國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

在軟體定義網路中負載平衡的影音群播系統
A Load-Balanced Video Multicast Routing System in Software-Defined Networks

李孟葳
Meng-Wei Lee

指導教授：徐正炘 博士
Advisors: Cheng-Hsin Hsu, Ph.D.

中華民國 104 年 10 月
October, 2015

國立清華大學
資訊工程研究所

碩士論文

在軟體定義網路中負載平衡的影音群播系統

李孟葳 撰

104
10

中文摘要

由於傳統的群播技術(IP multicast)需要高效能的路由設備以及許多複雜的設定， 對於網路營運商而言，傳統的群播技術成效不彰。 在本篇論文中，我們實做了一個基於SDN架構的群播路由系統。針對群播路徑的建立， 我們討論影音串流的多路徑群播樹於軟體定義網路(Software-Defined Networks)的問題。 我們的目標是建立一個健全、負載平衡、具可適性且相容於軟體定義網路的影音群播路由。 我們將此群播路由問題轉換為一個最佳化的數學問題，得出一個針對鏈結負載的min-max最佳解， 並取名此演算法為Robust Multipath Multicast Routing (RMMR*)， 但此演算法為了求得最佳解而需耗時較久，所以我們另外設計了一較有效率的啓發式演算法。 我們將這兩個演算法實作在群播路由系統並安裝在OpenFlow控制器中。 我們在真實系統架設的測試平台以及利用Mininet模擬器擬真出的網路中分別做實驗， 測試群播系統的可行性、效能以及可擴展性。 在測試平台的環境中，我們測量到的系統對於群播流的設置時間都小於5毫秒， 而對於群播收看者的偵測時間皆小於0.15秒。 在模擬器中所做的實驗，以下幾點能發現我們的演算法更優於IP Multicast: (i) 降低了19%到95%的幀遺失率、 (ii) 影像品質提高了4 dB到15 dB、 (iii) 影片收看者的輸送量增加了25%到66%、 (iv) 最大鏈結使用率降低了15%到50%。 另外我們也權衡了我們設計的兩個演算法的最佳性以及所費時間，找出他們所適合的網路環境。 最佳解演算法較適合比較小而且更穩定的網路環境， 而啓發式演算法則較適合使用於更大且常有變動的網路環境中。

Abstract

IP multicast in traditional networks, dictates high-end routers and incurs high administrative overhead, which is no longer suitable for deployment due to its complicated operations. In this thesis, we implement a multicast routing system based on SDN framework. For computing the multicast routes, we study the problem of establishing multipath multicast routing for streaming videos in Software-Defined Networks (SDNs). The objectives of the considered problem are robustness, load balance, adaptiveness, and SDN compatibility. We formulate the multicast routing problem into a mathematical optimization problem and propose a min-max link load multicast routing algorithm, called Robust Multipath Multicast Routing (RMMR*). We further design a heuristic algorithm to obtain the multicast trees efficiently. We implement our proposed algorithm in our multicast routing system on OpenFlow controller. We conduct the experiments in real testbed and Mininet emulator to demonstrate the practicality, performance and scalability. In the experiment of real testbed, we measure the response time of our system: (i) all operations of flow-entries insertion are completed in less than 5 milliseconds, (ii) the detection time of all clients are no more than 0.15 second. In the experiments in Mininet emulator, we compare our algorithms with IP multicast. The results show the merits of our algorithms over the IP multicast, e.g., we observe: (i) frame loss rate reduction between 19% and 95%, (ii) video quality improvement between 4 dB and 15 dB, (iii) sink throughput increase between 25% and 66%, and (iv) maximal link utilization reduction between 15% and 50%. We also show the tradeoff between optimality and running time of the two proposed algorithms: one of them is more suitable for smaller and more static networks, and the other one is more suitable for larger and more dynamic networks.

Contents

中文摘要 i

Abstract ii

1 Introduction 1
- 1.1 Contributions and Organization . . . 3

2 Related Work 5
- 2.1 Multiple Description Coding . . . 5
- 2.2 Software-Defined Networks . . . 5
- 2.3 Multicast in SDNs . . . 6
- 2.4 Multipath Multicast Routing in SDNs . . . 7

3 System Architecture 8
- 3.1 Routing Mechanisms . . . 10

4 Problem Formulations and Proposed Solutions 12
- 4.1 Notations and System Models . . . 12
- 4.2 Multipath Multicast Routing . . . 14
- 4.3 Optimal Algorithm for Robust Multipath Multicast Routing: RMMR* . . . 15
- 4.4 Efficient Algorithm for Robust Multipath Multicast Routing: RMMR . . . 15

5 Implementation 17
- 5.1 Overview on Ryu . . . 17
- 5.2 Enhancement on Ryu to Support Multicast Routing . . . 18
- 5.3 Multicast and Unicast Routing Applications . . . 20

6 Experiments in Real Testbed 22
- 6.1 Setup . . . 22
- 6.2 Results . . . 25

7 Experiments using Mininet 29
- 7.1 Setup . . . 29
- 7.2 Results . . . 31

8 Conclusion and Future Work 43

Bibliography 45

List of Figures

1.1 Sample usage scenarios: (a) video surveillance networks and (b) video conferencing services. 1

3.1 The proposed multicast system architecture overview. 8
3.2 System architecture to support multicast in SDNs. 9

4.1 The RAH heuristics of RMMR. 16

5.1 Architecture overview of Ryu SDN framework. 17
5.2 Component digram of our proposed multicast system 18

6.1 The network topology of the testbed in our Lab. 22
6.2 Our testbed running video streaming experiments. 23
6.3 The videos for streaming server and captured videos from sinks. 24
6.4 The overall mean/max link utilization of streaming of (a) 30 fps video, (b) 60 fps video. 25
6.5 The video packet transmission delay of streaming of (a) 30 fps video, (b) 60 fps video. 26
6.6 The CDF of flow-entry insertion time. 27
6.7 The CDF of IGMP response time of report/leave messages. 28

7.1 A sample network topology and multicast routes generated by the RMMR* algorithm. 30
7.2 The proposed algorithms are bandwidth-aware and lead to almost zero frame loss rate: (a) mean/min/max frame loss rates across all sinks, (b) average frame loss rates of individual sinks, (c) detailed frame loss rates from sink 5, and (d) average frame loss rates from different videos. 32
7.3 The proposed algorithms result in lower maximal link utilization: (a) mean/max link utilization across all links, (b), (c) two bottleneck links under the IPM, and (d) average maximal link utilization from different videos. 33

7.4 The proposed algorithms achieve higher video quality: (a) mean/min/max video quality, (b), (c) detailed video quality of sinks 1 and 5, and (d) average video quality from different videos. 34
7.5 Our proposed algorithms are robust against switch failures: (a) throughput and (b) video quality. 35
7.6 The IPM may result in lower/higher throughput at sinks: (a) mean/min/max throughput, (b), (c) detailed throughput of sinks 1 and 2, and (d) average throughput from different videos. 36
7.7 Implication of initial buffering time on video quality. 37
7.8 The scalability of the proposed algorithms: (a) various numbers of switches, (b) various numbers of video sources, and (c) various numbers of video sinks. 38
7.9 The frame loss rates in the large topology: (a) `Mobile` and (b) all videos. 39
7.10 The video quality in the large topology: (a) `Mobile` and (b) all videos. . 39
7.11 The throughputs in the large topology: (a) `Mobile` and (b) all videos. . . 40
7.12 The tradeoff between RMMR* and RMMR: (a) optimality and (b) running time. 41
7.13 The tradeoff of different period parameter. 42

List of Tables

4.1 Symbols Used Throughout This Paper 13

6.1 Response Time of Flow-Entry Insertion 27
6.2 Response Time of IGMP Packets 27

7.1 Per-sink Video Quality Achieved by Different Algorithms while Link Down 35

Chapter 1

Introduction

High-quality videos have become very popular in recent years, people can easily create, upload, and share the videos through the Internet. A forecast report [5] shows that 80% of network traffic would be video by 2019. Receivers acquire the videos by either downloading or subscribing the streaming from video source over the network. Among all kinds of video traffic around the Internet, real-time streaming imposes strict constraints on network conditions to guarantee the Quality-of-Service (QoS).

Fig. 1.1 shows two usage scenarios of video streaming. In the first scenario, video surveillance networks are deployed in power plants, campuses, or cities, where IP cameras are video sources, whereas client computers and the video server are video receivers. The video server archives the surveillance videos. The users use the portal server to subscribe video(s) from one or multiple IP cameras. In the second scenario, video conferencing services are deployed in campuses, corporates, or inter-campuses, where client computers

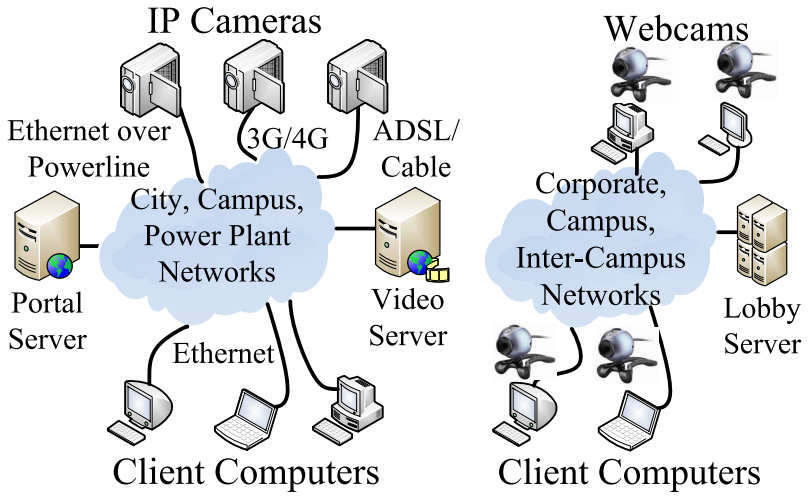


Figure 1.1: Sample usage scenarios: (a) video surveillance networks and (b) video conferencing services.

(including desktops and laptops) are video receivers, and the webcams on them are video sources. The users use client computers to connect to the lobby server, in order to set up conference calls among them. In both usage scenarios, multiple video sources concurrently stream videos to multiple video receivers over a potentially dedicated network, and each video sink may subscribe one or multiple videos.

IP multicast is a well-known method of one-to-many delivery, which reduce the traffic load by transmit only one copy in the network rather than multiple streams of content. However, IP multicast requires high-end router which support distributed protocols such as IGMP, PIM, RIP, MOSPF, and DVMRP. The complicated multicast routing protocols cause too much overhead on the routers [11]. The costly devices and complex operations make the IP multicast hard to deploy with scalability and adaptability. It incurs high maintenance/administrative complexity (high OPEX) and much expense (high CAPEX) while deploy IP multicast.

The emergence of Software-Defined Networks (SDNs), which decouples the control plane and data plane in networks, provides the centralized control and network programmability. Network administrator can manage their infrastructures through high-level functionality. Moreover, SDN architecture is open standards-based and vendor-neutral. OpenFlow is a most popular south-bound API in SDN architecture, which provides vendor-neutral standard communications interface between the control plane and data plane. In SDNs, network providers can realize multicast with lower cost on both CAPEX and OPEX. Through the centralized control of SDNs, it become possible to optimize the routing problems for different objectives with the global view of all network infrastructures. Some of works [22, 25] consider layer-2 multicast in SDNs, but the solutions are not sufficient when multicasting the videos, which have strict deadlines and complex interdependency [14]. Hence, we study video streaming that leverages multipath to deliver multicast streams for higher throughput, lower link utilization, and better error resilience, which in turns leads to higher streaming quality. Multiple description coding (MDC) segments the frame into multiple substreams called descriptors. Any descriptor received can be reconstructed as video with lower quality which is acceptable and all descriptors received with full quality. Enable the video to use multiple description coding would significantly reduce the link utilization over the Internet and avoid the probability of link congestion, which may cause the severe latency and fail to reach the streaming deadline. In our previous work [19], we consider the multipath multicast routing in SDNs and significantly improve the load balancing in link utilization. Based on the multipath multicast routing we proposed, we extend the work and implement the multicast routing system as a regular controller which is able to handle the traffic for both multicast and unicast.

In this thesis, we aim to balance the overhead on the network links causing by streaming in order to avoid the network congestion. Meanwhile, the video quality of receivers should not be degraded. However, there are few challenges to handle the multicast streaming with high performance in the dynamic networks. We list the following features for a desirable multicast system:

1. *Robustness*. The delivery of the video should not be interrupted even the network is not stable.

2. *Load balancing*. The system should balance the overhead among the network resources to avoid any overload on a single switch or link.

3. *Adaptation*. The system has to dynamically adaptive to network changes such as link or switch failure.

4. *SDN Compatible*. There are limited numbers of flow-entries on SDN switches, which is the hardware limitation, so the system must come up with the routes satisfy such constrains.

5. *Reliability*. The system should be able to handle the situations if network failure and rapidly recovery the service of streaming.

1.1 Contributions and Organization

The followings are the contributions of this thesis:

- We design and implement a multipath multicast routing system based on SDN controller which support video streaming with multiple description coding. The system works as a regular SDN controller which are able to support multicast and unicast routing simultaneously.

- We propose the optimized algorithm and corresponding heuristic algorithm in terms of load-balancing on link utilization. The outcome performance of optimized algorithm is better, while the heuristic algorithm is more quickly. We further discuss the situations suitable for each algorithm.

- We build up a real testbed in our Lab, adopt our system to validate the practicality, and proof the performance of our proposed multicast system.

- We conduct extensive emulation experiments to evaluate the performance and scalability of our proposed multicast routing algorithms. The experiment results show

our algorithms outperform the IP multicast in terms of frame loss rate, video quality, sink throughput, and load balancing.

The rest of the thesis is organized as follows. We review the related work in Chapter 2. We present the system overview in Chapter 3. This is followed by the problem statement and our solution in Chapter 4. We then describe the implementation of our system in Chapter 5. We present our real testbed in Chapter 6 and emulation result in Chapter 7. Then we conclude this thesis in Chapter 8.

Chapter 2

Related Work

2.1 Multiple Description Coding

Multipath multicast routing leads to higher network throughput and balanced link utilization, and has been studied in the literature. For example, Rahimi et al. [28] study how to maximize the multicast throughput of several multicast sessions, while the network bandwidth is fairly distributed across multicast sessions. Several distributed round-robin algorithms are proposed, and one of the algorithms achieves up to 90% of the theoretic bound on the throughput. However, their solution is for generic IP multicast without considering the property of real-time coded videos. Tamma et al. [33] study the multipath multicast routing problem in wireless ad hoc networks, which suffer from dynamic network conditions. They employ MDC and route each descriptor over a multicast tree. They consider a single video source, and strive to ensure the multicast trees are mutually vertex-disjoint in order to maximize the robustness of their distributed protocol. Vertex-disjoint trees, although adequate to ad hoc networks, are probably overkills to less dynamic wired networks and may result in degraded network throughput.

The major drawback of the distributed protocols proposed in [28,33] and other similar work is their high maintenance overhead. In contrast, our work leverages SDNs for a logically centralized algorithm that is friendly to network administrators.

