# APPLICATION-AWARE NETWORK DESIGN USING SOFTWARE-DEFINED NETWORKING FOR APPLICATION PERFORMANCE OPTIMIZATION FOR BIG DATA AND VIDEO STREAMING

A Dissertation
IN
Telecommunications and Computer Networking
and
Computer Science

Presented to the Faculty of the University
of Missouri–Kansas City in partial fulfillment of
the requirements for the degree

DOCTOR OF PHILOSOPHY

by
SHUAI ZHAO

M.S., University of Missouri - Kansas City, USA, 2011
B.S., Heze University, Heze, ShanDong, China, 2008

Kansas City, Missouri
2017

APPLICATION-AWARE NETWORK DESIGN USING SOFTWARE-DEFINED NETWORKING FOR APPLICATION PERFORMANCE OPTIMIZATION FOR BIG DATA AND VIDEO STREAMING

Shuai Zhao, Candidate for the Doctor of Philosophy Degree

University of Missouri–Kansas City, 2017

## ABSTRACT

This dissertation investigates improvement in application performance. For applications, we consider two classes: Hadoop MapReduce and video streaming. The Hadoop MapReduce (M/R) framework has become the de facto standard for Big Data analytics. However, the lack of network-awareness of the default MapReduce resource manager in a traditional IP network can cause unbalanced job scheduling and network bottlenecks; such factors can eventually lead to an increase in the Hadoop MapReduce job completion time. Dynamic Video streaming over the HTTP (MPEG-DASH ) is becoming the de facto dominating transport for today's video applications. It has been implemented in today's major media carriers such as Youtube and Netflix. It enables new video applications to fully utilize the existing physical IP network infrastructure. For new 3D immersive medias such as Virtual Reality and 360-degree videos are drawing great attentions from both consumers and researchers in recent years. One of the biggest challenges in streaming such 3D media is the high bandwidth demands and video quality. A new Tile-based video is introduced in both video codec and streaming layer to reduce the transferred media size.

In this dissertation, we propose a Software-Defined Network (SDN) approach in an Application-Aware Network (AAN) platform. We first present an architecture for our approach and then show how this architecture can be applied to two aforementioned application areas. Our approach provides both underlying network functions and application-level forwarding logics for Hadoop MapReduce and video streaming. By incorporating a comprehensive view of the network, the SDN controller can optimize MapReduce workloads and DASH flows for videos by application-aware traffic reroute. We quantify the improvement for both Hadoop and MPEG-DASH in terms of job completion time and user's quality of experience (QoE), respectively. Based on our experiments, we observed that our AAN platform for Hadoop MapReduce job optimization offer a significant improvement compared to a static, traditional IP network environment by reducing job run time by 16% to 300% for various MapReduce benchmark jobs. As for MPEG-DASH based video streaming, we can increase user perceived video bitrate by 100%.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Graduate Studies, have examined a dissertation titled "Application-Aware Network Design Using Software-Defined Networking for Application performance Optimization for Big Data and Video Streaming," presented by Shuai Zhao, candidate for the Doctor of Philosophy degree, and certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Deep Medhi, Ph.D., Committee Chair
Department of Computer Science & Electrical Engineering

Appie Van De Liefvoort, Ph.D.
Department of Computer Science & Electrical Engineering

Baek-Young Choi, Ph.D.
Department of Computer Science & Electrical Engineering

Ken Mitchell, Ph.D.
Department of Computer Science & Electrical Engineering

Praveen R. Rao, Ph.D.
Department of Computer Science & Electrical Engineering

Zhu Li, Ph.D.
Department of Computer Science & Electrical Engineering

CONTENTS

ABSTRACT . . . . . . . . . . iii

ILLUSTRATIONS . . . . . . . . . . viii

TABLES . . . . . . . . . . xi

ACKNOWLEDGEMENTS . . . . . . . . . . xii

Chapter

1 INTRODUCTION . . . . . . . . . . 1

1.1 Application Background . . . . . . . . . . 1

1.2 Application-aware Networks and Software-defined Networking . . . . . . 4

1.3 Motivation and Scope . . . . . . . . . . 5

1.4 Contribution . . . . . . . . . . 7

1.5 Additional Research Contributions . . . . . . . . . . 9

1.6 Organization . . . . . . . . . . 21

2 RESEARCH SURVEY . . . . . . . . . . 22

2.1 Related Work . . . . . . . . . . 22

2.2 Research Scope . . . . . . . . . . 26

2.3 Background Overview . . . . . . . . . . 27

3 Proposed Architecture . . . . . . . . . . 32

3.1 General AAN-SDN Architecture and Implementation . . . . . . . . . . 32

4 AAN-SDN For Hadoop . . . . . . . . . . 39

4.1 HiBench: Bigdata Micro Benchmark Suite . . . . . . . . . . . . . . . . . . 39

4.2 Data Collection Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

4.3 AAN-SDN Hadoop Architecture and Implementation . . . . . . . . . . . 52

4.4 MapReduce Traffic Optimization Using SDN . . . . . . . . . . . . . . . 53

4.5 Performance Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . 54

5 Study of User QoE Improvement for Dynamic Adaptive Streaming over HTTP (MPEG-DASH) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

5.1 Composite Video Streaming QoE Metrics . . . . . . . . . . . . . . . . . 71

5.2 Proposed Dynamic Composite QoE Algorithm . . . . . . . . . . . . . . 74

5.3 Implementation and Empirical Evaluation . . . . . . . . . . . . . . . . . 77

6 AAN-SDN For MPEG-DASH . . . . . . . . . . . . . . . . . . . . . . . . . 84

6.1 Regular MPEG-DASH Video Streaming QoE Improvement . . . . . . . . 84

6.2 Immersive/360-degree MPEG-DASH Video Streaming QoE Improvement 90

7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103

Appendix

A Mininet Topology Source Code For DASH Setup . . . . . . . . . . . . . . . . 106

B Hadoop Installation Source Code . . . . . . . . . . . . . . . . . . . . . . . 109

C Openvswitch Installation Source Code . . . . . . . . . . . . . . . . . . . . 118

D HiBench Installation Guide . . . . . . . . . . . . . . . . . . . . . . . . . . 120

REFERENCE LIST . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122

VITA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135

ILLUSTRATIONS

Figure Page

1 Video Content Size Comparison . . . . . . . . . . . . . . . . . . . . . . . . 4

2 I-CaN-MaMa Framework . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3 Streaming Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

4 Storm Topology . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

5 BuDDI Middlebox Architecture. . . . . . . . . . . . . . . . . . . . . . . 13

6 SPArTaCuS: Architecture Framework . . . . . . . . . . . . . . . . . . . 14

7 Home Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

8 DASH Over WebRTC . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

9 WebRTC and WebSocket in NS3 Experiment Setup . . . . . . . . . . . . 18

10 DASH SAND Architecture . . . . . . . . . . . . . . . . . . . . . . . . . 20

11 Research Scope . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

12 Software-Defined Network Architecture . . . . . . . . . . . . . . . . . . 28

13 MapReduce WorkFlow . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

14 MPEG-DASH System Overview [51] . . . . . . . . . . . . . . . . . . . . 30

15 General AAN-SDN Architecture . . . . . . . . . . . . . . . . . . . . . . 32

16 Traffic Reroute Workflow Example . . . . . . . . . . . . . . . . . . . . 36

17 HiBench Hadoop MapReduce Related Workload Summary . . . . . . . . 40

18 HiBench MapReduce Workloads Configuration Details . . . . . . . . . . 41

19 Topology Type (a) TA, (b) TB-1, (c) TB-2, (d) TC, TD . . . . . . . . . . 43
20 CPU Load Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
21 Memory Load Summary . . . . . . . . . . . . . . . . . . . . . . . . . 49
22 Job Completion time Summary . . . . . . . . . . . . . . . . . . . . . 51
23 SDN Hadoop Experimental Architecture . . . . . . . . . . . . . . . . . 52
24 Experimental Network Setup . . . . . . . . . . . . . . . . . . . . . . 54
25 Loop Network Topology Setup . . . . . . . . . . . . . . . . . . . . . 55
26 Hadoop M/R runtime Comparison . . . . . . . . . . . . . . . . . . . 56
27 SDN Controller System Load . . . . . . . . . . . . . . . . . . . . . . 57
28 Control Traffic . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
29 (a) Network Topology, (b) Traffic Path Before Reroute, (c) Traffic Path After Reroute . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
30 Hadoop MapReduce Intermediate Data Transfer . . . . . . . . . . . . . 63
31 ARP Broadcasting Packet Cause CPU Utilization High in a Loop Topology 64
32 Hadoop Job Runtime Comparison . . . . . . . . . . . . . . . . . . . 67
33 Network Profile for Testing Environment [21] . . . . . . . . . . . . . . 77
34 A Sample Run of Network Traces Using Our Dynamic Algorithm . . . . 79
35 System Buffered Details: (a) Buffered Time and Queue Size (b) Buffered Bitrate Quality and Queue Number . . . . . . . . . . . . . . . . . . . . 80
36 BenchMark Sample Run . . . . . . . . . . . . . . . . . . . . . . . . . 81
37 Rebuffer Comparison for Various Network Profiles . . . . . . . . . . . 82

