supports pipeline processing of packets across multiple flow tables.

[4]This shares similar goal with network verification. Since channels need to be established or updated in real time, we cannot rely on network verification engines [13] to check the policies due to the latency. By crafting the format of rules and groups, OFSFaaS achieves verification-free with correctness.



(a) Conventional JVB setup



(b) OFSFaaS-enabled JVB setup

Figure 5: Testbed setup

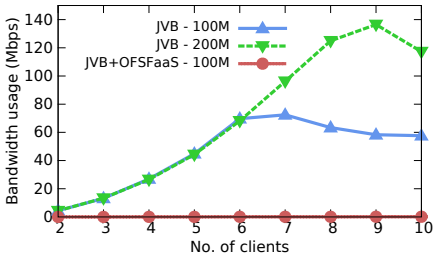
Metadata is a register attached to each packet to carry information across the tables. A rule may tag the metadata of the matched packets, which can be later used in the next table for classification. Metadata is removed before the packet is sent to the output ports of the switch.

In this implementation, We use two flow tables in the pipeline. Same as the strawman implementation, R_0, R_1, R_5 and G_0 are statically installed for Policy i and Policy iv. In flow table T_0, the source IP address is classified as traffic from accessed clients. The metadata is tagged with the conference ID to identify that all three clients are in the same conference C_1. Traffic from clients in other conferencing groups shall be tagged with different metadata. In flow table T_1, Rule R_7 matches the media packets carrying metadata C_1 and destined to the conferencing server H_C, and directs them to group G_1 (Policy iii). The established signaling traffic and non-conferencing traffic are forwarded through non-OpenFlow switching (Policy ii and Policy iv). In the group table, we generate a single group for each conference, which forwards the traffic to all the clients in the conference. Note that here the media traffic is sent back to the origin, which creates overhead. We argue that in a large conference, trading the bandwidth overhead for a smaller number of group entries is worthwhile. The played back traffic can be used at clients to evaluate media channel quality.

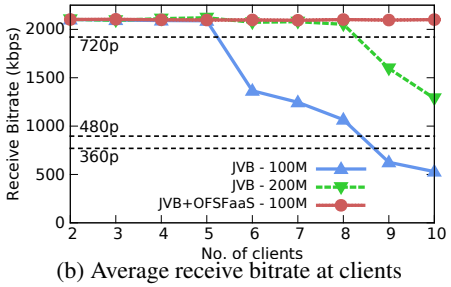
The memory efficient table management compresses flow table usage to $N + C$, and the group table usage to C, where C is the number of conferencing groups. For each joining client, only one rule and one group need to be updated.

4. PRELIMINARY EVALUATIONS

To justify our design, we implement an OFSFaaS-enabled multimedia conferencing prototype using *Jitsi VideoBridge (JVB)* as depicted in Figure 5. JVB is an open source selective forwarding unit supporting conferencing, VoIP, IM and WebRTC. The SDN environment is enabled by Open vSwitch and Ryu controller [3]. We deploy our testbed on a Dell server with two 8-core Xeon E5-2630 processors with 64G RAM. To isolate performance, the signaling and conferencing servers, and the OpenFlow controller are deployed at VMs with four virtual cores. The virtual clients and the Open vSwitch are running on the host machine.



(a) Outbound bandwidth usage of conferencing server



(b) Average receive bitrate at clients

Figure 6: Performance comparison of conventional JVB and OFSFaaS-enabled JVB conferencing services

In the conventional JVB setup (Figure 5a), JVB is deployed at the Conferencing/Signaling VM to support the virtual clients. The Open vSwitch runs in traditional L2/L3 modes. The media traffic from all virtual clients is relayed through the Conferencing/Signaling VM. In the OFSFaaS-enabled JVB (Figure 5b), the Open vSwitch is controlled by the Ryu controller. JVB leverages OFSFaaS to setup media channels. Therefore only the signaling traffic flows to the Conferencing/Signaling VM. To evaluate the scalability, in both systems we rate limit the outbound bandwidth of the Conferencing/Signaling VM at 100Mbps or 200Mbps [6]. We vary the group size to evaluate the scalability of both conferencing architectures.

The average outbound bandwidth used at the Conferencing/Signaling VM is depicted in Figure 6a. For conventional JVB, the bandwidth usage initially increased by the square of the group size. With 100Mbps rate limit, bandwidth usage starts to smooth out at 70% utilization. As JVB senses packet loss at the media channel, it initiates rate control to lower the video quality. With 200Mbps rate limit, rate control occurs at around 65% utilization. By comparison, OFSFaaS only has signaling traffic reach the Conferencing/Signaling VM, which consumes a small outbound bandwidth.