2.2 Software-Defined Networks

Software-defined networking (SDN) is an network paradigm emerges in recent years that separates the control logic from the forwarding devices. The concept of OpenFlow is first mentioned in [23]. It proposes to separate the data plane and the control plane of the legacy network devices and use a logically centralized controller to dynamically program the network devices for lower OPEX. Furthermore, it promotes to use off-the-shelf mer-

chant silicon to drive down CAPEX. So far, OpenFlow has attracted great attentions from the network vendors and service providers. Since that traditional IP networks are complicated and hard to manage the extremely increasing devices and services nowadays [6], SDN is considered to be the solution to enable scaling, efficiency, and easier network management. There are more and more network applications in both wired and wireless adopt SDN to improve their services. For example, [21] propose an SDN architecture for Vehicular Ad Hoc networks (VANETs) to reduce control overhead, bandwidth consumption, and latency. [9] propose a framework in SDN for wireless sensor network (WSN) to improve management of energy saving and topology discovery. [18] propose an approach to raise the utilization of networks by enabling traffic engineering of inter-DC WAN.

2.3 Multicast in SDNs

There are some recent papers discuss the multicast in SDNs. Nakagawa et al. [25] and Matias et al. [22] study efficient layer-2 multicast in OpenFlow networks. [25] enables IP multicast in overlay networks using OpenFlow by mapping multicast group in proper VXLAN (Virtual eXtensible LAN). [22] implements the multicast in OpenFlow with Layer 2 prefixes, which provides several virtual networks depending on MAC source and destination. In their works, they realize multicast in SDNs, but there are still rooms to improve the multicast routing by utilizing the global view of SDNs. In [20], they focus on customizing the multicast services and discuss the performance of existing tree constructed algorithms such as Shortest-Path tree and Steiner tree in SDNs.

Moreover, there are some papers design the centralized multicast routing algorithm in SDNs to improve the performance of multicast. Zhao et al. [37] propose a multicast multiparty video conferencing system in SDNs, which aims to reach higher network bandwidth and lower latency. They propose a multicast tree construction and packing method to establish multicast tree to achieve their goal. In [17], they propose the routing algorithm to minimize the size of multicast tree and increase the path diversity to avoid congestion. They focus on networks of data center topology like Fat-tree and tree structure. In contrast, we model the general topology and design the algorithms which are efficient enough to deploy in different environments. Freris et al. [16] propose one Branch-Aware Steiner Tree Algorithm and one corresponding approximation algorithm for multicast routing in SDNs. Their goal is to minimize the numbers of edges and branch nodes in the multicast tree. There algorithms aim to reduce more bandwidth consumption over the network than the Shortest Path tree and become more scalable than traditional Steiner trees. On the other hand, we do not focus on minimize the bandwidth consumption, but utilize MDC for multipath multicast routing to balance link overhead in the networks. In [32], they design

a reliable multicast routing algorithm in SDNs. Their approach is to deploy some recovery nodes over the network, which cache a few packets for recovery. The goal of their routing problem is to minimize the tree cost and recovery cost of multicast tree, called Recover-aware Steiner Tree (RST). They proof that RST problem is NP-Hard then design an approximate algorithm. The objective of our proposed algorithm is not to reduce the recovery cost. However, one of the benefits using MDC to encode video is to provide better error resilience. Additionally, we use the mechanism in our system to fast-recovery the multicast tree while the link or switch in the routes is down.

2.4 Multipath Multicast Routing in SDNs

There are some works leverage multipath to improve the performance of the multicast system. In [30], Julius et al. propose an approach called Software-Defined Multicast (SDM) to enable the Internet Server Providers (ISPs) to provide network layer multicasting support for over-the-top (OTT) live streams. In there later work [29], they propose a mechanism to utilize group table feature in OpenFlow to distribute multicast traffic over multiple trees. They define the edge weight by setting parameters of different metrics such as bandwidth, utilization, delay, loss rate, and failure rate then construct multiple multicast trees by Minimum Spanning Tree (MST) or Shortest Path Tree (SPT). They set multiple action buckets at the first switch of multicast trees and use `select` feature to split the traffic into substreams, which are distributed into individual multicast trees. In contrast, we focus on robust multipath multicast routing using multiple description coding (MDC) to achieve better link load-balancing. In [36], Liao et al. study on implementation of Scalable Video Coding (SVC) multicast streaming in SDNs that satisfied the heterogeneous devices in different video qualities. SVC encodes video into one based layer which provides basic quality and multiple enhancement layers with incremental quality. In contrast, we use MDC in our system to balance link overhead over the networks. Noghani et al. [26] also distribute the multicast videos with MDC. They leverage the benefits of MDC, which provides potential of lower link utilization and better error resilience in transmission. They compare three routing algorithm, respectively for minimum hop, shortest path, and min-max hop. Additionally, we leverage the centralized view of SDN to design the optimal solution and its corresponding heuristic to improve the multicast routing algorithms for link load balancing.

Chapter 3

System Architecture

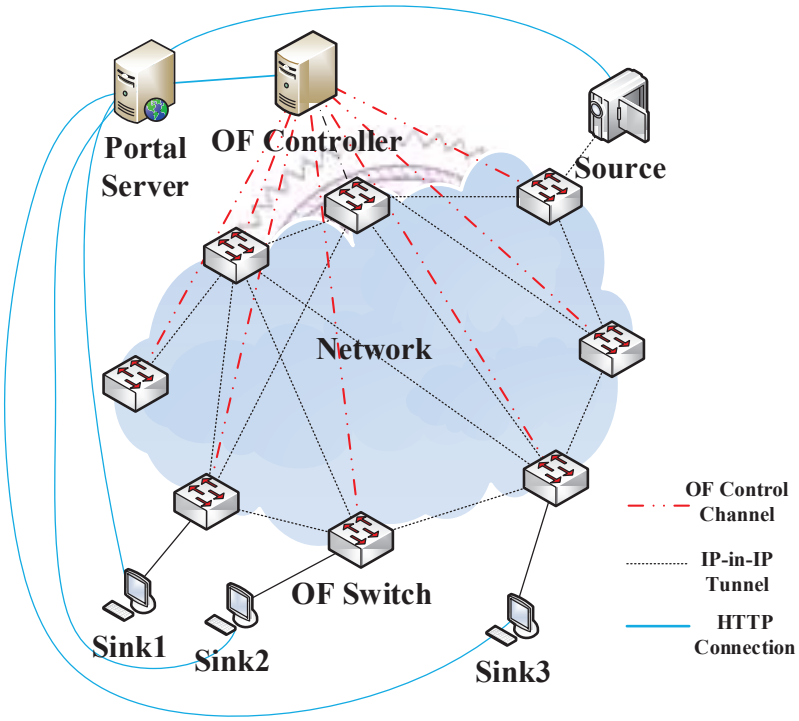


Figure 3.1: The proposed multicast system architecture overview.

We give a high-level system overview of our proposed multicast system in Fig. 3.1. Our system adopts hybrid SDN model, in which the OpenFlow-enabled switch and legacy routers work together enabling incremental deployments. Non-OpenFlow switches are supported in multicast by setting some routing policy to become forwarding devices. All OpenFlow-enabled switches are controlled by a logically-centralized controller build on the OpenFlow controller platforms, such as Ryu [31], Floodlight [13], and NOX/POX [27]. The portal server connects with OpenFlow controller and video sources/sinks through HTTP connections to maintain the group informations.

Fig. 3.2 shows our proposed system architecture which supports multicast in SDNs.

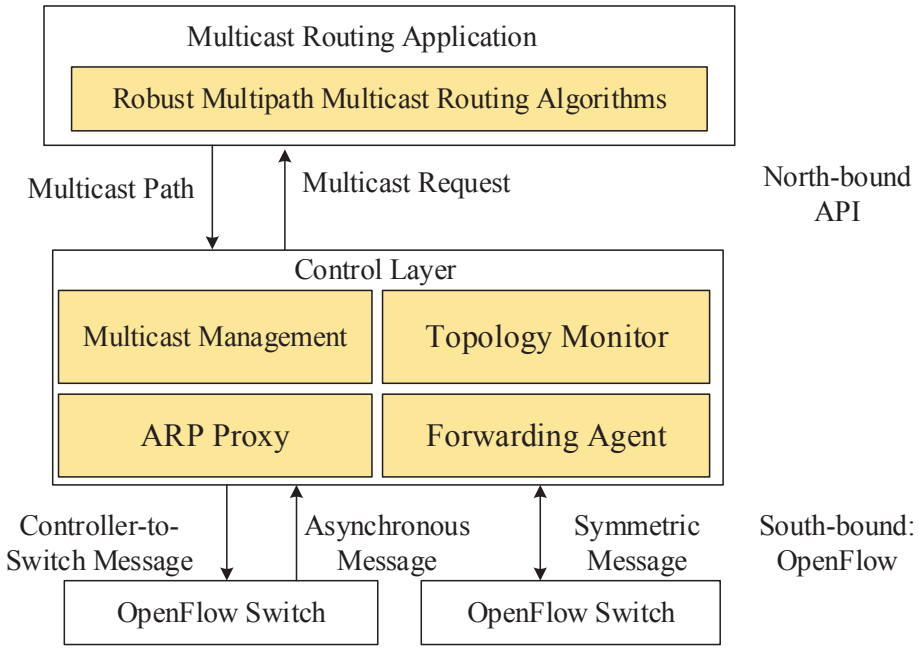


Figure 3.2: System architecture to support multicast in SDNs.

The system includes three layers, which are infrastructure layer, controller layer, and application layer. On the bottom is infrastructure layer, which are OpenFlow-enabled switches. The communications between switches are IP-in-IP tunnels. In the middle is control layer, which consists of the following functionalities: as the following:

- *Topology Monitor*: The component achieve the topology informations such as switch and link location and monitor the traffic in the topology.
- *Forwarding Agent*: The component carries unicast traffic and takes responsibility for establishing end-to-end routes.
- *ARP Proxy*: The component tackles with ARP packet-in, builds up the ARP table, and answers the ARP reply in the system.
- *Multicast Management*: The component collects the informations such as bit rate and group information including video source IP, multicast group IP, and video sink IP. There are two mechanisms to maintain such information. One is to build up portal server with RESTful interface for video sources/sinks to launch/subscribe the video streams. Another is to enable IGMP in the system and capture the IGMP report and leave messages sent by video sources/sinks.

Controller layer communicates with network devices via OpenFlow protocol, which is a south-bound API. The channel between OpenFlow controller and OpenFlow switch is either encrypted using Transport Layer Security (TLS) or TCP. According to OpenFlow

protocol, the communication messages between control layer and infrastructure layer are controller-to-switch, asynchronous, and symmetric message. These messages carry informations or instructions from controller to switch, in conversely, or both directions. On the top of control layer is Multicast Routing Application. We install the multicast routing algorithms here, which takes responsibility for computing the multicast routes. The communications between control layer and application layer is via well-defined north-bound API.

3.1 Routing Mechanisms

We define two routing modes in our system. One is event-driven mode and another is periodic mode. If the event-driven mode is on, the system would update the multicast routes whenever a sink joins/leaves or topology changes. If the periodic mode is on, the routing algorithm would be called periodically to compute new multicast routes. Moreover, we design a fast re-route mechanism to recovery the multicast routes while any port/link/switch down in the current path. There are two situations of port/link/switch down, one is that someone shutdown the port/link/switch. Controller is able to notice this circumstance by setting the handler for such events. The other situation is that the port/link/switch crashed without notification. We can only notice that there is port/link/switch disappeared by periodically check the current topology with previous one. In our system, we detect the such topology change and decide to re-route the video if there is any topology change happened in the current multicast routes.

We use an example here to demonstrate the operations of our proposed system while setting multicast:

1. After the system is launched as an OpenFlow controller, the OpenFlow switches setup the connections with controller.

2. Topology Monitor then periodically achieves topology connection and bandwidth information. ARP Proxy answers the reply if there is any ARP packet-in.

3. When a video streaming server starts to send the video, Multicast Management will detect the group IP with its IP address as a new source.

4. Once there are video sinks subscribe the certain video streaming, Multicast Management will detect sinks joining with their IP addresses. Multicast Management then make the decision whether to pass the multicast request to the Multicast Routing Application depends on the routing mode.

5. Multicast Routing Algorithm then starts to compute multicast routes according to the request input and return the answer to Multicast Management.

6. After the multicast routes is returned, Multicast Management converts the routes into flow-entries and inserts them into corresponding switches to complete the multicasting.

Chapter 4

Problem Formulations and Proposed Solutions

In this chapter, we first define the symbols and system models. We then present the formulation and proposed algorithms.

4.1 Notations and System Models

Table. 4.1 gives the symbols used in this thesis. We consider a video multicast system with S video sources, such as IP cameras, and C video sinks, such as client computers, connected to an IP network. This system is modeled as a directed graph with V vertices and E edges. Each vertex could be a video source, a switch, or a video sink. Each edge represents a network link between two vertices. A video source s ($1 \leq s \leq S$) encodes the captured video in real-time into K descriptors using Multiple Description Coding (MDC) [35], and sends each descriptor over a multicast tree. With MDC, a video sink with one or multiple descriptors can render the received descriptors to its user, and more descriptors lead to higher video quality. The mean bit rate of video s ($1 \leq s \leq S$) is R_s kbps, and the mean bit rate of each descriptor of s is R_s/K kbps. Both K and R_s are system parameters.

Each user uses a video sink to selectively subscribe videos from some of the video sources, and a video sink receives each video from multiple multicast trees for the sake of robustness and load balancing. For similar reasons, each switch may participate in up to $K - 1$ multicast trees from each video source, so that a single switch failure would never break all K multicast trees of a video sink. Hence, all video sinks can receive at least one descriptor for basic video quality in such unfortunate situations, while the controller would quickly update the multicast trees to accommodate the topology change. Each vertex v ($1 \leq v \leq V$) only supports up to f_v active flows at any moment due to hardware

limitations. For each v we use $\boldsymbol{I}_v$ and $\boldsymbol{O}_v$ to denote the ingress and egress edges[1]. Each link e ($1 \leq e \leq E$) has a bandwidth of b_e kbps. Edge e is also denoted as (ϵ_e, ζ_e), where edge e goes from vertex ϵ_e to vertex ζ_e.

We consider the problem of computing the optimal routes for a given network topology, which is solved by an algorithm running on the OpenFlow controller. The optimization goal is to minimize the maximal link utilization across all edges, i.e., in a min-max load balancing fashion. More specifically, our problem is to compute the multicast trees for optimal min-max load balancing, without violating the constraints on: (i) vertices (number of flows per switch), (ii) edges (bandwidth of each network link), and (iii) tree robustness (a switch can participate in up to $K - 1$ multicast trees for each video). The precise formulation is given in the next section.