38 Comparison in Video Bitrate Switchover Frequence, Magnitude and Video Quality Between Proposed Dynamic and ExoPlayer Reference Algorithm 83
39 AAN-SDN DASH Experimental Architecture . . . . . . . . . . . . . . . . 85
40 GENI Topology Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
41 SDN Adaptive Path Selection For DASH Traffic . . . . . . . . . . . . . . 88
42 Multiple Clients Fetch Video Segment at The Same Time Using SDN and Non-SDN . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89
43 Tile-based Streaming Workflow [22] . . . . . . . . . . . . . . . . . . . . 91
44 DASH SRD XML Example . . . . . . . . . . . . . . . . . . . . . . . . 92
45 DASH SRD Coordinate Example ($3 \times 3$ tiles) . . . . . . . . . . . . . . . 93
46 Tile-based Video Playback Example . . . . . . . . . . . . . . . . . . . . 94
47 SDN DASH Experimental Architecture . . . . . . . . . . . . . . . . . . 95
48 Mininet SDN Topology Setup . . . . . . . . . . . . . . . . . . . . . . . 97
49 ROI Movement for A $2X2$ Tile-based Video Segments . . . . . . . . . . 98
50 Bitrate for Both Non-SDN and SDN deployment . . . . . . . . . . . . . 99
51 Buffered Size for Both Non-SDN and SDN deployment . . . . . . . . . . 100
52 QoE Improvement Comparison . . . . . . . . . . . . . . . . . . . . . . . 102

TABLES

| Tables | | Page |
|---|---|---|
| 1 | Message types | 16 |
| 2 | Monitored Port Number | 60 |
| 3 | HiBench Workloads Traffic Reroute Using SDN | 66 |
| 4 | Hibench Workload: Hardware Related Data Collection | 68 |
| 5 | Mapreduce Workload Traffic Flow Data Summary | 69 |
| 6 | Example of MapReducde Shuffle Traffic For Each Flow | 70 |
| 7 | Composite QoE Metrics | 72 |
| 8 | Test Video Bitrate Index | 78 |
| 9 | MPEG-DASH Video Dataset Characteristics | 86 |
| 10 | MPEG-DASH Video Dataset Characteristics | 97 |

# ACRONYMS

**AAN** Application-Aware Networks

**ABR** Adaptive Bitrate

**ARP** Address Resolution Protocol

**AVC** Advanced Video Codec

**CSMA** Carrier-Sense Multiple Access

**CMFD** Common Mode Failure and Common Dependencies

**DANE** Dash Aware Network Elements

**FoV** Fielf of View

**GENI** Global Environment for Network Innovations

**HDFS** Hadoop File System

**HEVC** High Efficiency Video Coding

**HMD** Head Mount Display

**ICN/CCN** Information-Centric Networking/Content-Centric Networking

**IoT** Internet of Things

**JVM** Java Virtual Machine

**M/R** MapReduce

**MPEG-DASH** Dynamic Adaptive Streaming Over HTTP

**MPD** Media Presentation Description

**OVX** OpenVirtex

**OTT** Over-the-Top

**OVS** Openvswitch

**PSNR** Peak Signal-to-noise Ratio

**QoE** Quality of Experience

**RoI** Region of Interest

**SAND** Server and Network Assisted DASH

**FDH** The Full-Duplex HTTP based Protocols

**SDN** Software-Defined Networking

**STP** Spanning Tree Protocols

**SRD** Spatial Relationship Descriptor

**SP** Sliding Percentile

**TE** Traffic Engineering

**VM** Virtual Machine

**VR** Virtual Reality

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Prof. Deep Medhi for the continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Appie Van De Liefvoort, Dr. Baek-Young Choi, Dr. Ken Mitchell, Dr. Praveen R. Rao, and Dr. Zhu Li, for not only their insightful comments and encouragement, but also for their hard questions, which incentives me to widen my research from various perspectives.

I thank my fellow labmates for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last four years.

I also want to thank all of my mentors during my Internship experience. Dr. Guo-Qiang(GQ) Wang and Dr. Ravishankar Ravindran from FutureWei Inc, gave me the first internship opportunity. They are great scientists and taught me much valuable knowledge. During my second internship at Raytheon BBN, Dr. Ali Sydney and Heidi Picher Dempsey were wonderful and generous friends and mentors. I am very grateful for everything they have taught me. Dr. Shan Liu was an awesome mentor when I was interning at Mediatek USA, Inc. She is very passionate and willing to mentor me with all of her efforts. My Ph.D. study experience could not be considered as complete without those mentors' support.

Special thanks to my family. Words cannot express how grateful I am to my mother-in law, father-in-law, my mother, and father for all of the sacrifices that you've made on my behalf. I would also like to thank all of my friends who supported me in writing and helped me to strive towards my goal. In the end, I would like to express appreciation to my beloved wife Jingxian Chen who spent sleepless nights with me and was always my support in the moments when there was no one to believe in me.

This work is partially supported by NSF Grant # 1526299.

# CHAPTER 1

# INTRODUCTION

The rapid development of cloud services, mobility, Internet of Things (IoT) sensors, and video streaming services not only led to an explosion of network data but is also challenging the existing network management and monitoring system. Users pay the premium and also expect quality on-demand access to those applications, infrastructure, and other IT resources. Handling today's mega datasets requires massive parallel processing that also puts a constant need for flexible capabilities from the underlying network. Network-based applications themselves are increasingly growing and require massively distributed computation.

## 1.1 Application Background

### 1.1.1 Hadoop MapReduce Big Data Processing Platform

To handle the ever-increasing data size, Hadoop [7] MapReduce (M/R) is a scalable framework that allows dedicated and a seemingly unbound number of servers to participate in the analytics process. The response time of an analytics request is a major factor for time to gain data insights. Hadoop has been designed as a shared computing and storage platform and supports parallel computing of jobs for multiple users. While the computing and disk I/O requirements can be scaled with the number of servers, scaling the system leads to increased network traffic in the underlying network. Arguably, the communication-heavy phase of M/R contributes significantly to the overall response time. This problem is further aggravated, if communication patterns are heavily skewed, which is common in many MR workloads. Most of this caveat of the MapReduce program is because the default Hadoop resource manager does not take the network condition into consideration for job scheduling.

### 1.1.2 MPEG-DASH Video Streaming Status

In the video streaming filed, the report from Cisco system [13] demonstrates that video streaming over the Internet has become a major network traffic class, which could reach four-fifth of overall traffic by 2020. Many video providers such as YouTube and NetFlix are facing an ever-increasing challenge to keep a high QoE for their subscribers. Dynamic adaptive streaming over HTTP (MPEG-DASH) [66] was initially designed for efficient streaming of 2D flat videos. It has become one of the de facto effective adaptive streaming approaches that can fully utilize the existing physical IP network infrastructure. However, the current IP network has the limitation of effective dynamic bandwidth allo cation, which can lead to suboptimal streaming experience for immersive content consumers. MPEG-DASH as one of the promising solutions, has caught significant attentions by both industrial and academic researchers. In DASH , the video files are encoded with different bitrates and divided into smaller sized segments. The clients can adapt the video bitrate by requesting the most suitable video segment by underlying network conditions. DASH video segments are stored under various bitrates with different playback lengths. By delivering appropriate bitrate segments, user QoE can be guaranteed. A high user-perceived QoE is a combination of many factors: available bandwidth, available video bitrates, and rebuffers. The key is to keep a balance among those factors.

The public interest of immersive devices such as head-mounted displays (HMD) for Virtual Reality (VR), 360-degree video cameras for capturing immersive contents and 3D playback support from the commercial website such as YouTube are drawing great attentions from both consumers and researchers. Compare with regular 2D flat video contents, the 360 VR videos are extremely bandwidth intensive especially with the 4K/8K video resolution being widely accepted as a functional minimum resolution for current HMDs, while 8K or higher is desired. Therefore, a major challenge is how to efficiently transmit these bulky 360 VR videos to bandwidth-constrained wireless VR HMDs at acceptable quality levels given their high bitrate requirements. Recommended VR/360-degree video display aspect ratio from YouTube is 2:1 for monoscopic or panoramic videos and 1:1 for stereoscopic videos with a ratio of 2:1 per eye for better user QoE. The file size is double w.r.t the original 2D videos. Nowadays, YouTube provides video contents with different resolutions. For illustration, we obtained video file sizes of three sample videos at different resolutions from YouTube, which is shown in Figure 1. This depicts how video content size increases exponentially while the video resolution increases.

A recently proposed tile-based approach is attempting to reduce the transfer size based on user's region of interests (ROIs). In a typical scenario for streaming VR/360 videos, the user usually views only a portion of the video at a time, called field-of-view (FoV). As a result, there is a huge waste of bandwidth for streaming content to the client that is not visible to the user. By knowing the user ROIs, we can stream it with high

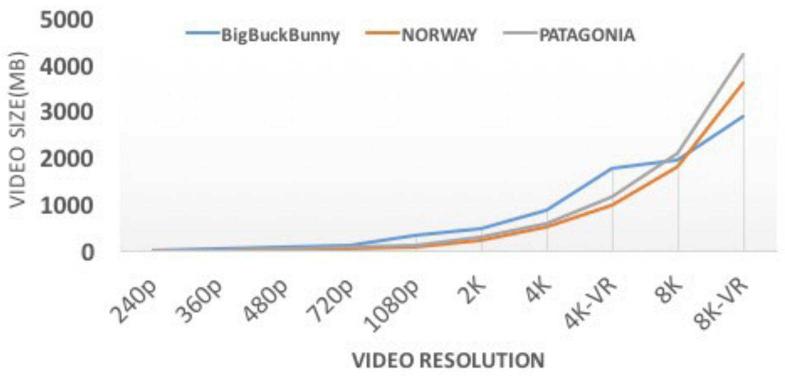


Figure 1: Video Content Size Comparison

quality while minimizing the quality of the rest of the video and thereby, saving the user bandwidth. In a video tiling scenario, the video is partitioned to multiple tiles and depending on the user's viewable area, we stream the overlapping tiles. Tiling has proven useful in domains such as online video lectures and sports.