The receive bitrate at the client shows consistent results in Figure 6b. Conventional JVB initially delivers a bitrate of 2.1Mbps, which supports an advertised resolution of 720p. However, the bitrate drops intensely when more clients join in. With 100Mbps outbound bandwidth at the conferencing server, the receive bitrate drops to 500kbps when there are ten clients in a conference, which fails to support a 360p resolution. With 200Mbps, the performance also starts dropping with more than eight clients. By comparison, OFSFaaS-enabled JVB achieves a stable bitrate of 2.1Mbps. Considering the CPU and memory, we test a maximum conference group of 32 clients with 1024 video streams. OFSFaaS-enabled does not suffer from quality degradations. The re-

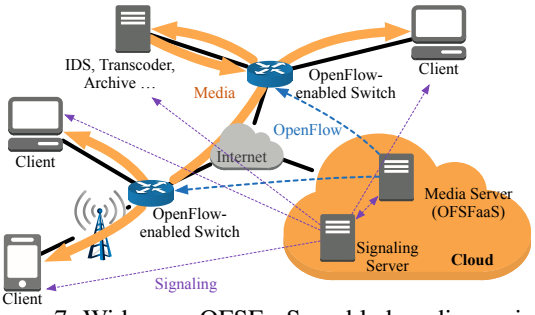


Figure 7: Wide-area OFSFaaS-enabled media services

sults prove that the bandwidth bottleneck at the conferencing server is relieved, and indicate support for a large number of clients when we deploy OFSFaaS using hardware switches.

We also compared the service setup time for new clients between conventional JVB and OFSFaaS-enabled JVB. We find that the channel negotiation dominates the latency, while the table update time in Open vSwitch is negligible.

5. DISCUSSIONS AND FUTURE WORK

In this work, we implement OFSFaaS in a LAN environment. We plan to extend OFSFaaS to support various network environments and general multimedia services. We list some of the research issues that pertain to this topic.

Wide-area Deployments. Figure 7 shows the wide-area deployment of OFSFaaS-enabled media services such as conferencing with multiple OpenFlow switches. To deliver high quality of experience (QoE), OFSFaaS needs to intelligently route media traffic to avoid congested paths, or to flexibly steer traffic through required media function boxes such as Intrusion Detection System (IDS), transcoder, or archive. With multiple switches, we also have more flexibility to optimize both the use of link bandwidth and switch memory. For instance, the conference traffic between two switches (e.g. two distant groups in New York City and Los Angeles joining the same conference session) can be trunked in one bundle before being replicated at the edge.

Securing Conferencing Service. Today, many conferencing services encrypt the media traffic. In conventional JVB, clients exchange key information with the conferencing server, and encrypt the media channels using Secure RTP protocol. OFSFaaS does not support traditional pairwise encryption due to its multicast nature. To enable secure conferencing, we plan to implement the Encrypted Key Transport (EKT) for Secure RTP originally proposed by [14]. The idea is to negotiate a shared/common key amongst all clients in the same conference for media channels.

Support for General Multimedia Applications. By design, OFSFaaS supports session-based media services such as VoIP and WebRTC. We will further investigate its support for other multimedia applications, such as live content delivery and crowd-sourced live streaming [16].

6. CONCLUSION

In this work, we proposed a novel and alternative approach to a scalable multimedia conferencing service using SDN. We exposed OpenFlow's group-based traffic forwarding capability as a service API for application developers. Our implementation shows that this approach helps scale multimedia conferencing significantly while at the same time keeping application development simple. Our broader vision is to extend this design to more general network scenarios, and eventually build a framework to discover and advertise various data plane capabilities for application developers to compose various multimedia services.

7. REFERENCES

[1] Getting Started with WebRTC. http://www.html5rocks.com/en/tutorials/webrtc/basics/.

[2] NoviSwitch Overview. http://noviflow.com/products/noviswitch/.

[3] Ryu SDN Framework. http://osrg.github.io/ryu/.

[4] OpenDaylight: Network Intent Composition Project. https://wiki.opendaylight.org/view/Network_Intent_Composition:Main, 2015.

[5] Pica8 Switch. http://www.pica8.com/products/pre-loaded-switches, 2016.

[6] Amazon AWS. Amazon EC2 Instance Configuration. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-ec2-config.html.

[7] S. A. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *arXiv preprint cs/0412017*, 2004.

[8] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming Protocol-independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[9] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *Network, IEEE*, 14(1):78–88, 2000.

[10] M. Handley, J. Crowcroft, C. Bormann, and J. Ott. The Internet Multimedia Conferencing Architecture. Technical report, Internet Draft, Internet Engineering Task Force, 1997.

[11] E. Ivov. Hangout-like Video Conferences with Jitsi Videobridge and XMPP. 2013.

[12] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the One Big Switch Abstraction in Software-Defined Networks. *Proc. ACM CoNEXT*, 2013.

[13] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *NSDI*, pages 113–126, 2012.

[14] D. McGrew and D. Wing. Encrypted Key Transport for Secure RTP. *Work in Progress*, 2007.

[15] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, et al. Composing Software Defined Networks. In *NSDI*, pages 1–13, 2013.

[16] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang. Practical, real-time centralized control for cdn-based live video delivery. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 311–324. ACM, 2015.

[17] Y. Nakagawa, K. Hyoudou, and T. Shimizu. A management method of ip multicast in overlay networks using openflow. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 91–96. ACM, 2012.

[18] K. A. Noghani and M. O. Sunay. Streaming multicast video over software-defined networks. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 551–556. IEEE, 2014.

[19] OpenFlow Switch Specification. Version 1.5.0 (Wire Protocol 0x06). https://www.opennetworking.org/, Dec 2014.