Sym.	**Description**
S	Number of video sources
s	Index of a video source
C	Number of video sinks
c	Index of a video sink
K	Number of descriptor using MDC
R_s	Mean bit rate of video s
V	Number of vertex in the network
v	Index of a vertex
E	Number of edge in the network
e	Index of an edge
$\boldsymbol{I}_v$	Ingress edges of node v
$\boldsymbol{O}_v$	Egress edges of node v
(ϵ_e, ζ_e)	Another representation of edge e, which goes from vertex ϵ_e to vertex ζ_e
f_v	The flow capacity of vertex v
b_e	The link bandwidth of edge e
σ_s	Vertex index of video source s
$\boldsymbol{\tau}_s$	Vertex set of s's sink, which $\boldsymbol{\tau}_s = \{\tau_{s,t} \mid 1 \leq t \leq \lvert\boldsymbol{\tau}_s\rvert\}$
$x^{s,k}_{v_1,v_2}$	Decision variable of the problem formulation

Table 4.1: Symbols Used Throughout This Paper

[1]Throughout this thesis, we use bold font to represent matrices/vectors.

4.2 Multipath Multicast Routing

We let σ_s ($1 \leq \sigma_s \leq V$) be the vertex index of video source s ($1 \leq s \leq S$). Let $\boldsymbol{\tau}_s = \{\tau_{s,t} \mid 1 \leq t \leq |\boldsymbol{\tau}_s|\}$ be the vertex indexes of s's video sinks. Both σ_s and $\boldsymbol{\tau}_s$ are inputs to our routing algorithm. We define the 0-1 decision variables as $x^{s,k}_{v_1,v_2}$ for $1 \leq s \leq S, 1 \leq k \leq K$, and $1 \leq v_1 \neq v_2 \leq V$, where $x^{s,k}_{v_1,v_2} = 1$ if edge (v_1, v_2) is on the multicast tree of video s's descriptor k; $x^{s,k}_{v_1,v_2} = 0$ otherwise.

With the notations defined above, we mathematically formulate our considered problem as:

$$\text{minimize} \max_{1 \leq e \leq E} \sum_{s=1}^{S} \frac{R_s}{K} \sum_{k=1}^{K} x^{s,k}_{\epsilon_e,\zeta_e} / b_e \tag{4.1a}$$

$$\text{s.t.} \sum_{e \in \boldsymbol{I}_{\sigma_s}} x^{s,k}_{\epsilon_e,\sigma_s} = 0, \ \forall s \in [1,S], k \in [1,K]; \tag{4.1b}$$

$$\sum_{e \in \boldsymbol{O}_{\sigma_s}} x^{s,k}_{\sigma_s,\zeta_e} = 1, \ \forall s \in [1,S], k \in [1,K]; \tag{4.1c}$$

$$\sum_{e \in \boldsymbol{I}_{\tau_{s,t}}} x^{s,k}_{\epsilon_e,\tau_{s,t}} = 1, \ \forall s \in [1,S], k \in [1,K], t \in [1,|\boldsymbol{\tau}_s|]; \tag{4.1d}$$

$$\sum_{e \in \boldsymbol{O}_{\tau_{s,t}}} x^{s,k}_{\tau_{s,t},\zeta_e} = 0, \ \forall s \in [1,S], k \in [1,K], t \in [1,|\boldsymbol{\tau}_s|]; \tag{4.1e}$$

$$\sum_{e' \in \boldsymbol{I}_v} x^{s,k}_{\epsilon_{e'},v} \leq 1, \ \forall s \in [1,S], k \in [1,K], v \in [1,V],$$
$$v \neq \sigma_s, v \notin \boldsymbol{\tau}_s; \tag{4.1f}$$

$$x^{s,k}_{v,\zeta_e} \leq \sum_{e' \in \boldsymbol{I}_v} x^{s,k}_{\epsilon_{e'},v}, \ \forall s \in [1,S], k \in [1,K], v \in [1,V],$$
$$v \neq \sigma_s, v \notin \boldsymbol{\tau}_s, e \in \boldsymbol{O}_v; \tag{4.1g}$$

$$\sum_{e \in \boldsymbol{O}_v} x^{s,k}_{v,\zeta_e} \geq \sum_{e' \in \boldsymbol{I}_v} x^{s,k}_{\epsilon_{e'},v}, \ \forall s \in [1,S], k \in [1,K], v \in [1,V],$$
$$v \neq \sigma_s, v \notin \boldsymbol{\tau}_s; \tag{4.1h}$$

$$\sum_{s=1}^{S} \sum_{k=1}^{K} \sum_{e \in \boldsymbol{O}_v} x^{s,k}_{v,\zeta_e} \leq f_v, \ \forall v \in [1,V]; \tag{4.1i}$$

$$\sum_{s=1}^{S} \frac{R_s}{K} \sum_{k=1}^{K} x^{s,k}_{\epsilon_e,\zeta_e} \leq b_e, \ \forall e \in [1,E]; \tag{4.1j}$$

$$\sum_{k=1}^{K} \sum_{e \in \boldsymbol{I}_v} x^{s,k}_{\epsilon_e,v} \leq K - 1, \ \forall s \in [1,S], v \in [1,V]. \tag{4.1k}$$

$$x^{s',k'}_{v_1,v_2} \in \{0,1\}, \ \forall v_1, v_2 \in [1,V], s' \in [1,S], k' \in [1,K]. \tag{4.1l}$$

The objective function in Eq. (4.1a) minimizes the maximal edge utilization. The constraints in Eqs. (4.1b)–(4.1e) ensure that the video sources and sinks are indeed the roots

and leaves of the multicast trees. The constraints in Eq. (4.1f) allow the intermediate vertices to avoid receiving duplicated descriptors from multiple links, and the constraints in Eq. (4.1g) ensure that a vertex sends a descriptor only if it's received, and the constraint in Eq. (4.1h) ensures that a vertex must send a descriptor once it's received. The constraints in Eqs. (4.1i), (4.1j), and (4.1k) impose limitations due to switch capability, link capacity, and tree robustness, respectively.

4.3 Optimal Algorithm for Robust Multipath Multicast Routing: RMMR*

The formulation Eq. (4.1) is an Integer Programming (IP) problem and can be solved using commercial solvers, such as IBM CPLEX [8]. We refer to the optimal algorithm as RMMR*. However, the formulation does not prevent cycles, the solution may result in multicast trees with cycles. We adopt the following approach [15, 34] to eliminate cycles. Upon the RMMR* algorithm produces the multicast trees, we check whether there exists any cycles. For each cycle with length l, we generate l new formulations, where each formulation explicitly prohibits a link of that cycle from being part of the corresponding multicast trees. We then solve all the l formulations again, and check if there are additional cycles. For any remaining cycles, we again generate new formulations with explicit constraints to get rid of the cycles. Once we solve all the formulations, we return the solution with the best objective function value. We have integrated this cycle elimination approach in the RMMR* algorithm.

4.4 Efficient Algorithm for Robust Multipath Multicast Routing: RMMR

Although the RMMR* algorithm gives the optimal routes, running it on a large network too often incurs excessive computational overhead. Such negative impact is amplified when the network condition largely remains the same: the new optimal routes will be very similar to the old ones. Hence, in such situations, some lightweight route adaptation heuristics may be applied to quickly *revise* the existing multipath multicast trees for a reasonably good solution. We call these heuristics Route Adaptation Heuristics (RAHs), and we propose two such heuristics for video sink dynamics. When a video sink joins, for each descriptor, we perform Breadth-First Search (BFS) from the sink to find the shortest path to reach the existing multicast tree for that descriptor. When traversing through new vertices, the heuristic makes sure that the constraints in our formulation are

satisfied. Upon reaching the existing tree, we then search for all shortest paths with some length. Among these shortest paths, we select one which maximal link utilization along the path is minimize and attach this path to the multicast tree of that descriptor. When a video sink leaves, for each descriptor, we traverse from the sink toward the source, and we remove an edge from the multicast tree if there is no video sink in the edge's down-stream. The heuristic ends after the first edge is kept, because it has at least one down-stream video sink. These two RAHs, as summarized in Fig. 4.1, impose very low computational complexity. We integrate them with RMMR* for an efficient RMMR algorithm, which: (i) runs RAHs when video sink joins/leaves for fast route adaptation and (ii) calls RMMR* every P minutes for optimal multicast trees, where P is a system parameter.

```
1: function JOIN(video sink c, video s)
2:     for all descriptor k = 1, 2, ..., K do
3:         perform BFS from c for the paths with shortest distance to tree k
4:         add the shortest path with minimum maximal link utilization to tree k
5: function LEAVE(video sink c, video s)
6:     for all descriptor k = 1, 2, ..., K do
7:         loop switches from c to the root of tree k
8:             reduce the count of down-stream video sinks
9:             remove the switch with 0 count from tree k
```

Figure 4.1: The RAH heuristics of RMMR.

Chapter 5

Implementation

We present the detailed implementation of our proposed multicast routing system in this chapter. Our proposed system works as a regular OpenFlow controller, which is able to handle the traffic for both multicast and unicast.

5.1 Overview on Ryu

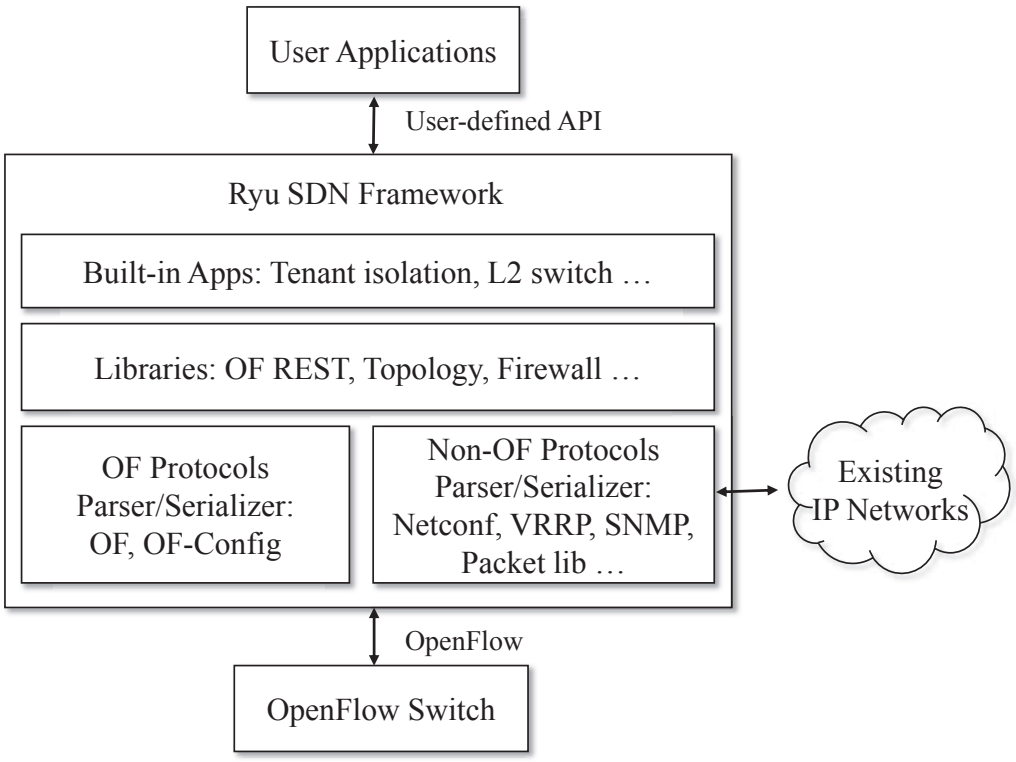


Figure 5.1: Architecture overview of Ryu SDN framework.

Our multicast routing system is implemented in SDN controller on top of Ryu. Ryu is an open source software written in Python. Ryu provides the component-based framework

with well defined API for customers to develop their own SDN applications. Ryu supports various protocols for managing network infrastructures includes OpenFlow, Netconf and OF-config. Fig. 5.1 shows the overview of Ryu SDN framework. There are some built-in applications and libraries in Ryu such as tenant isolation, L2 switch, OF REST, topology discovery, firewall, etc. For example, we can directly achieve the topology connection between OpenFlow switches through the library topology discovery in Ryu. There are also OpenFlow protocols and Non-OpenFlow protocols implemented in Ryu for user to communicate with network devices. Developers are able to combine/modify the existing components or add new components to build up their own applications.

5.2 Enhancement on Ryu to Support Multicast Routing

In this section, we describe the detailed component implementation in our system. We modify some existing components in Ryu then build up our multicast routing system with five components and two applications. As illustrated in Fig. 5.2, the five components are Event Dispatcher, Topology Monitor, ARP Proxy, Multicast Management, and Forwarding Agent. Ryu use the term `event` as the message between entities.

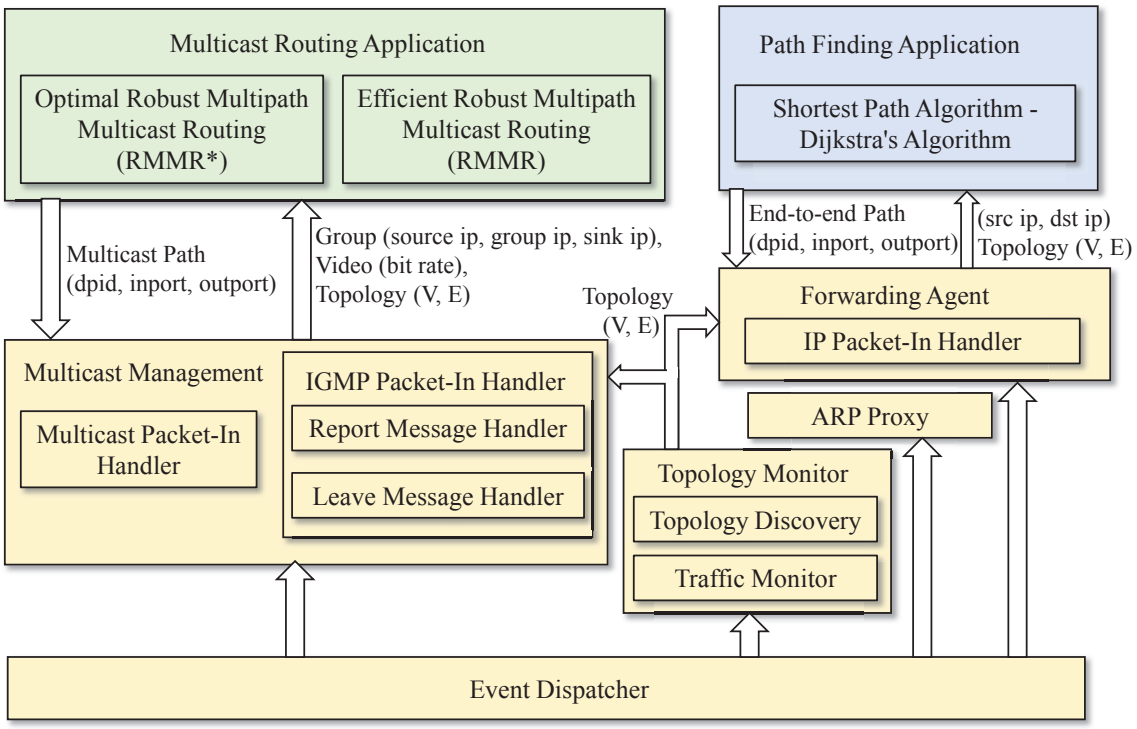


Figure 5.2: Component digram of our proposed multicast system

- **Event Dispatcher.** We implement all the event handler required in the system at this component. Event Dispatcher collects the events from switches and then dispatch them into the corresponding components. For example, the IGMP/multicast packet-in messages would be passed to Multicast Management; IP packet-in messages other than multicast packets would be passed to Forwarding Agent; and ARP packet-in message would be passed Topology Discovery and ARP Proxy.