### 1.2 Application-aware Networks and Software-defined Networking

The emergence of Application-aware Networks (AAN) provides a new approach for managing Hadoop network traffic.The AAN provides the capability of an intelligent network to maintain current information about applications that connect to it and as a result, optimize their functioning as well as that of other applications or systems that they control. The information maintained includes the application state and resource requirements. The main benefits of our AAN approach are 1) It allows a controllable Hadoop cluster management system and a fine-granular application control platform using the SDN architecture 2) It provides an open programming interface for more intelligent

Hadoop resource allocations with consideration of global network traffic information.

A software-based solution using software-defined networking (SDN) is a fine-grained way of controlling individual application and network devices. AAN benefits from SDN in two ways: first, by enabling dynamic control, configuration and giving the ability of AAN to allocate resources at any given moment; second, by running network controls on a separate server from the traffic forward device. The SDN is a relatively recent networking paradigm that has been conceived to address certain limiting of IP networking. With decoupled control and forwarding layers, SDN can dynamically optimize network flows traffic based on global network traffic information. It also has a finer quality of service (QoS) control based on assigned flow priorities. Our work aims to investigate how the future immersive video streaming scheme can exploit SDN for better QoE. Specifically, we propose a SDN-based approach to design a tile-based VR/360 streaming platform using DASH. In our work, we focus on 1) Hadoop computation using an AAN framework through SDN when Hadoop nodes are spread out over a network 2) Video Streaming using MPEG-DASH with proposed AAN-SDN.

## 1.3 Motivation and Scope

In general, the goal of our research is to improve applications' performance by proposed application-awareness networking framework. To prove its usability and efficiency, we tested our framework using two important applications in both big data and video streaming realm. We analyze the current issues and factors, which can affect the performance for those applications.

### 1.3.1 The Hadoop MapReduce Run Time Performance Optimization

A great deal of consideration must be put in place when managing a Hadoop cluster using resources for running MapReduce jobs in order to fully understand Hadoop traffic. The key elements are summarized as follows:

- *The block size and split size*: Hadoop uses blocks and split size to control how many blocks are being divided and used when running MapReduce jobs.
- *Block replication factor*: Hadoop uses this approach to prevent data loss because of common hardware failures.
- *The physical configuration of hardware resources*: This includes CPU, memory, hard disk capacity, interconnection network link speed and the number of slave nodes.
- *Java Virtual Machine(JVM)*: Hadoop uses JVM to complete jobs. The number of resources, mainly CPU and memory, can be assigned to each created JVM. Because the process of creation and killing of such resources takes time, the rule of thumbs when using JVM is that the less mapper is better, which leads to less JVM creation and less killing time. However, this must be accomplished by being given sufficient resources to start with for the submitted jobs.
- *Hadoop cluster topology*: How slaves deploy across the Hadoop cluster can be critical and depends on the assigned network bandwidth between master nodes and slave/data nodes.

- *Other Hadoop performance tuning methods*: Additional factors such as the number of files, file size, JVM Reuse and combiners are also important for Hadoop performance tuning.

### 1.3.2 User QoE improvement for MPEG-DASH Based Video Streaming

As video gets popular, user requires higher quality-of-experience (QoE). With limited network bandwidth and architecture, network side video traffic optimization can play an important role. The key elements are summarized as follows:

- *Video Streaming QoE Metrics*: Design QoE metrics for regular 2D video streaming is very important process to benchmark user perceived video qualities.
- *Network Profiles for Video Streaming*: Video quality changes based on networking condition such as bandwidth and delays.
- *Region of Interests (ROIs) Based Immersive/360-Degree Video Streaming*: Immersive video content streaming using various number of ROIs.

## 1.4 Contribution

In our research, we present AAN-SDN architecture for application performance optimization. We introduce tools and traffic reroute mechanisms for Hadoop and video streaming application optimization. We ran real-world MapReduce and MPEG-DASH streaming applications using our proposed AAN-SDN architecture to show improvement in both application use cases. Briefly, we make the following contributions:

1. w.r.t general AAN-SDN architecture design:
    - An application-aware framework with SDN
    - A new ARP (address resolution protocol) flooding avoidance resolver algorithm for our framework.
2. w.r.t Hadoop MapReduce optimization:
    - A data flow model to improve the data movement efficiency for MapReduce related workloads
    - An adaptive traffic engineering mechanism for the AAN-SDN environment for Hadoop applications
3. w.r.t MPEG-DASH streaming QoE improvement:
    - We present the difference in immersive content streaming between traditional- and SDN-based network.
    - We introduce a SDN-based framework to assist tile-based immersive content streaming.
    - We quantify the benefits brought by our SDN approach using network simulation. The results indicate that our scheme can increase of user's quality of experience.

## 1.5 Additional Research Contributions

During my Ph.D. study, I have conducted collaborative research that resulted in published papers in various journal and conferences in the area of networking, multimedia, and big data processing.

### 1.5.1 I-Can-MaMa: NetFlow Data Processing and Traffic Grouping

In I-Can-MaMa [80], the goal was to explore Cisco NetFlow Data from campus data center network and analyze traffic pattern and traffic grouping. In this paper, we focus on two important components that the CampusIS needs to address. The first is the campus physical network, and the second are the campus data centers that connect users to computing resources through the network. The goal of this project has been to take an integrated approach that gives the CampusIS system administrators (sysadmin, in short) the ability to understand the impact on the network. The impact may be due to different services using the data centers over the network, or due to users accessing external sites. The aim is to allow the CampusIS sysadmin to have a comprehensive view so that they can see how the network is being used by the data centers as well as external applications through a common framework. Because of virtual machines (VMs) provided at the data centers, it is also necessary to see how different traffic flows are incoming or outgoing from a specific VM. While the CampusIS sysadmin uses a number of commercial and public-domain tools, most of them are geared for a specific situation such as `mrtg`that gives views on a link use. We proposed the I-Can-MaMa Framework (Fig 2) for large NetFlow data processing. Our data integration environment focuses on creating a few key

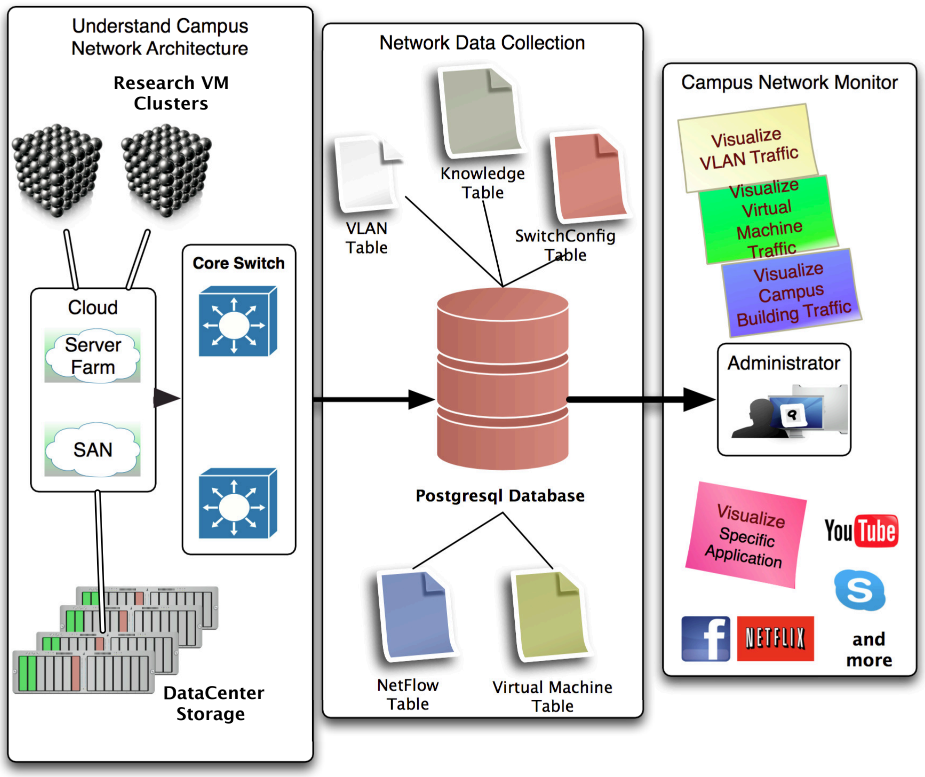


Figure 2: I-CaN-MaMa Framework

tables in the `PostgreSQL` database to store a few key pieces of information on a periodic basis. We have created a number of tables. Collectively, we have built a KnowledgeBase through these tables.