- **Topology Monitor.** In order to acquire the information required by multicast routing, there are two main function implemented in Topology Monitor, Topology Discovery and Traffic Monitor. We extend the Topology Discovery which is already in implemented in Ryu. Ryu implement their topology discovery using Link Layer Discovery Protocol (LLDP), which is a layer 2 protocol. LLDP is used by network devices to advertise informations such as MAC address, chassis id, port id and so on with their neighbors. By collecting these informations from switches, controller is able to build up a topology.

  The original Topology Discovery only discover the link and switch, we also discovery the host location by ARP packet-in message to complete the topology information. Meanwhile, in Traffic Monitor, we achieve the current link utilization by link bandwidth in `curr_speed` of `OFPMP_PORT_DESCRIPTION` and link consuming in `rx_byte` of `OFPMP_PORT_STATS`. By periodically running these procedure, we achieve the latest topology information includes network connection, link utilization, and host location.

- **ARP Proxy.** We implement a ARP proxy in our system to answer the ARP reply. This component listens on ARP packet-in messages and build up an ARP table. The ARP table records the MAC address and the forwarding port for each switch while receiving corresponding destination IP address. If the ARP destination is in the ARP table, ARP Proxy then answer the ARP reply directly. Otherwise, the component instructs the switch to flood the ARP message to other switches.

- **Multicast Management.** This component collects the multicast groups and video information required for multicast routing. We maintain the multicast source by listening on multicast packets-in messages. Once the multicast packet-in handler receive the multicast packet with valid destination IP, we add the source with its IP address and multicast destination IP. If there is no other sink subscribe this destination IP, we set a drop action on the switch. Then we achieve the value of video bit rate from the `byte_count` of `flow_state_reply` for certain multicast IP address. We maintain the multicast sink information by supporting Internet Group Management Protocol (IGMP). The module listens on Report/Leave messages of

IGMPv2 [12] to add/delete the sink from certain multicast group. There are two mechanism for multicast routing, periodic mode and event-driven mode. If the periodic mode is on, the Multicast Management would call Multicast Routing Application periodically to setup the multicast routing. Otherwise, Multicast Routing Application would be triggered whenever there is any source/sink change or topology change. The component then pass the multicast group information, topology information, and video information as the request to Multicast Routing Application and ask for the routes. While Multicast Routing Application returns the answer of multicast routes, Multicast Management then converts the multicast routes into flow entries by setting the action with multiple outgoing ports and inserts it into corresponding switches to complete the multicasting.

- **Forwarding Agent.** This component maintains all the unicast routes. It carries the traffic of IP packet-in other than multicast packets then call Path Finding Application and get the end-to-end route for the source/destination pair. After that, Forwarding Agent inserts the flow-entries into switches to setup the unicast routing.

5.3 Multicast and Unicast Routing Applications

In this section, we describe the multicast and unicast application in our system.

- **Multicast Routing Application.** This application takes responsibility for computing the multicast routes. We implement two robust multipath multicast routing algorithms (RMMR*/RMMR) described in Chapter 4. The application takes the following inputs from Multicast Management: (i) group information including source IP address of video source, multicast group destination IP, and all the sinks IP addresses which subscribe the certain multicast group. (ii) video information, which is the estimated bit rate of each multicast streaming. (iii) topology information including vertices, which are switches and hosts and their flow capacities and edges, which are the network links and their link bandwidth/consuming. Once the Multicast Routing Application is called, the informations would be updated. The application maintains one instance of multicast routing information, including the data just mentioned and the current multicast routes. This application then chose one algorithm to compute the multicast routes. The optimal Robust Multipath Multicast Routing algorithm (RMMR*) is realized in python and CPLEX [8]. The efficient Robust Multipath Multicast Routing algorithm (RMMR) is written in python. Once the algorithm achieves the new multicast trees, the application then saves the new

answer and compares it with the previous multicast trees to get the modified multicast branches. Finally, the application return these modified branches to Multicast Management to setup the routes.

- **Path Finding Application.** This application takes responsibility for computing the end-to-end path for unicast. Path Finding Application takes input of source IP, destination IP address and topology information. We implement Dijkstra's algorithm [10] and set the weight of the edge to be inverse of link bandwidth. Dijkstra's algorithm computes the shortest path between the source and destination. After that, the application return this end-to-end path to Forwarding Agent to setup the unicast route.

Chapter 6

Experiments in Real Testbed

6.1 Setup

We present the real experiment running on physical machine with our multicast routing system in this chapter. We set up a testbed in our Lab with a small network topology to demonstrate the practicality of our proposed system. Fig. 6.1 illustrates the testbed setup, which consists of 4 OpenFlow switches, 1 OpenFlow controller, 1 layer 2 switch and 4 hosts for video server and clients.

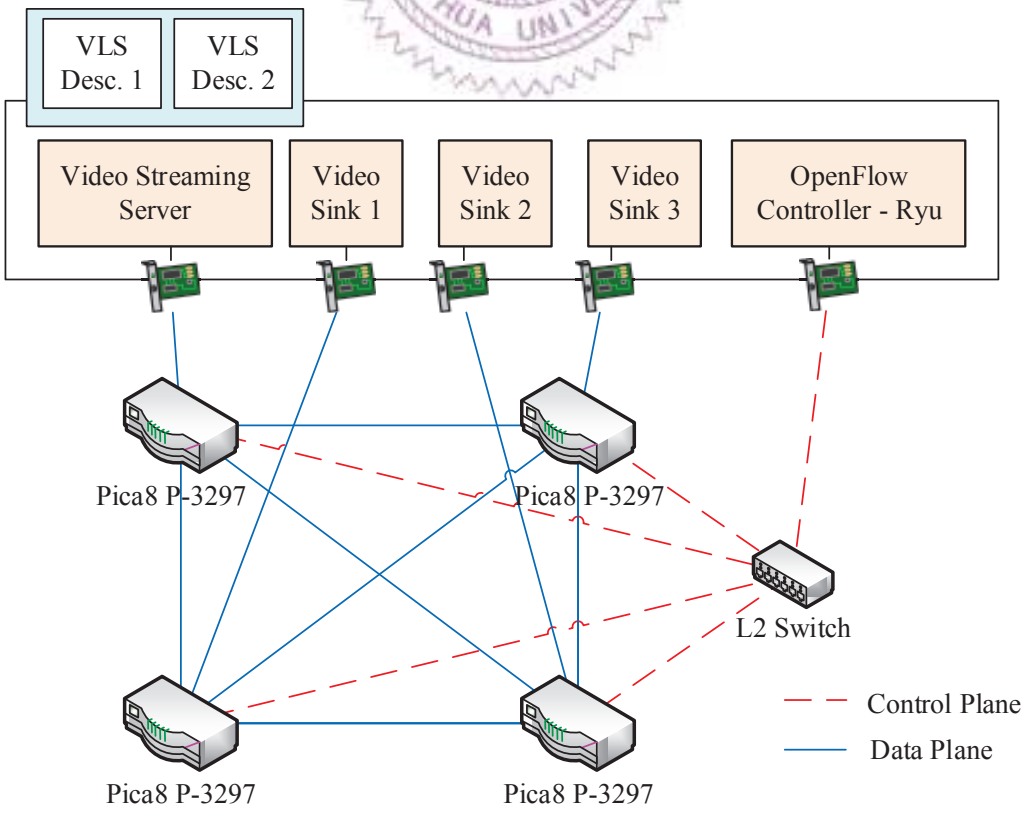


Figure 6.1: The network topology of the testbed in our Lab.

4 OpenFlow switches are Pica8 P-3297 installed PicOS 2.4 and running in OVS mode. We install the bridge in secure fail-mode and connect it to the specific controller IP address, then turn on the interfaces we need on to the bridge. For configuring the link bandwidth, we set the `link_speed` feature on interface by OVS command. The OpenFlow controller runs in a virtual machine installed Ryu and our proposed multicast application. The ethernet port of virtual machine is bridged to the physical ethernet port. We use out-of-band control management and connect the controller with all Pica8 switches' management port through a layer 2 switch for saving port occupied of controller. The video server and clients are running in virtual machines and connect to the corresponding port of each switch. We use VideoLAN Server (VLS) and VideoLAN Client (VLC) to stream and capture the video streams, which are open source tools and can be found in [3]. We stream 2 videos from streaming server simultaneously to represent 2 descriptions of one video. Fig. 6.1 is the picture of our testbed in our Lab.

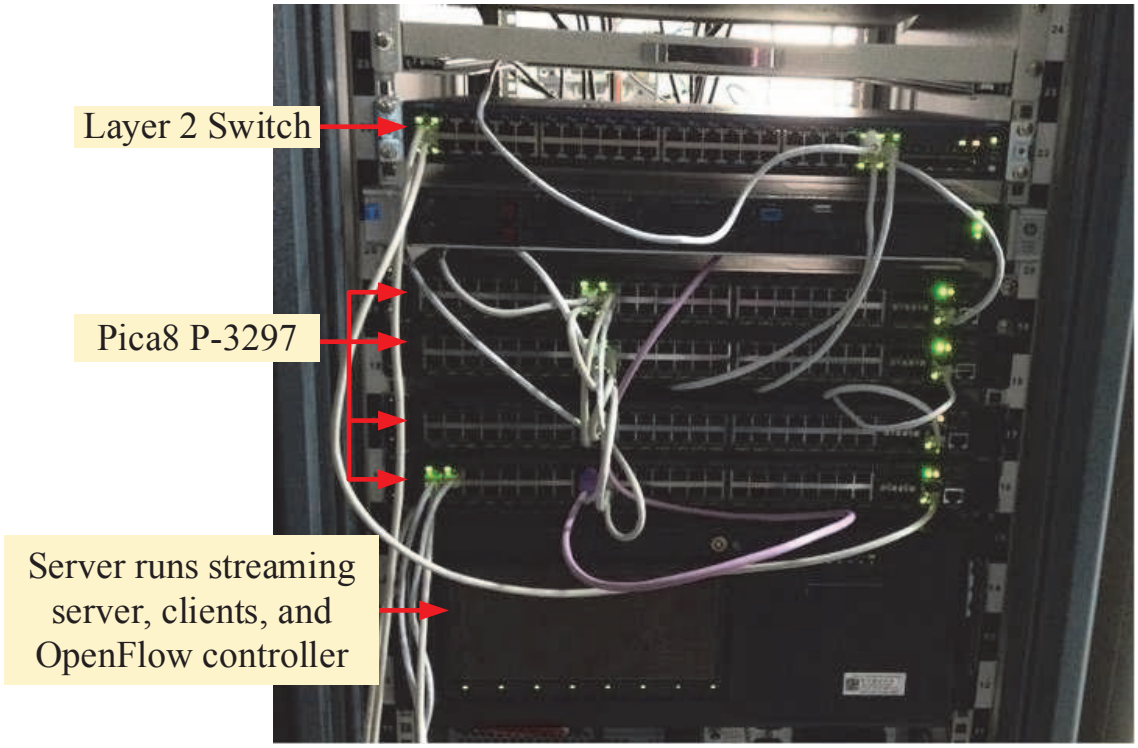


Figure 6.2: Our testbed running video streaming experiments.

To evaluate the performance of our proposed system and algorithms, we conduct the following experiments. We stream 2 1920x1080 videos with 30 fps and 60 fps (frame-per-second) [7]. The bit rate of videos are 3Mbps and 4Mbps. We set the bandwidth of links between switches as 10Mbps. The bandwidth of links connect to the hosts are

1Gbps. Video streaming server streams videos in MPEG Transport Stream format. Clients capture the videos by converting the RTP multicast streaming into MPEG-4. We run our multicast system in event-driven mode with RMMR* and RMMR algorithms. That is, RMMR algorithm runs RAH when the video sink joins or leaves except the first round.

Fig. 6.3 shows the streaming videos and the captured videos from sinks of testbed experiment.

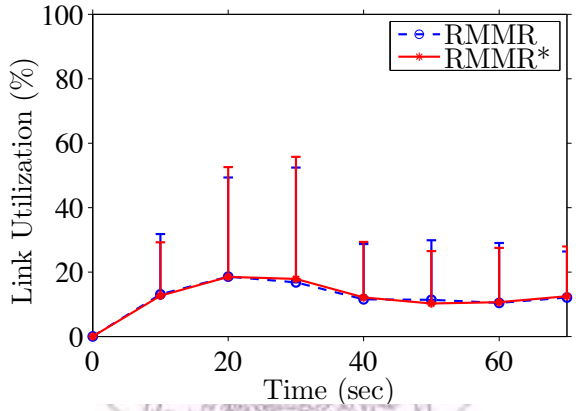
Figure 6.3: The videos for streaming server and captured videos from sinks.

We consider the following performance metrics in the testbed experiment:

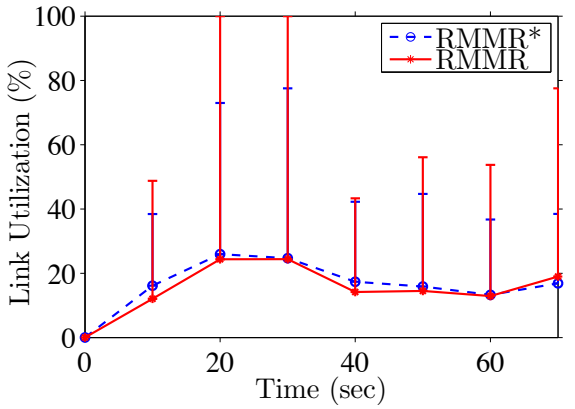
- *Link utilization*. We achieve the link utilization of each link from Topology Discovery module of our multicast system.
- *Video transmission latency*. We capture the video packets from the interface of each host to measure the transmission latency of video.
- *Flow insertion time for multicasting*. We send `barrier request` from controller to switch right after the multicast flow-entry insertion message is sent and listen to the `barrier reply` with corresponding `xid`. By collecting these two types of message, we measure the time cost for switch to insert the multicast flow-entry.
- *Controller response time*. We capture the IGMP packets from the interface of each host and record the time that the controller receive the IGMP packet-in messages to measure the time cost for our system to detect the sinks join and leave.

To evaluate the overall time cost of our multicast system, we divide all processes of multicasting into three parts: i) controller response time, ii) algorithm running time, and iii) flow insertion time. We measure the controller response time and flow insertion time in our testbed and evaluate the algorithm running time for different algorithms in next section.

6.2 Results



(a)



(b)

Figure 6.4: The overall mean/max link utilization of streaming of (a) 30 fps video, (b) 60 fps video.

RMMR* prevents high link utilization. We launch the OpenFlow controller with our multicast system to let switches connect to it. Then we multicast two videos with multicast IP addresses `224.1.1.1` and `224.1.2.1` simultaneously from streaming server. Then Sink 1, Sink 2, and Sink 3 capture the multicast videos from these two IP addresses. We plot the overall mean/maximal link utilization of streaming 2 different videos in Fig. 6.4. In the case of 30 fps video, which is illustrated in Fig. 6.4(a), both

algorithms are able to control the overall maximal link utilization less than 60%. The difference of the utilization of RMMR and RMMR* is less than 3% in this case. The data rate of the video is higher then average between [20, 30) and reduces at 40 sec. In the other case of 60 fps video, illustrated in Fig. 6.4(b), the maximal link utilization of the RMMR is up to 100% in [20,30), while the RMMR* can reduce maximal link utilization to 78%.

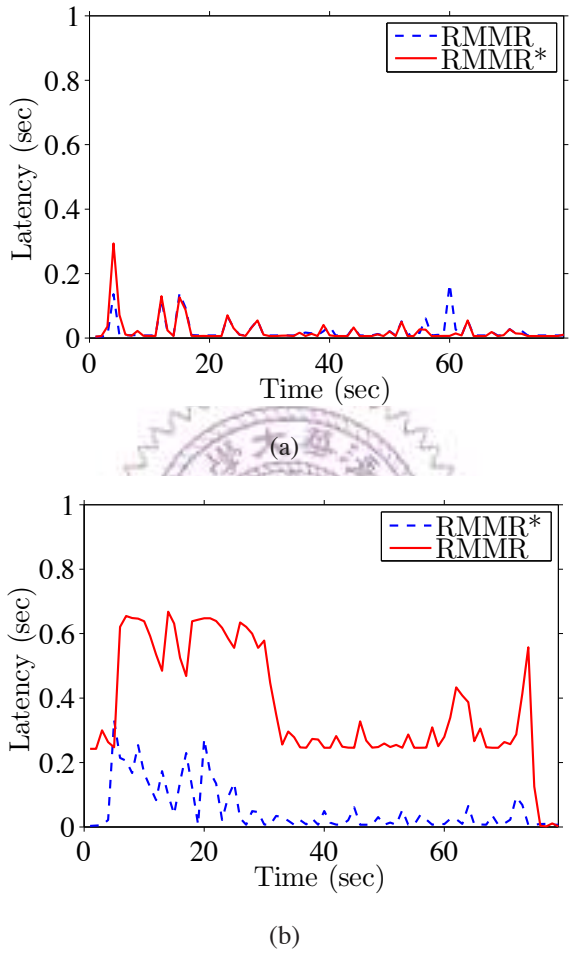


Figure 6.5: The video packet transmission delay of streaming of (a) 30 fps video, (b) 60 fps video.

High link utilization cause longer video latency. We also plot the overall video transmission delay in Fig. 6.5. Fig. 6.5(a) shows that all packets of 30 fps video are delivered to the destination within 0.3 sec. In Fig. 6.5(b), the case of 60 fps video, the packets latency of RMMR is up to 0.6 sec between (0, 30]. Correspond to the maximal link utilization in Fig. 6.4(b), it shows that there are links overloaded which cause the link congestion and longer video latency. While the latency of RMMR* increases a little in first 30 sec, but it still no longer than 0.3 sec.

Our multicast system incurs short flow install time. We plot the CDF of flow-

entries insertion time in Fig. 6.6. Next, we list the average flow-entry insertion time of each switch in Table. 6.1. It shows that all the switches are able to finish flow-entry insertion within only 5 milliseconds.

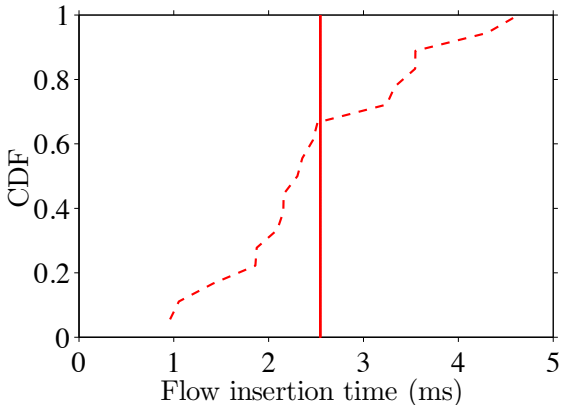


Figure 6.6: The CDF of flow-entry insertion time.

Switch	Response Time (ms)
Switch 1	1
Switch 2	3.6
Switch 3	2.9
Switch 4	3.7
Average	2.5

Table 6.1: Response Time of Flow-Entry Insertion

Our multicast system incurs short response time for sinks join/leave. We plot the CDF of IGMP messages response time in Fig. 6.7 and then we list the average IGMP packet response time of our multicast system in Table. 6.2. It is the time cost for our system to response the sinks join or leave. The Fig. 6.7 shows that the response time for both packets are all less than 0.15 sec. The average response time is 0.071 sec for IGMP Report and 0.084 sec for IGMP Leave.

Host	IGMP Report (sec)	IGMP Leave (sec)
Client 1	0.087	0.104
Client 2	0.07	0.062
Client 3	0.056	0.085
Average	0.071	0.084

Table 6.2: Response Time of IGMP Packets

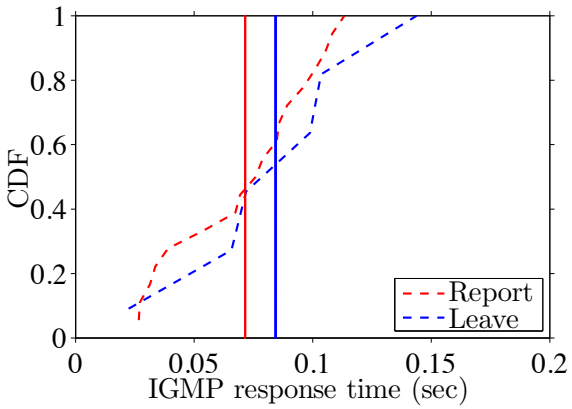


Figure 6.7: The CDF of IGMP response time of report/leave messages.

Chapter 7

Experiments using Mininet

We evaluate the performance and scalability of our proposed multicast system since that the testbed presented in Chapter 6 is fairly small, which is not sufficient for conducting large scale experiments. We adopt Mininet [24], an emulator, in this chapter to conduct the experiment.

7.1 Setup

We adopt Mininet to emulate large OpenFlow networks, which are hard to set up as real testbeds. We instruct Mininet to read a *manifest* file that describes the network topology, including vertices and edges, and the network conditions, such as link bandwidth, network latency, and packet loss rate. The switches in Mininet is configured as `OVS Kernel Switch` and support OpenFlow 1.3 protocol. We setup the link conditions by enabling the `TCLink` (Traffic Control Link) in Mininet. Each vertex in the is either a video source, a video sink, or a switch; all switches are connected to a controller running our multicast system. For comparisons, we also set up IP multicast in Mininet using XORP [4], and denote its results as IPM. We configure XORP to run PIM-SM (Protocol Independent Multicast Sparse Mode). PIM-SM builds the multicast tree from source to receivers based on the topology information supplied by RIP (Routing Information Protocol).

We implement a topology generator script to create the manifest files with different numbers of video sources, video sinks, switches, and descriptors. The topology generator also takes several settings to randomly create link conditions. If not otherwise specified, the following settings are used in our experiments: (i) the link bandwidths are uniformly chosen between 1 and 5 Mbps, (ii) the link delays are uniformly chosen between 5 and 25 ms, (iii) the link packet loss rates are set to zero, (iv) the sink leave/join rates are 2 per minute, and (v) the sink initial buffering time is 1 sec. Two randomly generated manifest files are used throughout the experiments: (i) a small topology with 1 source, 9 switches,

6 sinks, and 2 descriptors, and (ii) a large topology with 4 sources, 24 switches, 12 sinks, and 4 descriptors. Fig. 7.1 shows the small topology with sample optimal routes computed by RMMR*. We consider four videos [2]: `Mobile`, `Highway`, `Paris`, and `Tempete`, with 2 to 16 descriptors. In our experiments, we stream each video with 2 descriptors in the small topology, and stream all four videos with 4 descriptors in the large topology.

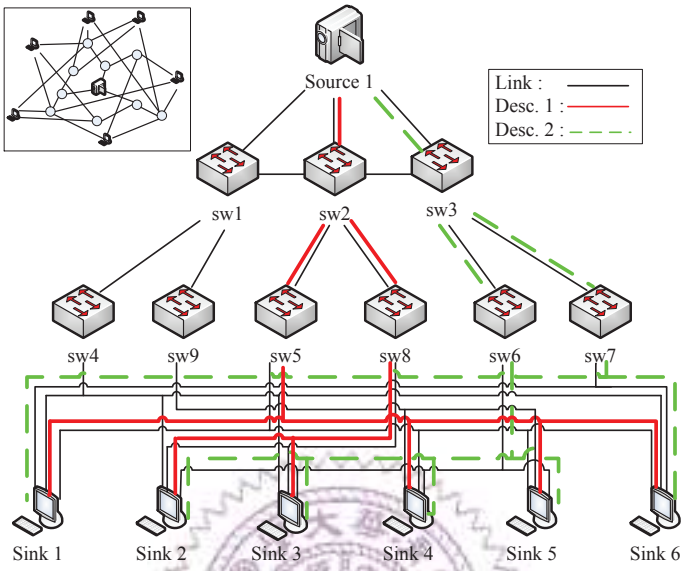


Figure 7.1: A sample network topology and multicast routes generated by the RMMR* algorithm.

We have also implemented two utilities for experiments in Mininet: the *sender* and *receiver* using Python for UDP video streaming. The sender multicasts video packets following the video traces from Arizona State University [2]. The destination multicast IP addresses are uniquely determined based on the video v ($1 \leq v \leq V$) and descriptor k ($1 \leq k \leq K$), in particular, the sender uses `224.1.v.k` as the destination IP address of descriptor k of video v. The receiver receives the UDP packets belonging to the correspondent video v, e.g., with IP addresses of `224.1.v.*`. Both sender and receiver save packet-level logs with timestamps for performance analysis by Tcpdump [1].

We consider the following performance metrics:

- *Frame loss*. The ratio of lost video frames at each video sink. A frame is considered lost if any of its packet gets lost.
- *Throughput*. The received data rate at each video sink.
- *Link utilization*. The load of each link.
- *Video quality*. We employ Peak Signal-to-Noise Ratio (PSNR) [35] as the quality metric. The video frames missing their playout deadlines are not decodable and the

last received frames are replayed to conceal undecodable frames. We also conceal the frame losses using the same approach.

- *Running time*. The execution time of the RMMR* and RMMR algorithms.

We conduct 10-min experiments with the RMMR*, RMMR, and IPM. We set our system to use periodic mode in all Mininet experiments and set the interval as 1 minutes. If not otherwise specified, we set P = 10 minutes. That is, RMMR$*$ and RMMR would be triggered to compute the multicast routes every minute. RMMR computes optimal solution at first minute, then update the routes by RAH in the remaining time. We collect the performance of each video sink (or link). We also calculate the average performance across all video sinks (or links) over time. Last, we compute the overall performance of each experiment over the 10-min duration. In the figures and tables, we give min/max intervals whenever applicable.

7.2 Results

Bandwidth-awareness of our proposed algorithms. We multicast each of the four videos in the small topology. We present: (i) detailed results from `Mobile` and (ii) overall results from all four videos in Fig. 7.2. In particular, we plot the mean/min/max per-min frame loss rates of all sinks receiving `Mobile` in Fig. 7.2(a), which shows that the proposed RMMR* and RMMR algorithms result in almost zero frame loss rates among all sinks. In contrast, the IPM leads to nontrivial and diverse frame loss rates: between 0% and 60%. We next plot the 10-min average frame loss rate of individual sinks in Fig. 7.2(b), which reveals that, with the IPM, sinks 1 and 5 suffer from high frame loss rates. We zoom into sink 5 and plot its per-sec frame loss rate in Fig. 7.2(c), which shows that, with the IPM, sink 5 experiences very high frame loss rates when network is congested. Such inferior performance of the IPM can be attributed to the fact that IP multicast does *not* take the available link bandwidths into consideration when constructing multicast trees. Therefore, some links may become bottleneck links with high packet loss rates, which lead to high frame loss rates. We will plot the link utilization of these bottleneck links below. Last, we plot the average frame loss rates in Fig. 7.2(d). This figure shows that the IPM suffers the most under `Mobile` and `Tempete`. A closer look indicates that the mean bit rates of the four 2-descriptor videos are: 1.27 Mbps (`Mobile`), 0.92 Mbps (`Tempete`), 0.51 Mbps (`Paris`), and 0.20 Mbps (`Highway`). The diverse bit rates explain why the `Mobile` and `Tempete` result in much higher frame loss rates under the IPM. Our proposed algorithms, however, are bandwidth-aware and work for all four videos.

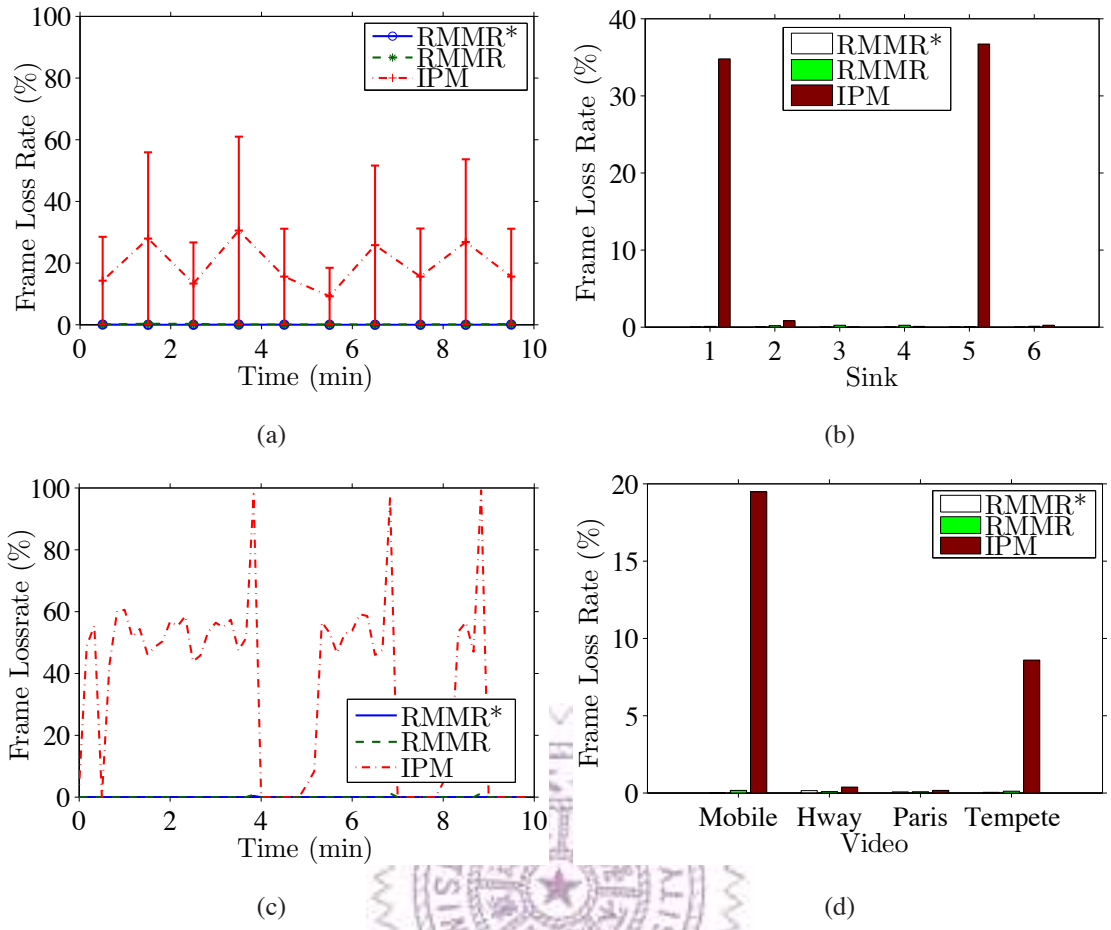


Figure 7.2: The proposed algorithms are bandwidth-aware and lead to almost zero frame loss rate: (a) mean/min/max frame loss rates across all sinks, (b) average frame loss rates of individual sinks, (c) detailed frame loss rates from sink 5, and (d) average frame loss rates from different videos.