### 1.5.2 Real-time Network Anomaly Detection System Using Machine Learning

In [2], we presented the design and implementation of a real time network anomaly detection system for network-flow data in a campus network using Storm [69] real time data processing framework. The real network traffic of the campus network based using

Cisco NetFlow was collected at the campus data center at the University of Missouri–Kansas City (UMKC). Our proposed system is currently being developed using Apache Hadoop, Apache Storm, and Apache Kafka. Fig. 3 shows the architecture of our system. Apache's Hadoop distributed file system (HDFS) is an open source system for reliable, scalable, distributed computing. Hadoop is comprised of two major components, HDFS and MapReduce. For our real time streaming purpose, the HDFS component serves as a flow data producer. Our Storm topology can be mapped to Kafka topics.

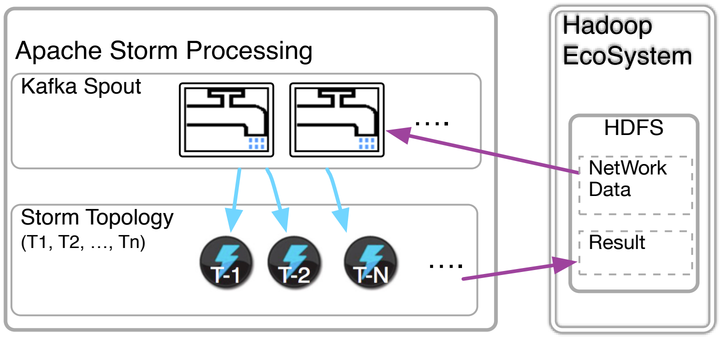


Figure 3: Streaming Architecture

Fig. 4 shows a topology setup for the anomaly detection in one of the switches. The streaming process is divided into four components: Kafka Spout, Data Preprocessing Bolt, Anomaly Detection Bolt, and Machine Learning Bolt. The Machine Learning Bolt conducted advanced anomaly detection over the anomaly traffic data detected by the Anomaly Detection Bolt. The data were used as training data for the machine learning process. The Machine Learning Bolt was implemented using a Weka Machine Learning tool [27] to improve the accuracy of anomaly detection tasks. The output was automatically saved and stored in HDFS for future use.

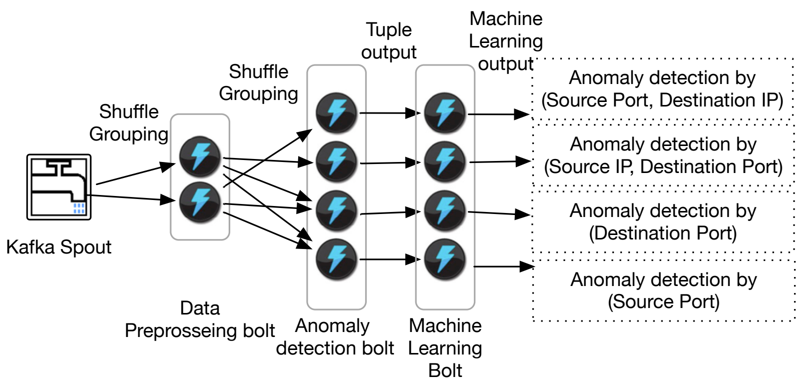

Machine Learning output
Tuple output
Shuffle Grouping
Shuffle Grouping
Anomaly detection by (Source Port, Destination IP)
Anomaly detection by (Source IP, Destination Port)
Anomaly detection by (Destination Port)
Anomaly detection by (Source Port)
Kafka Spout
Data Preprosseing bolt
Anomaly detection bolt
Machine Learning Bolt

Figure 4: Storm Topology

In this work, we presented a novel real-time system for network anomaly detection by utilizing state-of-the-art approaches including Apache Storm, Apache Kafka and applying real-time analytics on streaming data for network monitoring and management. We obtained promising preliminary results for real-time network anomaly detection.

### 1.5.3 BuDDI: Bug Detection, Debugging, and Isolation Middlebox for Software-Defined Network Controllers

In BuDDI [2], we used SDN concepts and proposed an online software Bug Detection, Debugging, and Isolation (BuDDI) middlebox architecture for SDN controllers. Software bugs cannot be resolved with them due to unexpected failure behavior. Furthermore, they are often bounded by common mode failure and common dependencies (CMFD). BuDDI consists of a shadow-controller based online debugging facility and a CMFD mitigation module in support of a seamless heterogeneous controller failover. For on-line bug detection and debugging, unlike a traditional $N + 1$ redundancy cluster system, we propose an $N + 2$ load balancing cluster system where components ($N$) have at least two independent failover components (+2). BuDDI enables a shadow controller that mirrors the active controller functions and turns on a verbose debugging mode for a specific failure module. Eventually, the two failover components will converge into one active controller. If the shadow-controller cannot identify a software bug in a given period, it sends a preemption message to the active CMFD module to take over the active role. Otherwise, it will confirm an active role for the CMFD module.

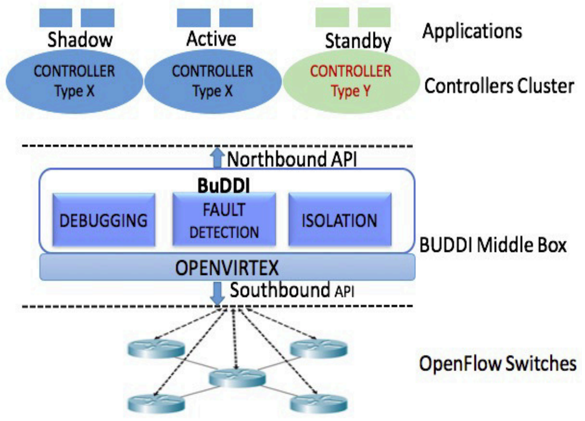


Figure 5: BuDDI Middlebox Architecture.

The proposed architecture is shown in Figure 5. The middlebox is connected to the controllers via northbound OpenFlow, whereas with the physical network, via southbound OpenFlow. Ryu, POX, FloodLight, Trema, and OpenDaylight are some of the most commonly used open source controllers. These controllers vary from each other in one way or another, which gives them diversity and supports our claim of using a heterogeneous controller approach. We verified that BuDDI supports our claim of a heterogeneous controller switchover without causing additional performance overhead.

### 1.5.4 SPArTaCuS: Service Priority Adaptiveness for Emergency Traffic in Smart Cities Using Software-defined Networking

In [1], we proposed SPArTaCuS (Service Priority Adaptiveness for Emergency Traffic in Smart Cities using Software-defined networking), a framework for smart cities on how to prioritize services for emergency needs in a stressed situation. Our approach is based on a promising new networking technology, known as software-defined networkings (SDN). Our approach resorts to virtualizing networks for different service classes, and dynamic allocation of resources as the need be.

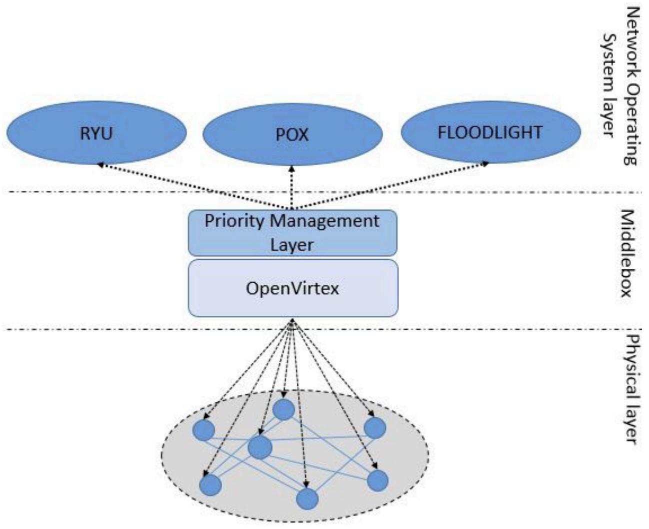


Figure 6: SPArTaCuS: Architecture Framework

Our proposed approach SPArTaCuS uses SDN to accomplish service prioritization for emergency services in a stressed situation. In particular, SPArTaCuS uses the SDN framework with OpenVirtex (OVX) to create virtual SDN networks for different service classes that are mapped to the physical infrastructure. Fig. 6 presents a high-level view of the SPArTaCuS framework. In our approach, the middlebox layer has a priority management layer on top of OVX; that is connected to multiple SDN controllers on the northbound interface. We argue to divide the traffic based on different organizations and prioritize them using the priority management layer in the middlebox. We illustrate two situations where SPArTaCuS can be helpful.

### 1.5.5 SeSAMe: Software Defined Smart Home Alert Management System for Smart Communities

In SeSAMe [3], we proposed an architectural vision for software defined smart community home alarm management based on software defined networks. We presented the protocol messages and systems components for the operation of SeSAMe. With our approach, should any alert/event such as a fire occur, an automated notification is sent to all the homes in the neighborhood and to the fire department and the police department about the fire. At the same time, alerts can also be forwarded to the police and the fire departments.

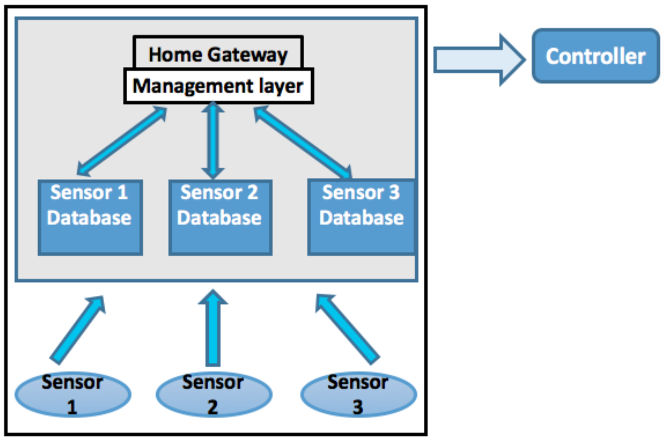


Figure 7: Home Architecture

Fig. 7 shows the high level architecture of a smart home in SeSAMe. It can be categorized into two categories: home gateway and sensors. The sensors include different sensors that are part of the home, e.g., fire sensor, temperature sensor, light sensor, motion sensor, and so on. All sensors send their data to the home gateway. The controller creates a database of the readings from various sensors. As shown in the figure, there are three sensors in the home and the controller creates a database for each of the sensors. The home gateway consists of a database where the reading from different sensors is stored, at least temporarily.

Table 1: Message types

| **Message Type** | **Sender-receiver** | **Description** |
|---|---|---|
| Update | Home-to-Controller | Sends updates to the controller |
| Trigger | Home-to-Controller | Event triggered message to notify the controller of any alert |
| Announce | Home-to-Homes | Notify the homes of any alert |
| Keepalive | Home-to-Controller | Notify the controller that the home is connected |

In SeSAMe, we define four different types of messages as shown in Table 1. By varying the message size and frequency, we measure the efficiency of proposed system. The preliminary results show that using SDN approach alerts can be communicated very quickly. Moreover, they give the flexibility of programmable control functions, lower operating costs, and centralized management, to name a few.

### 1.5.6 Low Delay Video Streaming Using WebRTC for MPEG-DASH in Both Wired and Wireless Networks

In [81, 83], we studied MPEG-DASH streaming over the WebRTC data channel. We explored the low delay WebRTC data channel as a transport vehicle for carrying DASH video sessions, and deploy our own sender side pacing solution for low delay DASH streaming. Simulation over a relay network using the NS-3 platform, demonstrates the significant improvement in end-to-end QoE delivery and reduction of start up and channel switching delays.

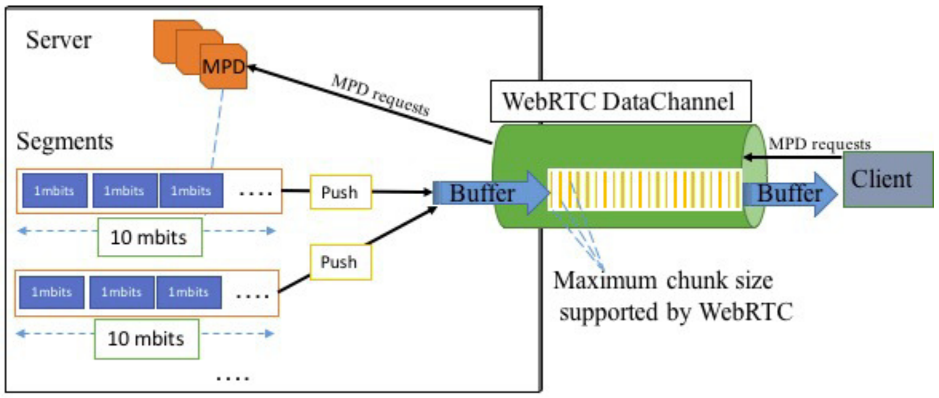


Figure 8: DASH Over WebRTC

With WebRTC based solution, which is RTP based, we are not suffering from TCP under-utilization. However, this comes with a cost because the sender needs to take over the congestion avoidance logic and pacing the transmission. We use a sender self-timing transmission scheme, with underlying webRTC congestion signals exposed via WebRTC APIs to help avoid congestion. Fig. 8 shows our system architecture. A server serves DASH MPD files and segment data. A DASH client communicates with the server through a WebRTC*datachannel*.

In this work, we conduct our experiments on both DASH and WebRTC using an NS-3 simulated network. In order to run the simulated network with an existing DASH .js player and WebRTC applications, we deploy two Linux containers using a Linux LXC library and Chrome as our test web browser. The DASH.js player and WebRTC are deployed as user level applications running on top of two different Linux containers that are connected by tap devices using an NS-3 simulated CSMA network. Figure 9 shows the topology setup using NS-3.

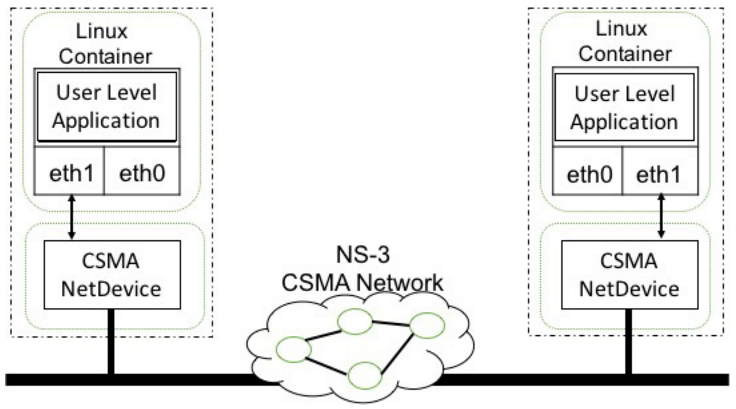


Figure 9: WebRTC and WebSocket in NS3 Experiment Setup

Initial simulation demonstrated very high link utilization and a low delay for DASH over WebRTC as compared with a typical DASH .js client over the HTTP link. In the future, we will further invest and expose the underlying WebRTC congestion model to support better QoS adaptation and introduce new sptaio-temporal QoE metrics to MPD to facilitate better end-to-end QoE optimization. Also, it will be interesting to introduce a thin middleware layer plug-in that enables WebRTC based peer-to-peer DASH streaming.

### 1.5.7 Wireless Video Traffic Bottleneck Coordination With A DASH SAND Framework

In [45], we introduced new Quality of Experience (QoE) metrics for DASH sub-representations, and proposed a marginal utility maximization-based resource pricing scheme to coordinate multiple video traffic sharing the bottleneck resource. The solution is based on the DASH SAND (Server And Network assisted DASH ) messaging framework. Simulation results demonstrated the QoE multiplexing gains from this solution, and the pricing control scheme is adopted in SAND messages. The growth of video data traffic is far out pacing the network capacity growth, and it dictated the reality that a large portion of mobile video sessions will be operating at a quality of service (QoS) deficit over a bottleneck. The bottleneck could happen over both the eNodeB wireless channels, over the home broadband gateway, and the links in the mobile core networks. The original DASH framework offers an effective rate adaptation scheme within a single streaming session, but lacks the coordination mechanisms among users sharing a common bottleneck. In recent work, DASH SAND (MPEG-DASH Part 5) is addressing this problem by introducing new messaging and control schemes among DASH servers, clients and middle boxes (known as DANE: Dash Aware Network Elements). The overall SAND architecture is illustrated in Fig. 10 [61].

We also introduced a new temporal quality metric and DASH sub-representation packing solution, to offer finer granular streaming operating points in rates and graceful spatio-temporal quality degradation when streaming rates need to be reduced. For a

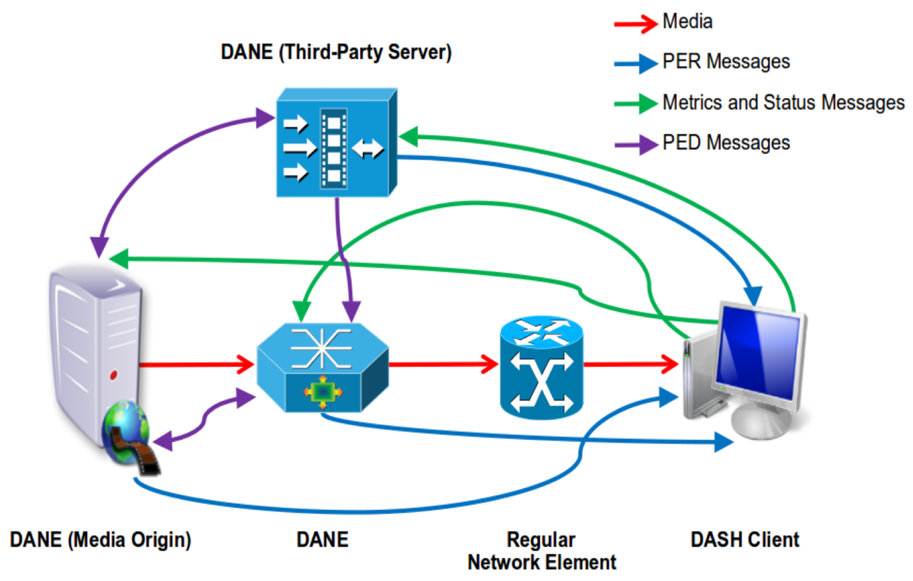


Figure 10: DASH SAND Architecture

pre-coded DASH segment at a certain PSNR quality level, we can further derive temporal quality operating points by packing the I, P, and B frames according to their if-loss distortion-induced characteristics. This is achieved by a frame significance function based frame loss distortion metric [44].