Our algorithms balance the link utilization. We analyze the link utilization of multicasting each of the four videos in the small topology. The idling links with zero utilization are not reported in the figures. We first plot the per-min mean/max link utilization from `Mobile` in Fig. 7.3(a). This figure shows that the RMMR* and RMMR algorithms result in much lower mean and maximal link utilization, compared to the IPM. This can be partly attributed to the multi-tree design of RMMR*/RMMR, which provides room for load balancing. Moreover, the RMMR*/RMMR algorithms systematically reduce the maximal link utilization by using it as the objective function in Eq. (4.1), which further capitalizes the optimization room for better load balancing. Next, we take a closer look at the per-sec link utilization of individual links, and we identify the two bottleneck links that cause the high frame loss rates of sinks 1 and 5 (see Fig. 7.2(b)). We plot their per-sec link utilization over time in Figs. 7.3(b) and 7.3(c), which correspond to link (sink 1, switch 4) and (switch 2, switch 5) in Fig. 7.1. These two figures demonstrate the high loads caused

by the IPM: as high as 90% link utilization, which explain the high frame loss rates of sinks 1 and 5. Last, we report the average maximal link utilization in Fig. 7.3(d). This figure shows that, compared to the IPM, our proposed RMMR* and RMMR algorithms reduce the maximal link utilization by half for `Highway`, `Paris`, and `Tempete`.

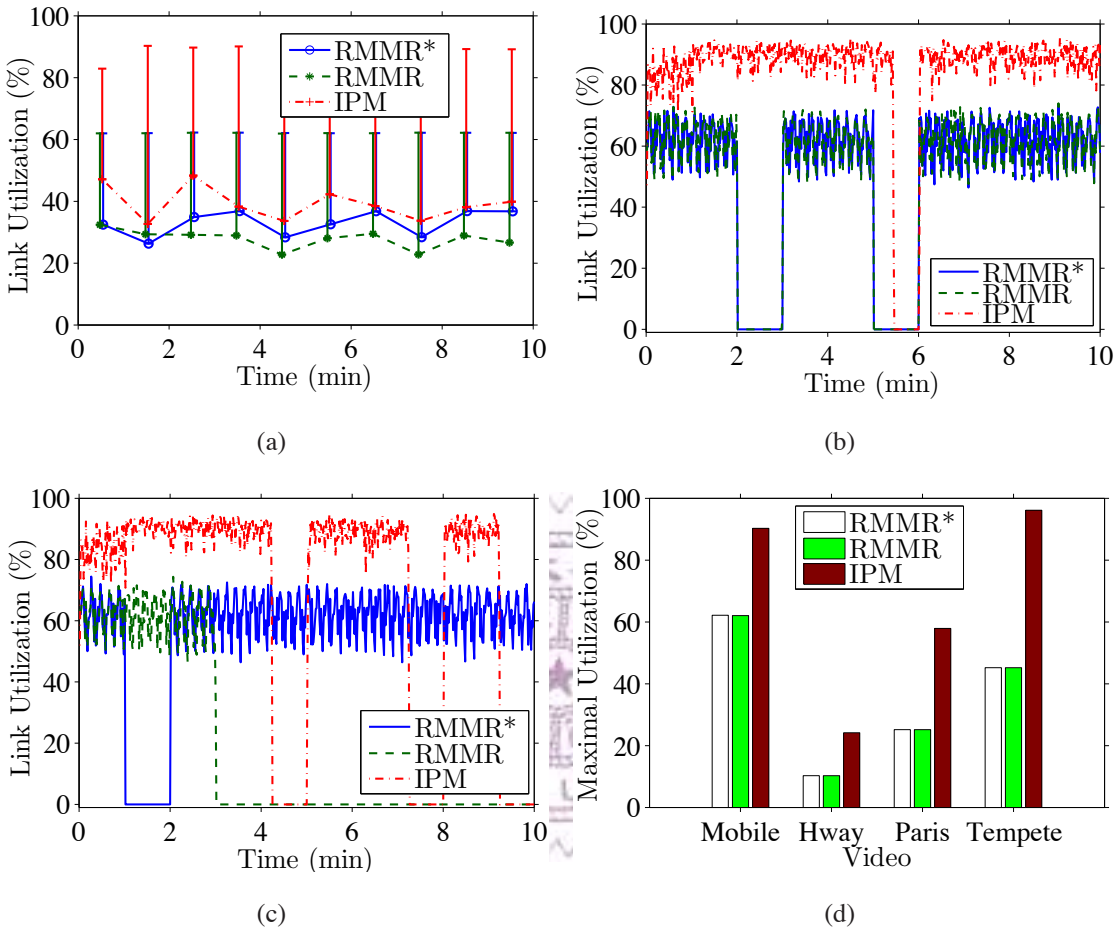


Figure 7.3: The proposed algorithms result in lower maximal link utilization: (a) mean/max link utilization across all links, (b), (c) two bottleneck links under the IPM, and (d) average maximal link utilization from different videos.

Our algorithms achieve higher video quality. Next, we compute the video quality in PSNR after concealing the lost and late frames. We report the results from the small topology. We first plot the per-min mean/min/max video quality of all sinks receiving `Mobile` in Fig. 7.4(a). This figure shows that the RMMR* and RMMR algorithms constantly achieve high video quality, while the IPM suffers from lower video quality: about 5 dB different in terms of mean video quality is observed. We also plot the per-sec video quality of sinks 1 and 5 in Figs. 7.4(b) and 7.4(c). Notice that, sink 1 leaves the multicast groups between [2, 3) and [5, 6) mins, and sink 2 leaves between [4, 5), [7, 8), and [9, 10) mins. These two figures show that the IPM leads to quality fluctuations: at most 10 dB, which greatly degrade the viewing experience. Last, we plot the average video quality of

individual videos in Fig. 7.4(d). We observe non-trivial gaps between our proposed algorithms and the IPM with `Mobile` (4dB) and `Tempete` (1dB). There are no clear gaps for `Highway` and `Paris` because their bit rates are much lower and thus do not overload the bottleneck links. Fig. 7.4 reveals that our proposed algorithms achieve higher video quality than the IPM.

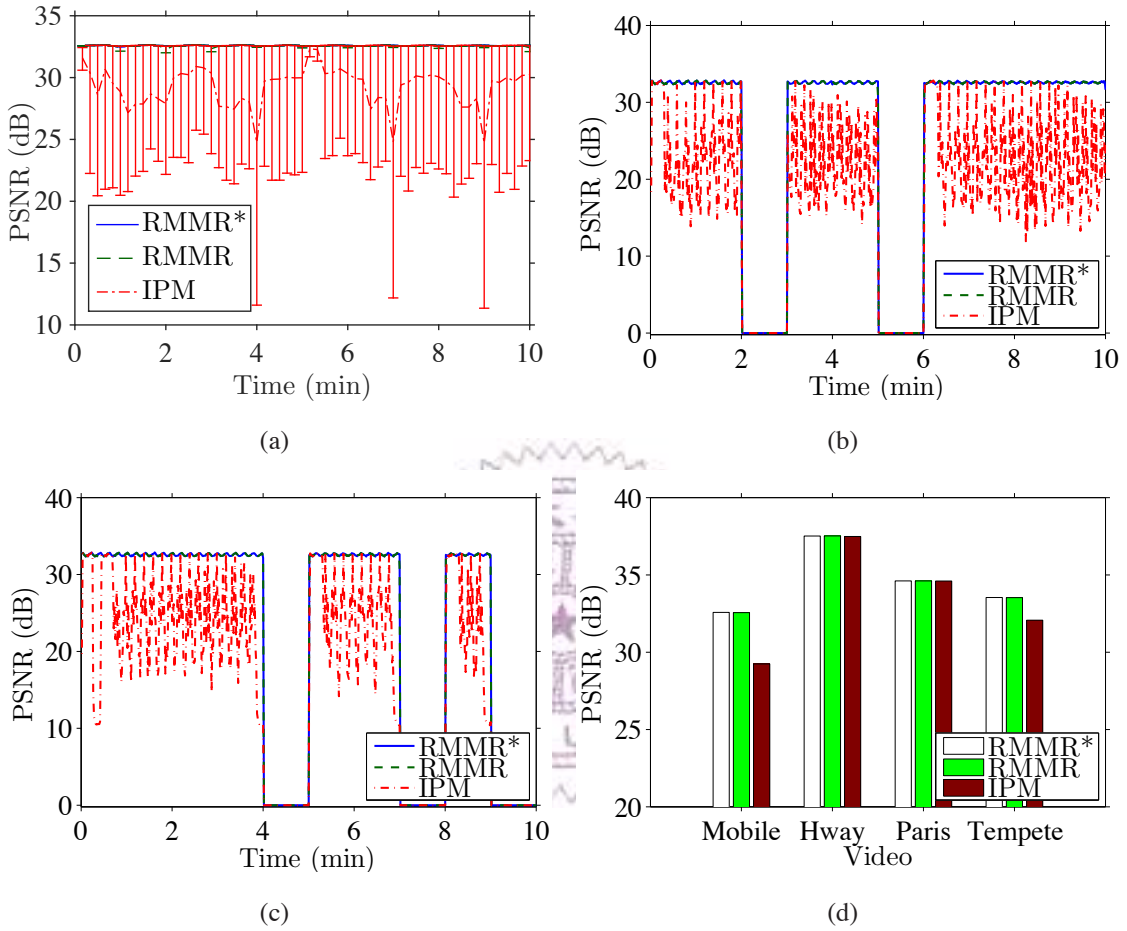
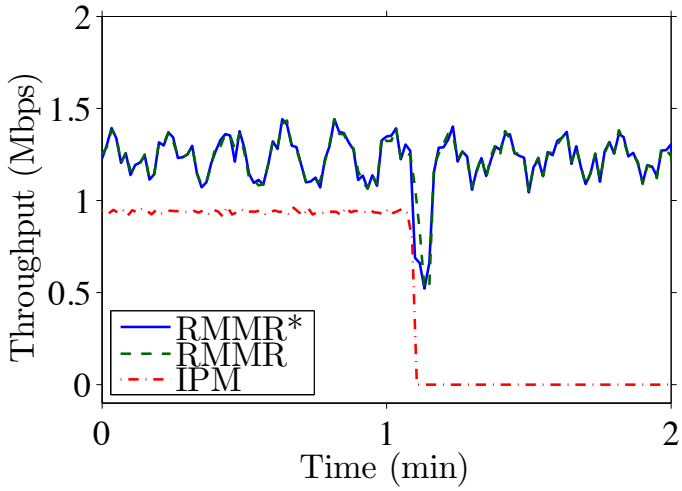
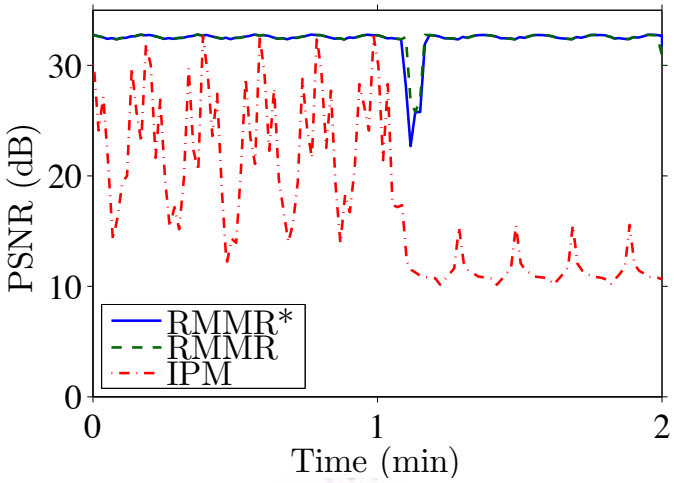


Figure 7.4: The proposed algorithms achieve higher video quality: (a) mean/min/max video quality, (b), (c) detailed video quality of sinks 1 and 5, and (d) average video quality from different videos.

Our proposed algorithms are robust against switch failures. The IPM multicasts the whole video over a single multicast tree, and any link and switch failures may cause video outage at some sinks. To validate this, we conduct a 2-min experiment using `Mobile` and small topology. We then instruct Mininet to tear down switch 1 at 65 sec. We calculate the throughput and video quality of individual sinks under RMMR*, RMMR, and IPM, and plot the results from sink 5 in Figs. 7.5(a) and 7.5(b). We observe that, with IPM, sink 5 does not receive any packets after 65 sec (Fig. 7.5(a)), and thus its video quality drops significantly (Fig. 7.5(b)). In contrast, our proposed RMMR* and RMMR algorithms only suffer from minor throughput and video quality drops after 65



(a)



(b)

Figure 7.5: Our proposed algorithms are robust against switch failures: (a) throughput and (b) video quality.

Video Quality in PSNR (dB)				
Algorithm	**Minute**	**Max**	**Mean**	**Min**
RMMR*	1	32.58	32.57	32.54
	2	32.00	32.00	31.99
RMMR	1	32.58	32.54	32.51
	2	32.28	32.26	32.25
IPM	1	32.58	26.00	20.77
	2	32.49	26.00	11.41

Table 7.1: Per-sink Video Quality Achieved by Different Algorithms while Link Down

sec, as our algorithms ensure all the sinks would receive at least one descriptor. Moreover, our algorithms quickly recompute the multicast trees, and the throughput and video quality bounce back a few seconds after the switch failure. Table 7.1 gives the per-min video quality, which clearly shows that the RMMR* and RMMR outperform the IPM by up to 6 dB in terms of mean PSNR.