### 1.5.8 Multi-party Conference Over Virtual Service Edge Router (VSER) Platform

In [8], we investigated Virtual Service Edge Router (VSER) platform a realized using OpenStack, which is an ICN edge service router with the capability of hosting arbitrary realtime and non-realtime services as virtual machines (VM). The platform services are orchestrated through a programmable framework and takes advantage of scalable forwarding plane for content distribution.

## 1.6 Organization

The rest of this dissertation is organized as follows: Chapter 2 discuss our research scope and background knowledge on SDN, Hadoop MapReduce, and MPEG-DASH . We present our proposed the general AAN-SDN architecture for application performance optimization in chapter 3. We discuss how AAN can benefit Hadoop MapReduce jobs in chapter 4. We show our study of user QoE improvement for dynamic adaptive streaming over HTTP (MPEG-DASH) in chapter 5. We then discuss how AAN can benefit video streaming using MPEG-DASH in chapter 6. We conclude our work in chapter 7.

# CHAPTER 2

# RESEARCH SURVEY

In this chapter, we will discuss the state-of-art researches that have been done with regarding to improve the performance for both Hadoop MapReduce and MPEG-DASH streaming. We then discuss our research scope and related background review for Hadoop MapReduce, MPEG-DASH, and Software-defined networking.

## 2.1 Related Work

### 2.1.1 Hadoop MapReduce Related work

The Hadoop MapReduce [26, 41, 72] as a distributed processing framework has become the dominant approach for processing volumetric traffic in the big data era. Many researchers have studied several options to improve MapReduce's performance. Recent work, using traditional IP networks, can be grouped into two categories within given hardware resources: (1) an advanced Hadoop resource scheduling algorithm design in [24, 25, 34, 68, 71, 73, 76, 78, 79] and (2) job optimization with optimized configuration parameters using specific hardware in [5, 43, 74].

The Existing Hadoop MapReduce resource scheduling algorithm manages to optimize the Hadoop cluster resources such as slave nodes, CPUs, memories, networks, and disks. Those algorithms fall mainly into three categories: FIFO, capacity-based, and fairness schedulers. There are also heuristic designs that focus on data locality with simulations. BAR [34] proposed a heuristic task scheduling based on data locality by simulation, by initially finding an ideal data location for job processing in order to reduce the job running time. However, the assumption of initial job starting and completion time cannot stand in a real network. The proposed wait and random peeking approach in [68] and a fair scheduler [78] improves data locality. However, those methods can be further improved if integrated with network information and by using real-time data traces. SHadoop [23] takes an approach of modifying standard Hadoop's map and reducing execution methods to avoid employing any particular hardware or supporting software. Other similar works, which use adaptive job performance scheduling under various cluster circumstances are in [25, 48].

Another aspect of improving MapReduce job completion time can be achieved using a hardware acceleration approach [6, 74]. Special software and hardware need to be deployed that may not be readily accessible for normal cluster setups. Job specific optimization for MapReduce works is presented in [43, 62]. However, it lacked generalized methods for the overall performance of MapReduce jobs.

By using the Hadoop cluster under the traditional IP network, MapReduce's performance can be substantially degraded due to (1) the inherent characteristic of intensive data shuffle frameworks to transfer a large amount of intermediate data among slave nodes, and (2) default resources' allocation methods that lack the global view of real-time network traffic information. TCP related optimization work for MapReduce workloads is invested in [9, 15, 77], but the overall operation still bears low-performance improvement.

New network frameworks have been studied to identify new approaches to achieve a better MapReduce performance, such as in MROrchestrator [63], Coflow [11] and Orchestra [12] with much more sophisticated application integration designs. Pythia [52] has similarities with our approach but lacks a clear and comprehensive SDN system design with respect to MapReduce and a related application-aware approach. A preliminary idea on our approach was presented in [87]. Our application-Aware network design on top of SDN provides a common API interface, which can provide a full range of capabilities for network management and monitoring for different applications. We also measure the SDN control latency cost in our test cases. Based on the test results, our AAN realization can be better utilized for applications such as Hadoop M/R, which does not rely on a low lantency requirement. Our recent work published [85] shows the benefits how such an AAN-SDN can improve the Hadoop's run time efficiency.

### 2.1.2 MPEG-DASH MapReduce Related work

With modern capturing systems for 4K or Ultra High Definition (UHD) and 8K or ultra high definition television (UHDTV) video, new types of media experiences are possible where end users have the possibility to choose their viewing direction. Also with increased interest of immersive HMD devices for VR/360 content playback, user's viewing experience suffers because of the present limitations in both of existing network framework and video delivery methods.

A great deal of work has been done on both video encoder and delivery methods to reduce transferred media size. Tiling, in the video codec level such as the H.265/HEVC [**?**, 14] refers to a spatial partitioning of a video where tiles correspond to independently decodable video streams, which takes a divide-conquer approach to encoding and can reduce video content size by half. However, it is currently not widely adopted compared with the H.264/AVC encoding method.

With a 2D flat video, a tiled video can be obtained from a single video by partitioning each frame into multiple frames of smaller resolution and by aggregating the smaller frames coming from the same partition/region of the input frame into a new video. Here, tiles are defined as a spatial segmentation of the video content into a regular grid of individual videos. Similar ideas are also being used for creating tile-based VR/360 contents [28, 29, 42, 47, 59, 67].

MPEG-DASH is one of de facto effective adaptive streaming approaches that can fully utilize the existing physical IP network infrastructure. To utilize the DASH approach for video streaming, various client sides' ABR algorithm development have been proposed. Some of the representatives of the bandwidth-based approach such as PANDA [46], Elastic [16], Festive [33], SARA [35]. The performance of this approach can be affected by its bandwidth estimator's accuracy. Bandwidth estimation and prediction are known to be tough tasks [60, 75]. The fundamental design of DASH architectures makes the DASH client play important roles for a smooth and quality playback by implementing the mechanism for selecting the bitrate and resolution is built into the player. Buffer-based algorithms (BBA) are also studied such as in BOLA [65]. Such BBA-based approach is adopted by NetFlix. We also studied the low delay MPEG DASH streaming over the WebRTC data channel related in [82, 83] and wireless streaming in [45].

DASH also supports the tiling scheme in the Media Presentation Description (MPD) file [54]. By specifying spatial relationship description (SRD) in the DASH MPD file, the client can fetch video segments based on current ROIs such as applications in [10,18,38].New streaming architectures is proposed in [58] to provide an efficient streaming experience. However, such experiments have been conducted using the existing IP network architecture. Related research such as improving the quality of experience for streaming tile-based videos are also being investigated in [50,59]. We studied the DASH QoE in [84] using the traditional IP network as our benchmark.

Our research is to exploit how SDN can be used in Hadoop and DASH including regular 2D and immersive streaming scenarios. One recent work in [86] shows our AAN-SDN architecture design for MPEG-DASH based streaming and Tile-based immersive streaming.

## 2.2 Research Scope

In our research, we focus on understanding and exploring the issues regarding the current network and application. Fig. 11 shows our research scope. We started with Hadoop as one experimental application to build an AAN-SDN based network architecture. We investigate the bottleneck point under traditional network architecture. We then propose our AAN-SDN architecture. We explain our architecture in details. With intensive real test cases, we show the performance gain under proposed architecture. We then extend our methodology to improve the video streaming performance using MPEG-DASH.

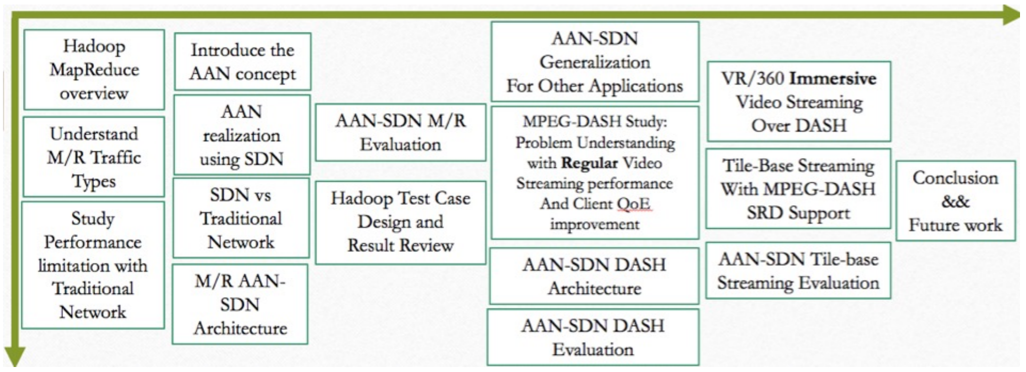

Hadoop MapReduce overview
Understand M/R Traffic Types
Study Performance limitation with Traditional Network
Introduce the AAN concept
AAN realization using SDN
SDN vs Traditional Network
M/R AAN-SDN Architecture
AAN-SDN M/R Evaluation
Hadoop Test Case Design and Result Review
AAN-SDN Generalization For Other Applications
MPEG-DASH Study: Problem Understanding with Regular Video Streaming performance And Client QoE improvement
AAN-SDN DASH Architecture
AAN-SDN DASH Evaluation
VR/360 Immersive Video Streaming Over DASH
Tile-Base Streaming With MPEG-DASH SRD Support
AAN-SDN Tile-base Streaming Evaluation
Conclusion && Future work

Figure 11: Research Scope

## 2.3 Background Overview

We first present a brief background on three important parts for our work: software-defined networking, MapReduce framework, HiBench benchmark suite and MPEG-DASH.

### 2.3.1 Software-defined Networking

Software-Defined Networking (SDN) [39, 57] provides a dynamic, manageable and cost-effective platform for making it an important platform for the high-bandwidth, dynamic nature of today's network applications. Fig 3 shows the SDN architecture. It decouples the control and data forwarding layers and provides the programming interface for the underlying forwarding devices as well as upper application layer. The SouthBound and NorthBound APIs are provided as communication channels between the SDN layers. AAN can be realized using an SDN architecture that includes two main components: an SDN controller and a traffic forwarding protocol using the forwarding devices. An SDN controller is a software application that manages application flows to enable a dynamic and controllable networking environment. The popular SouthBound communication protocols between SDN controllers and forwarding devices are OpenFlow [39,49,57], which allows servers to instruct forwarding devices where to send packets.

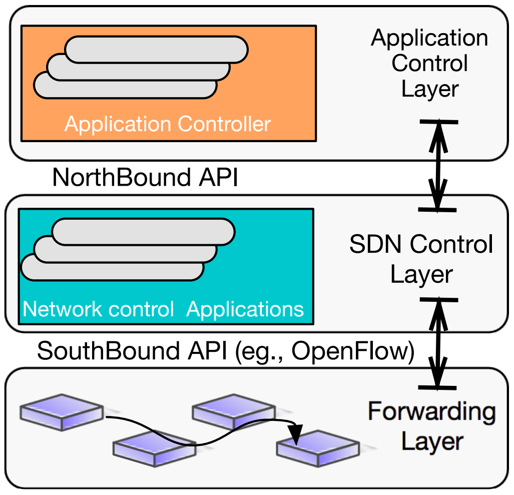

Application Control Layer
Application Controller
NorthBound API
SDN Control Layer
Network control Applications
SouthBound API (eg., OpenFlow)
Forwarding Layer

Figure 12: Software-Defined Network Architecture

### 2.3.2 MapReduce Framework

A Hadoop cluster includes one master node to manage the cluster's metadata, such as the Hadoop File System (HDFS) and a resource manager such as YARN [70] that manages Job and task tracker for each submitted MapReduce job. Designated slave nodes run as computing powers. A Hadoop cluster is normally deployed in a closed and control environment such as in an Enterprise or a Campus datacenter (DC). Hadoop MapReduce [72] is a distributed and parallel computing framework that runs on top of the Hadoop File System (HDFS). A typical MapReduce program is composed of mixed operations among various numbers of mapper and reducer functions. Fig. 13 shows the

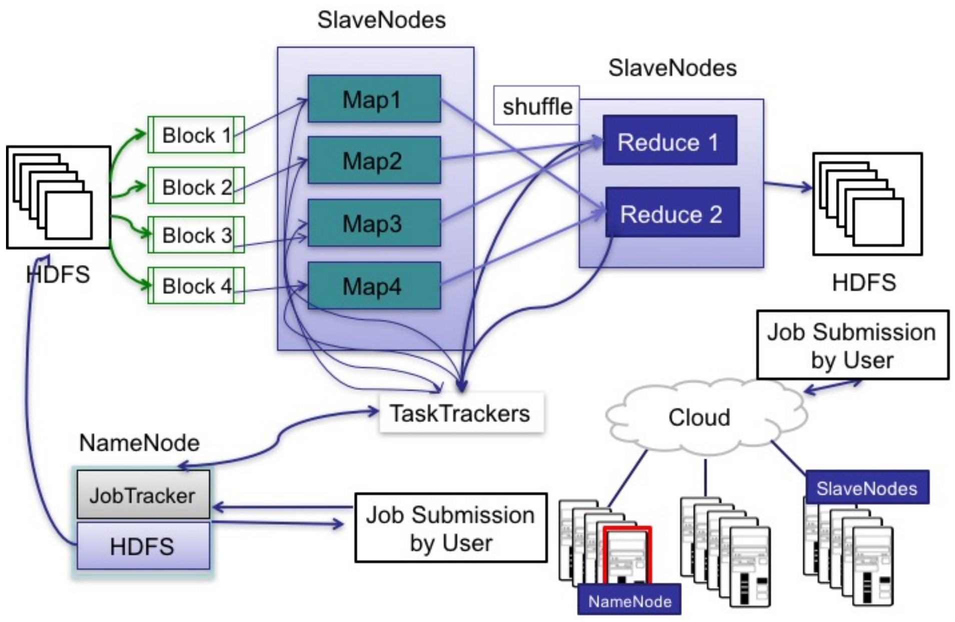


Figure 13: MapReduce WorkFlow

major MapReduce workflows and data movements. After the job submission, the input data split into blocks of data. The number of mapper and reducer functions plays a vital role to decide how MapReduce jobs are running on a Hadoop cluster. Based on the design of the MapReduce platform, there is a critical data movement phase when a job is running (called shuffle) that represents the output of a mapper function that is transferred to reduce functions for the final processing. How fast the shuffle phase is completed can affect the overall job completion time. We summarize the possible situations where various traffic patterns might occur in a Hadoop cluster:

- HDFS management such as a cluster health check
- File reads and writes from HDFS such as data replication, MapReduce input and output, and Cluster balancing

- Data shuffle among data nodes
- Interaction between TaskTracker such as data shuffles from mapper to reducer functions and data write back to HDFS as the final output.

### 2.3.3 MPEG-DASH

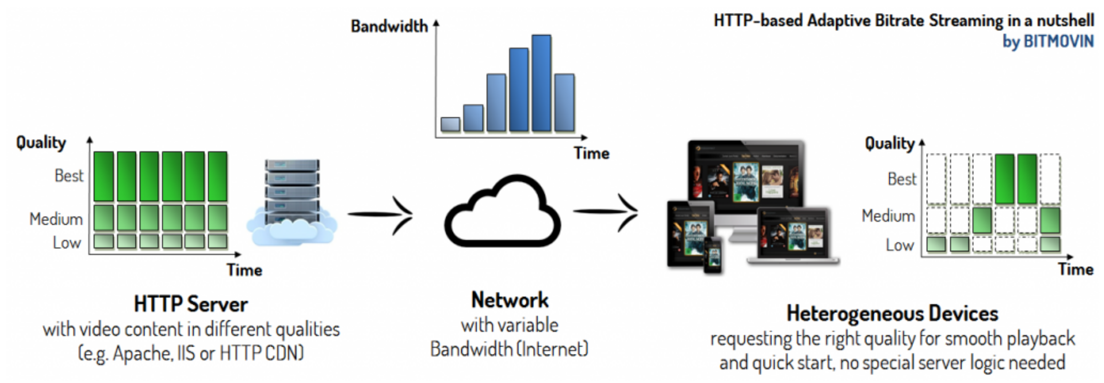

HTTP-based Adaptive Bitrate Streaming in a nutshell
by BITMOVIN
Bandwidth
Time
Quality
Best
Medium
Low
Time
HTTP Server
with video content in different qualities
(e.g. Apache, IIS or HTTP CDN)
Network
with variable
Bandwidth (Internet)
Quality
Best
Medium
Low
Time
Heterogeneous Devices
requesting the right quality for smooth playback
and quick start, no special server logic needed

Figure 14: MPEG-DASH System Overview [51]

MPEG-DASH (ISO/IEC 23009-1) is a vendor-independent, international standard. It bears the same video streaming approaches such as Apple HLS, Microsoft Smooth Streaming, and Adobe HDS. The goal of the DASH approach is to provide continued adaptation to the bandwidth situation of the client, reduce playback delay and increase the scalability and flexibility of client's adaptive bitrate (ABR) schema. It utilizes the existing HTTP-based CDN and caches related techniques to bypass NATs and firewalls and ease the network packet traversal in the network. The fundamental idea of DASH is depicted in Fig 14 [51]. A Media Presentation Description (MPD) file stored on the server side depicts the metadata of video segments such as segment durations, video/audio codec, bitrate, video resolutions, and how segments are stored indicated by segment reference schemes The client first fetches the MPD file to learn the URLs of all video segments. Thus, the video segments available from a video server. The ABR adaptation mechanism is conducted on the client side for each segment to determine which quality of a video segment is to be fetched. Bitrate switchover happens when network bandwidth or client buffer changes. By adaptively changing the downloaded bitrate, it provides a smooth playback experience for the user without much rebuffer situations.

# CHAPTER 3

# PROPOSED ARCHITECTURE

In this section, we discuss the status for our design approach of AAN-SDN architecture and preliminary results w.r.t our two use cases: Hadoop and MPEG-DASH.