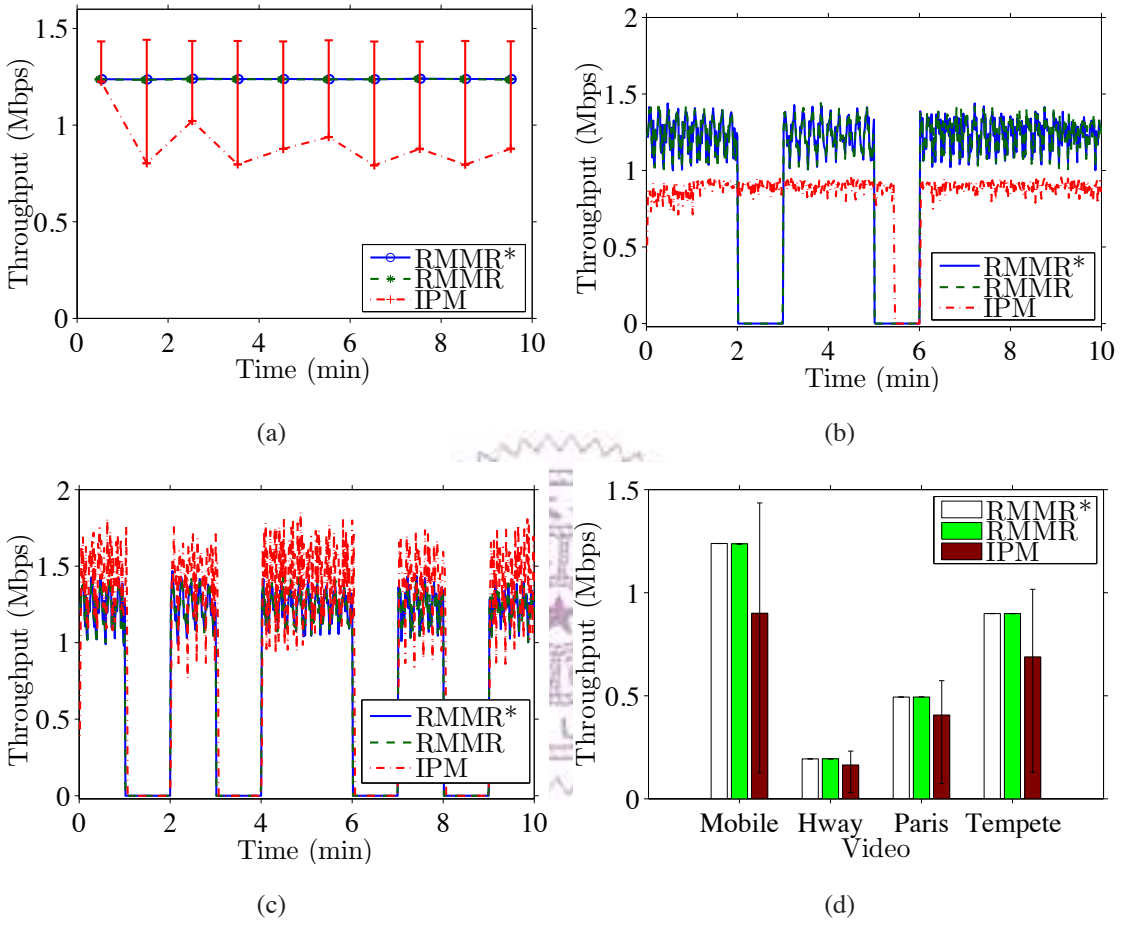


Figure 7.6: The IPM may result in lower/higher throughput at sinks: (a) mean/min/max throughput, (b), (c) detailed throughput of sinks 1 and 2, and (d) average throughput from different videos.

The IPM may incur higher/lower sink throughputs. We compute the per-sink throughput when multicasting each video over the small topology. We first plot the mean/min/max per-min throughput of all sinks receiving `Mobile` in Fig. 7.6(a). This figure shows that the proposed RMMR* and RMMR algorithms result in uniform throughput at all sinks. In contrast, the IPM results in huge variations on throughput among sinks. We then zoom into two sample sinks with lower throughput (sink 1) and higher throughput (sink 2) with the IPM, and plot their per-sec throughput in Figs. 7.6(b) and 7.6(c). Fig. 7.6(b) reveals that: (i) with the IPM, sink 1 misses many packets and suffers from lower throughput and (ii) the IPM takes some time to react to the departure of sink 1

between [5, 6) min and even longer between [2, 3) min. Fig. 7.6(c) shows that, with the IPM, sink 2 receives at about 1.4 Mbps on average, which is even higher than the mean video bit rate (1.27 Mbps). This is because sink 2 receives some duplicated packets under the IPM, contributing to the high link utilization. With the IPM, sink 1 (Fig. 7.6(b)) and sink 2 (Fig. 7.6(c)) suffer from degraded video quality and high network load. Last, we report the average per-sink mean/min/max throughputs in Fig. 7.6(d), which shows that the RMMR* and RMMR algorithms lead to the sink throughputs very close to the video bit rates, while the IPM incurs diverse throughputs (due to packet losses or duplicated packets).

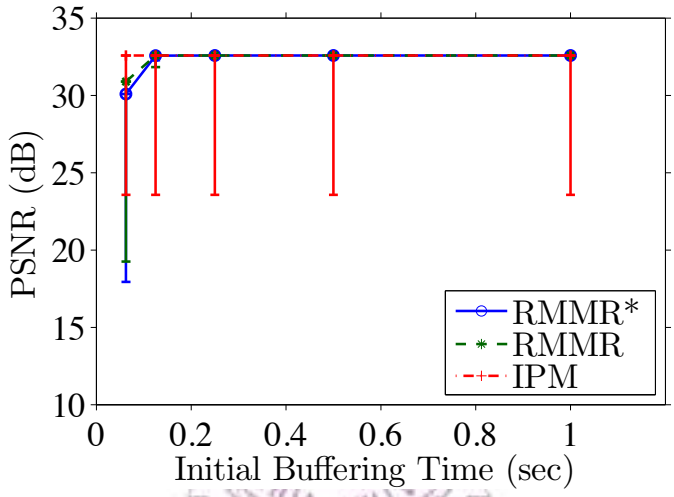
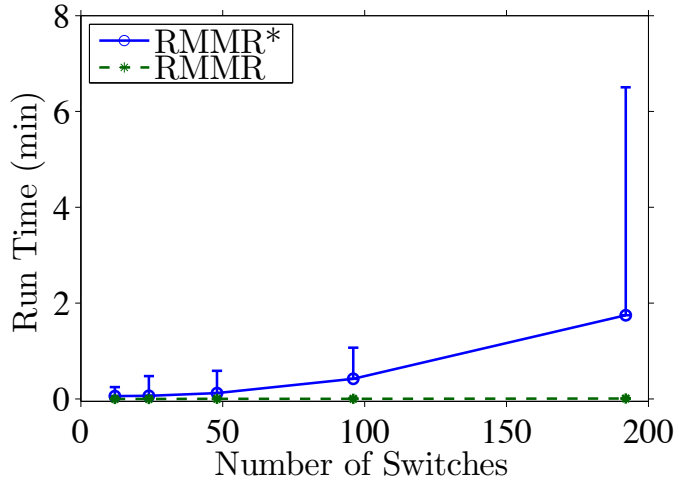


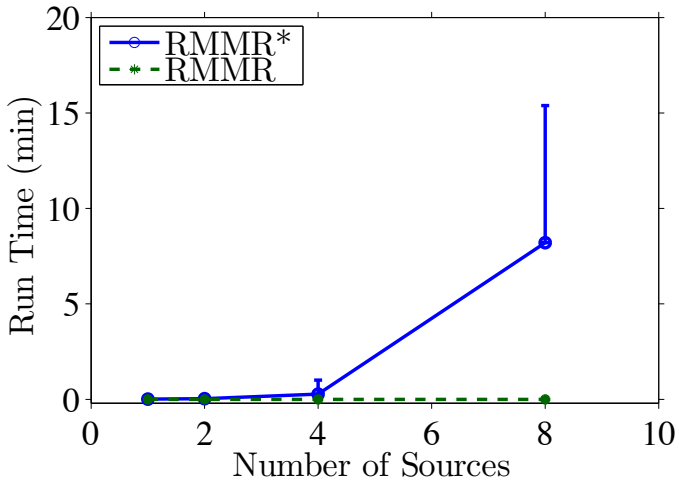
Figure 7.7: Implication of initial buffering time on video quality.

The proposed algorithms require short initial buffering time. The RMMR* and RMMR algorithms minimize the maximal link utilization and thus may lead to packet deliver time later than that of the IPM. To quantify this issue, we multicast `Mobile` multiple times with different initial buffering times and compute the average video quality of individual sinks in the small topology. We plot the mean/min/max sink video quality in Fig. 7.7. The figure shows that the RMMR* and RMMR algorithms only need very short initial buffering time to maintain the full video quality: (i) with a short buffering time ≥ 0.25 sec, all the sinks receive the video at the full quality and (ii) with a buffering time ≥ 0.125 sec, the worst case sink video quality of the RMMR* and RMMR algorithms are only 0.30 dB and 0.75 dB lower than the full quality. This figure shows that our proposed algorithms work with fairly short buffering time, yet achieve much higher video quality than that of the IPM.

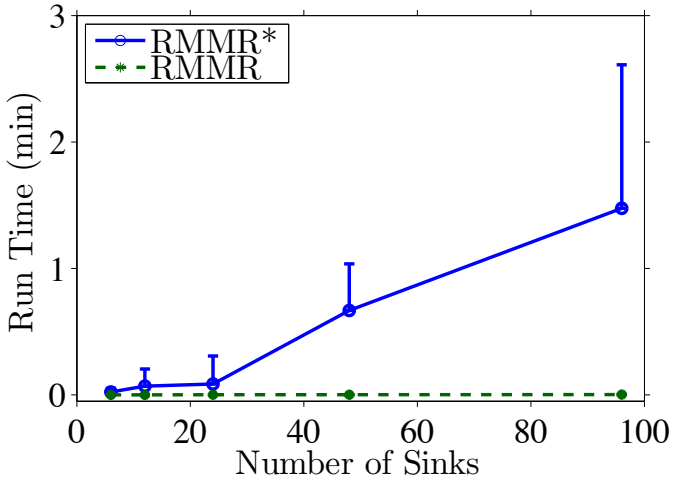
The scalability of our proposed RMMR algorithms. Next, we generate larger network topologies and measure the average execution time of the RMMR* and RMMR under different numbers of switches, video sources, and video sinks. We start from the



(a)



(b)



(c)

Figure 7.8: The scalability of the proposed algorithms: (a) various numbers of switches, (b) various numbers of video sources, and (c) various numbers of video sinks.

large network topology, and vary one of the parameters each time. For each configuration, we generate 10 random topologies, and measure the running time on an Intel i7 3.4 GHz Linux PC. We report the average/max running time in Fig. 7.8. The RMMR algorithm always terminates in < 200 ms, although this is not visible in the figure because of the y-axis scale. This figure shows that the RMMR* algorithm has 1+ min running time with 125+ switches, 5+ video sources, or 70+ video sinks. This figure demonstrates that the RMMR algorithm scales to larger networks, while the optimal RMMR* algorithm can be used with smaller network topology.

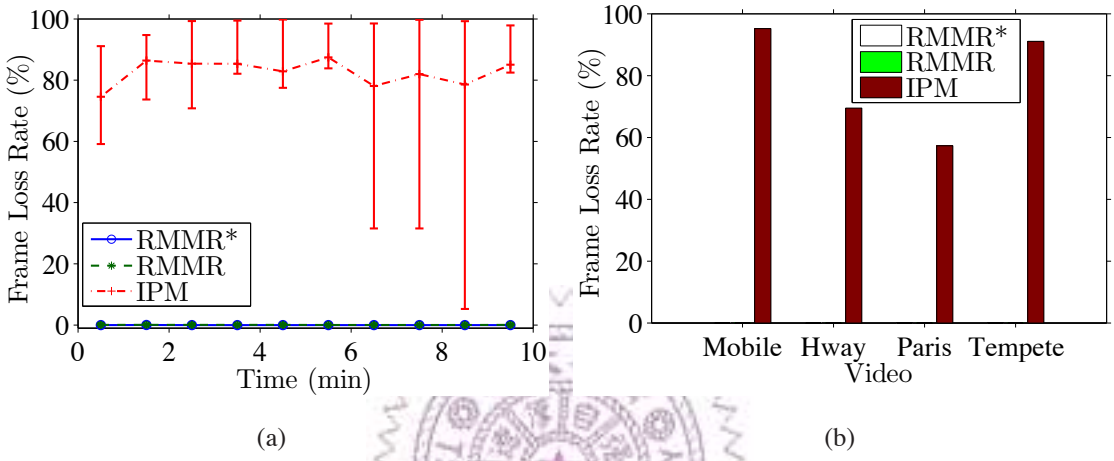


Figure 7.9: The frame loss rates in the large topology: (a) `Mobile` and (b) all videos.

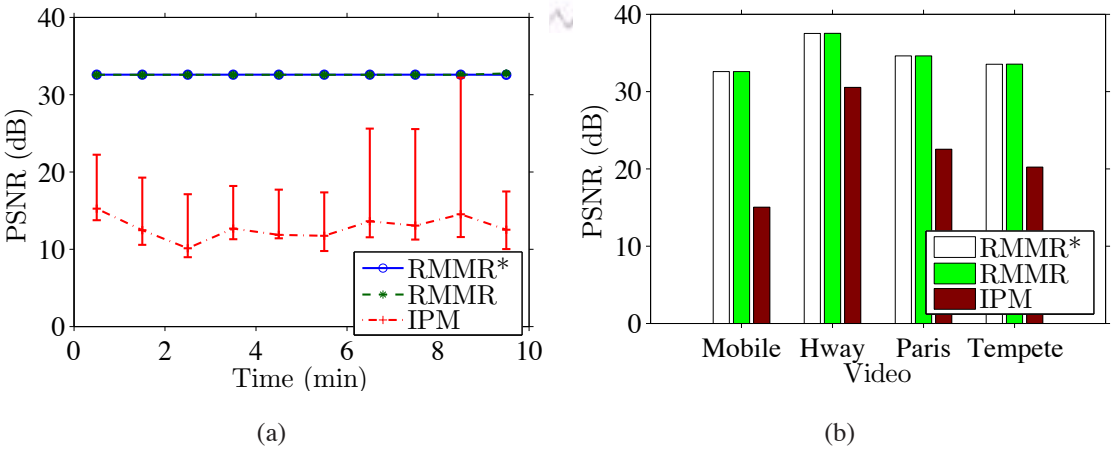


Figure 7.10: The video quality in the large topology: (a) `Mobile` and (b) all videos.

Our algorithms outperform the IPM in the large topology. Next, we conduct the experiments with the large topology, in which all four videos are concurrently streamed by four different video sources. Each sink randomly decides to join/leave the multicast group of each video once every minute, and on average there are about 10 video sinks watching every video at any moment. We compute the per-sink performance in terms of

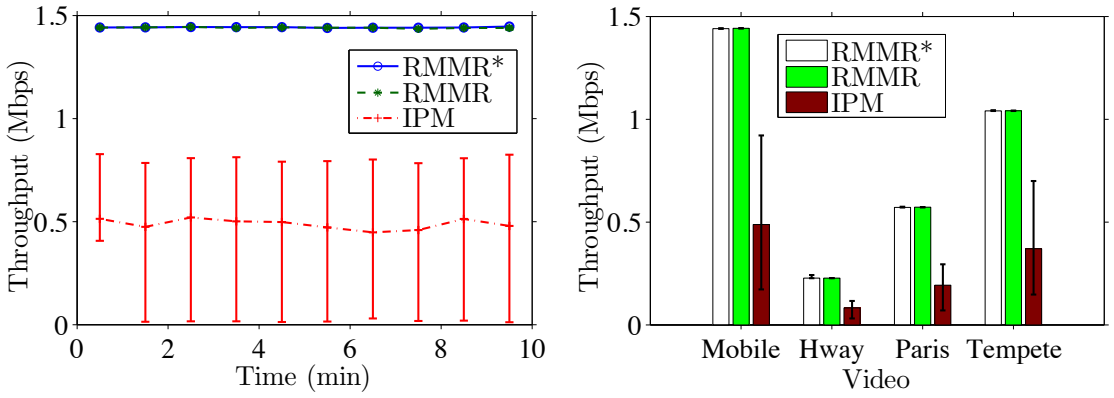
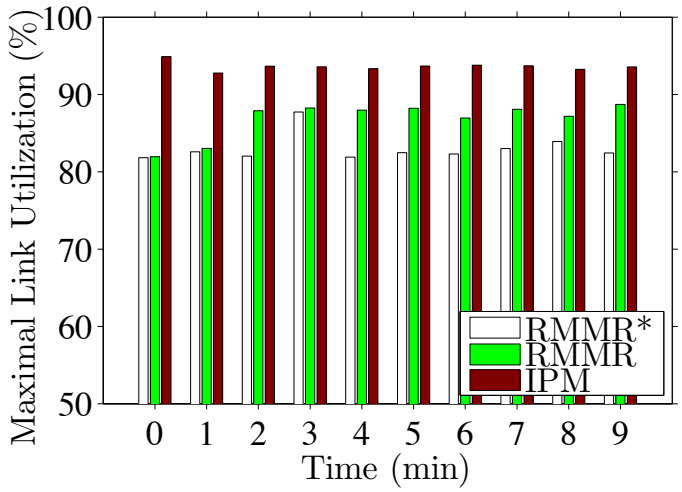


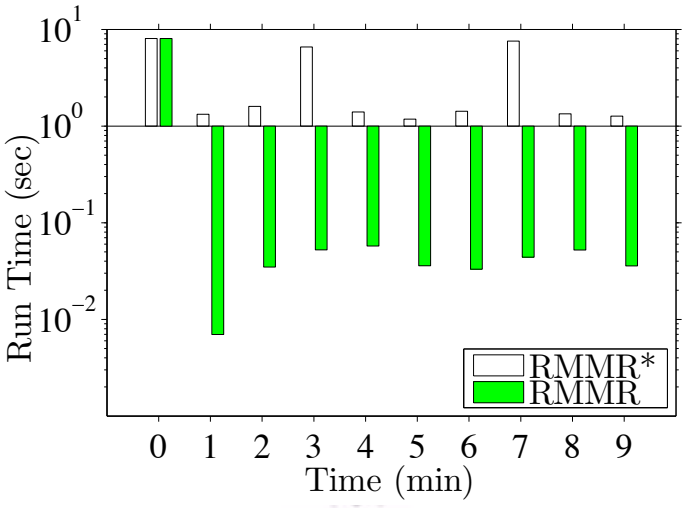
Figure 7.11: The throughputs in the large topology: (a) `Mobile` and (b) all videos.

frame loss rates, video quality in PSNR, and throughput. We then report results from sinks watching `Mobile` in Figs. 7.9(a), 7.10(a), and 7.11(a), and overall results in Figs. 7.9(b), 7.10(b), and 7.11(b). These figures clearly show that the IPM results in high frame loss rates (up to 95%), low video quality (up to 14 dB lower then RMMR*/RMMR), and low throughput (as low as 1/3 of the bit rate). Compared to the results from small topology, the gap between the RMMR*/RMMR and IPM is larger. This shows the potentials of deploying the RMMR* and RMMR algorithms in larger network topologies, as these topologies have more room for optimization.

The tradeoff between optimality and run time. The efficient RMMR algorithm results in close-to-optimal maximal link utilization in the small topology, compared to the RMMR* algorithm. The performance gap, however, may be bigger in the large topology. We plot the per-min maximal link utilization of multicasting the four videos in the large topology in Fig. 7.12(a). This figure shows that the RMMR* algorithm constantly maintains maximal link utilization at about 82%, while the RMMR algorithm leads to up to 88% maximal link utilization. We also plot the running time of the two algorithms in Fig. 7.12(b). This figure shows that the RMMR* algorithm may take up to 8 sec to complete, whereas the RMMR algorithm terminates in < 50 ms except in the first minute. It is because the RMMR algorithm invokes the RMMR* algorithm to compute the optimal multicast trees every $P = 10$ minutes, and the remaining invocations rely on the efficient RAH heuristics. Next, we plot the per-min maximal link utilization and running time of multicasting the four videos in the large topology with different P value in Fig. 7.13. Fig. 7.13(a) shows that RMMR* keeps overall maximal link utilization around 50%. The RMMR with larger P value prone to suffer from higher maximal link utilization. Fig. 7.13(b) shows the constant result with Fig. 7.12(b) that RMMR* takes up to 12 seconds to compute the routes, while the RMMR is able to terminate in only 0.01 sec-



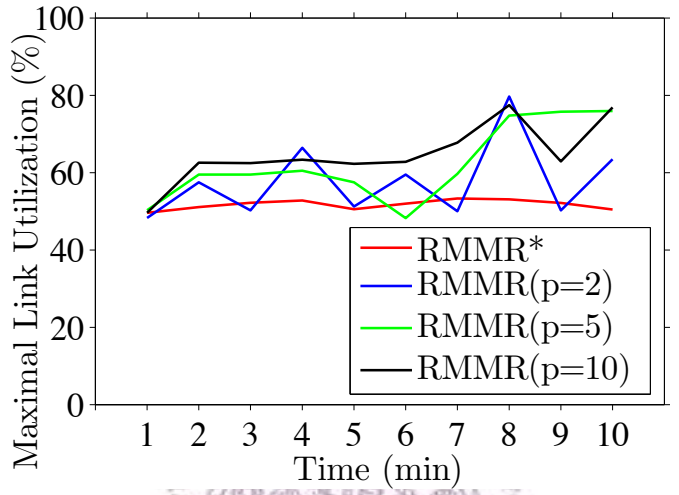
(a)



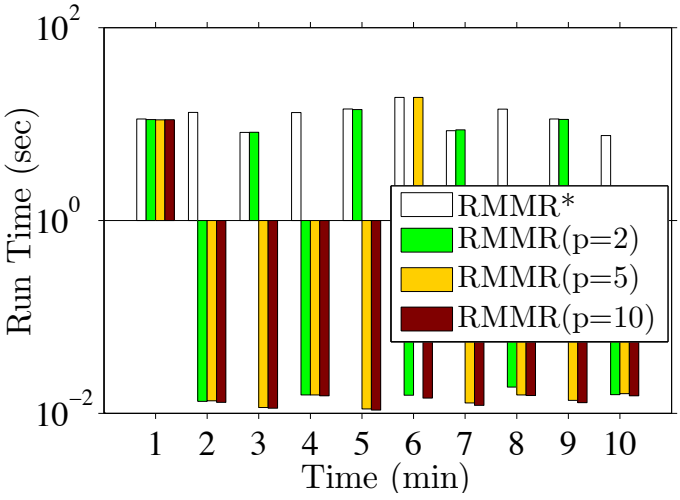
(b)

Figure 7.12: The tradeoff between RMMR* and RMMR: (a) optimality and (b) running time.

ond if the RAH is invoked. Fig. 7.12 demonstrate the tradeoff between the RMMR* and RMMR algorithms: the RMMR* algorithm is preferred when optimality is crucial, and the RMMR algorithm is preferred when the networks are large. We note that by changing the P value in the RMMR algorithm, we may better exercise the tradeoff in finer granularity. Designing an adaptive algorithm to pick the most suitable P value is one of our future tasks.



(a)



(b)

Figure 7.13: The tradeoff of different period parameter.

Chapter 8

Conclusion and Future Work

Supporting video streaming using IP multicast incurs high network infrastructure expense (CAPEX) and administrative expense (OPEX). In contrast, leveraging SDNs may not only reduce the expenses but also offer more room for optimization due to the global view of the networks. In this thesis, we studied the problem of multipath multicast routing for MDC videos, and strived to be robust against switch/link failures, load-balanced across all links, compatible to SDNs, and adaptive to topology changes. We proposed two algorithms, RMMR* and RMMR, to solve the problem. Our algorithms employ multipath multicast for robustness, mathematical optimization for load balancing and SDN compliance, and lightweight routing heuristics for adaptability. We have implement a system based on Ryu [31] and install our proposed two algorithms to support the multicast routing. We set up a real testbed and conduct the experiment using 4 Pica8 P-3297 OpenFlow switches and several PCs to run as video streaming server/client, which demonstrates the practicality of our algorithms. The experiment in testbed shows that (i) the video transmission latency raise extremely if the link is overload, (ii) it takes less than 5 milliseconds for system to insert the multicast flow on each switch, (iii) it takes no longer than 0.15 second for system to detect the video sinks change. We also used Mininet to emulate large network topologies for extensive experiments, and compared the performance of the proposed algorithms against the IP multicast. The experiment results show that, compared to the IP multicast, our algorithms: (i) lead to almost no frame loss and high video quality, (ii) minimize the maximal link utilization, (iii) avoid the duplicated packets, and (iv) are robust against switch failures. The performance gaps between our algorithms and the IP multicast are as high as: (i) 95% in frame loss rates, (ii) 15 dB in video quality, (iii) 2/3 in sink throughput, and (iv) 50% in maximal link utilization. Such clear performance gaps can be attributed to the fact that our algorithms are bandwidth-aware, mathematically rigorous, and mutlipath-enabled. Among the two proposed algorithms, the RMMR* algorithm results in optimal multicast trees (in terms of maximal link utilization), but may

take long to terminate; the RMMR algorithm runs fast at the expense of lower optimality. We recommend the RMMR* algorithm for smaller and more static networks, and the RMMR algorithm for larger and more dynamic networks. This work can be extended as the following: (i) Our multicast routing system does not consider the background traffic on the links. We may measure the background traffic from current link consuming and current multicast traffic on the link. The challenge is how to measure these values precisely to improve the performance. (ii) In RMMR, we decide to update the optimal routes by setting system parameter P. We may further design an adaptive algorithm to measure the difference between current solution and optimal one then determine whether update the routes by optimal solution.

Bibliography

[1] Tcpdump home page. http://www.tcpdump.org/.

[2] Video trace files and statistics. http://trace.eas.asu.edu.

[3] VideoLAN home page. http://www.videolan.org/.

[4] XORP home page. http://xorp.org.

[5] Cisco visual networking index: Forecast and methodology, 2014–2019. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf, 2015.

[6] T. Benson, A. Akella, and D. A. Maltz. Unraveling the complexity of network management. In *NSDI*, pages 335–348, 2009.

[7] Big Buck Bunny home page. https://peach.blender.org/.

[8] IBM ILOG CPLEX optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[9] A. De Gante, M. Aslan, and A. Matrawy. Smart wireless sensor network management based on software-defined networking. In *Communications (QBSC), 2014 27th Biennial Symposium on*, pages 71–75. IEEE, 2014.

[10] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[11] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the ip multicast service and architecture. *Network, IEEE*, 14(1):78–88, Jan 2000.

[12] W. C. Fenner. Internet group management protocol, version 2. 1997.

[13] Floodlight home page. http://www.projectfloodlight.org/floodlight/.

[14] N. Freris, C. Hsu, J. Singh, and X. Zhu. Distortion-aware scalable video streaming to multi-network clients. *IEEE/ACM Transactions on Networking*, 21(2):469–481, April 2013.

[15] P. Holub, H. Rudová, and M. Liška. Data transfer planning with tree placement for collaborative environments. *Constraints*, 16(3), July 2011.

[16] L.-H. Huang, H.-J. Hung, C.-C. Lin, and D.-N. Yang. Scalable and bandwidth-efficient multicast for software-defined networks. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 1890–1896. IEEE, 2014.

[17] A. Iyer, P. Kumar, and V. Mann. Avalanche: Data center multicast using software defined networking. In *COMSNETS*, pages 1–8, 2014.

[18] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[19] M.-W. Lee, Y.-S. Li, X. Huang, Y.-R. Chen, T.-F. Hou, and C.-H. Hsu. Robust multipath multicast routing algorithms for videos in software-defined networks. In *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of*, pages 218–227. IEEE, 2014.

[20] S. Liao, X. Hong, C. Wu, B. Wang, and M. Jiang. Prototype for customized multicast services in software defined networks. In *Software, Telecommunications and Computer Networks (SoftCOM), 2014 22nd International Conference on*, pages 315–320. IEEE, 2014.

[21] Y.-C. Liu, C. Chen, and S. Chakraborty. A software defined network architecture for geobroadcast in vanets. In *Communications (ICC), 2015 IEEE International Conference on*, pages 6559–6564. IEEE, 2015.

[22] J. Matias, B. Tornero, A. Mendiola, E. Jacob, and N. Toledo. Implementing layer 2 network virtualization using OpenFlow: Challenges and solutions. In *Proc. of European Workshop on Software Defined Networking (EWSDN'12)*, pages 30–35, Darmstadt, Germany, October 2012.

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74, Apr. 2008.

[24] Mininet home page. `http://mininet.org/`.

[25] Y. Nakagawa, K. Hyoudou, and T. Shimizu. A management method of IP multicast in overlay networks using OpenFlow. In *Proc. of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN'12)*, pages 91–96, Helsinki, Finland, August 2012.

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

[27] NOX/POX home page. `http://www.noxrepo.org`.

[28] M. Rahimi, A. Bais, and N. Sarshar. On fair and optimal multi-source IP-multicast. *Journal of Computer Networks*, 56(4):1503–1524, March 2012.

[29] J. Rückert, J. Blendin, R. Hark, T. Wächter, and D. Hausheer. An extended study of dynsdm: Software-defined multicast using multi-trees. Technical report, 2015.

[30] J. Rückert, J. Blendin, and D. Hausheer. Software-defined multicast for over-the-top and overlay-based live streaming in isp networks. *Journal of Network and Systems Management*, 23(2):280–308.

[31] Ryu SDN Framework Home Page. `http://osrg.github.io/ryu/`.

[32] S.-H. Shen, L.-H. Huang, D.-N. Yang, and W.-T. Chen. Reliable multicast routing for software-defined networks.

[33] B. Tamma, A. Badam, C. Murthy, and R. Rao. K-Tree: A multiple tree video multicast protocol for ad hoc wireless networks. *Journal of Computer Networks*, 54(11):1864–1884, August 201).

[34] P. Troubil and H. Rudová. Integer linear programming models for media streams planning. *Lecture Notes in Management Science*, 2011(3), August 2011.

[35] Y. Wang, J. Ostermann, and Y. Zhang. *Video Processing and Communications*. Prentice Hall, 2001.

[36] E. Yang, Y. Ran, S. Chen, and J. Yang. A multicast architecture of svc streaming over openflow networks. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 1323–1328. IEEE, 2014.

[37] M. Zhao, B. Jia, M. Wu, H. Yu, and Y. Xu. Software defined network-enabled multicast for multi-party video conferencing systems. In *Communications (ICC), 2014 IEEE International Conference on*, pages 1729–1735. IEEE, 2014.