## 3.1 General AAN-SDN Architecture and Implementation

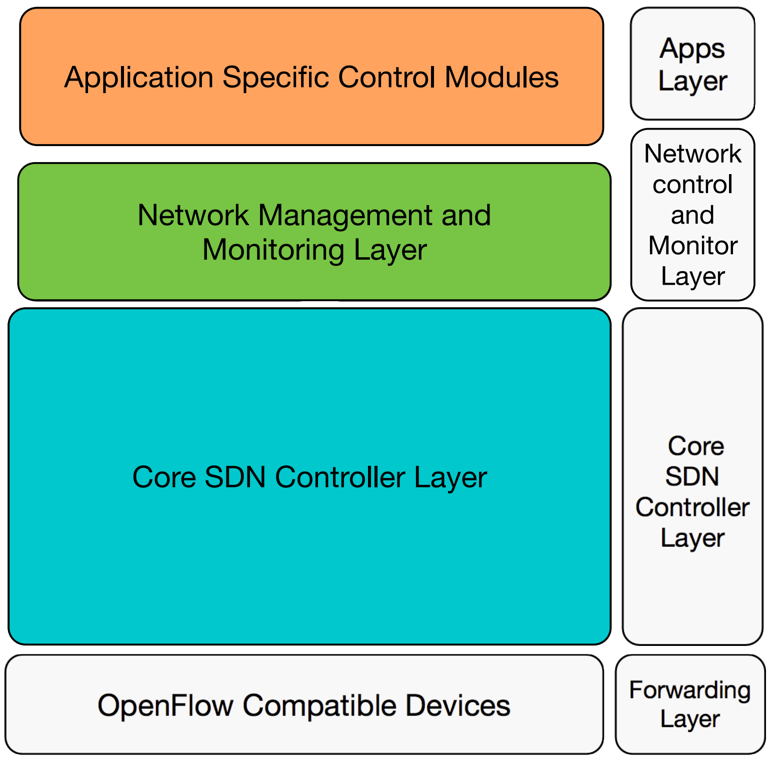

Application Specific Control Modules
Apps Layer
Network Management and Monitoring Layer
Network control and Monitor Layer
Core SDN Controller Layer
Core SDN Controller Layer
OpenFlow Compatible Devices
Forwarding Layer

Figure 15: General AAN-SDN Architecture

In this section, we present our proposed AAN-SDN platform design and implementation in general. We first introduced our layered SDN network architecture then conducted SDN-assisted MapReduce job completion time optimization. Starting from the bottom to the top layer, our proposed architecture (see Fig. 15) segregated our design into three main components:

- Core SDN Controller Layer
- Network control and monitor Layer
- Application-specific Layer

### 3.1.1 Core SDN controller Layer

In the core SDN layer, we implemented two network modules, Packet Forwarding, and Traffic Monitoring. In the packet forwarding module, we applied the network primary forwarding functions including the link layer discovery protocol (LLDP) in the physical network layer. The implementation was based on OpenFlow-compatible forwarding device. In the network layer, we implemented the forwarding function for the Internet Control Message Protocol (ICMP) messages, which is the key mechanism used to give feedback on network problems that could prevent packet delivery. Due to the flexibility provided by the SDN framework, we also addressed a new physical layer flooding avoidance mechanism such as for the address resolution protocol (ARP). In a traditional IP network, variations of spanning tree protocols (STP) are widely used to build a loop-free topology. The configuration of such an STP protocol can be cumbersome and complicated based on the used forwarding devices. We designed and implemented an ARP resolver in Algorithm 1 that offers smooth ARP package flooding, instead of using a costly STP protocol as would be the case in a traditional IP network environment. It also takes care of ARP cache expiration issues by avoiding to send additional ICMP messages to get an updated ARP entry.

Above the network transport layer, we implemented TCP and UDP packet forwarding functions for application-aware networking. Based on the application layer's port number and protocol type, it will forward packets accordingly. In the traffic monitor module, we implemented lightweight REST-API services to proactively fetch global network information such as port traffic for each forwarding device, flow installation/-modification, and traffic details in a managed time interval. The REST-APIs are designed to be lightweight without introducing extra overhead for the SDN controller. One Apache web server collects the pulled results from the REST-APIs and aggregates traffic details to provide any traffic alerts and Traffic Engineering (TE) recommendations.

### 3.1.2 Network Control and Monitor Layer, and Adaptive Traffic Engineering

In the network control and monitoring layer, the global network topology was discovered where we took an adaptive traffic engineering approach by feeding into a shortest path algorithm module to calculate a path for each pair of network node/hosts on an on-demand basis. The traffic monitor component, using REST-APIs' services, deployed at the core SDN controller layer proactively pulled network traffic information from the network. If there were any pre-defined traffic priority violations, a traffic reroute using a flow

**Algorithm 1:** ARP Resolver Algorithm

```
Data: ARP Flows $ARP_r$
Result: ARP Processing Decision (Forwarding/Blocking/NoAction)
ARP cache initialization for each connect switch;
Read incoming ARP packets: ;
if is ARP Broadcast then
    if ARP cache is empty then
        Add this entry to ARP Cache;
        Add expire timer to this entry;
        Flood this packet;
    else
        if exists then
            if entry timer $=< ARP\_Timer$ then
                Renew entry timer;
                Do not flood this packet;
            else
                Renew entry timer;
                Flood this packet;
            end
        else
            if is coming from a different port from existing entry then
                Do not flood this packet;
            else
                Add this entry to ARP Cache;
                Add expire timer to this entry;
                Flood this packet;
            end
        end
    end
else
    Forward ARP Request/Reply Packet;
end
```

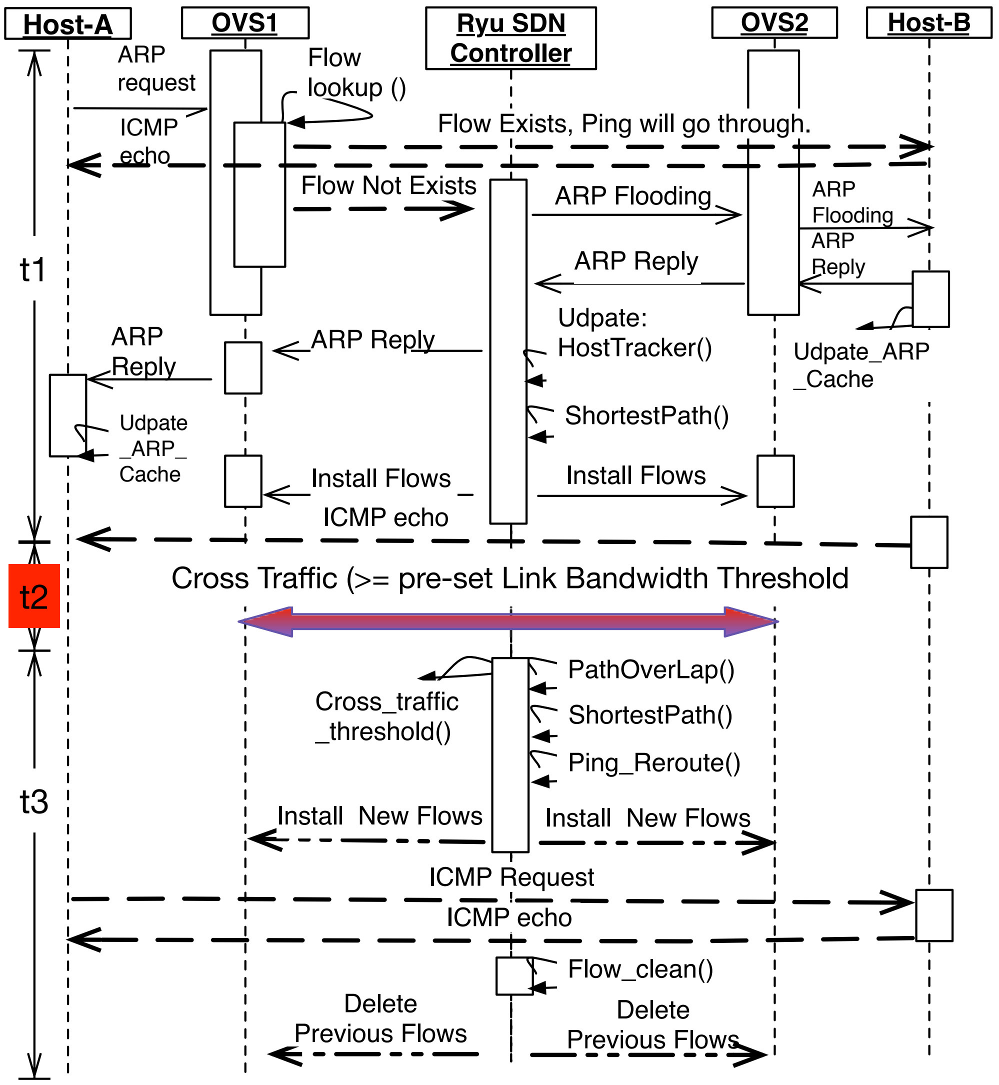


Figure 16: Traffic Reroute Workflow Example

reroute component might happen as explained in Fig. 16 that depicts the primary traffic reroute workflow. From the beginning, the ARP message for a network request such as Ping, SSH, or other applications. It first looks at the flow table and passes the traffic if there is an existing matching flow or checks if there are ARP broadcasting messages, otherwise. Flows are installed based on the path provided by the shortest path components. Flow rereoutes can happen when background traffic (at time $t2$) on the same route has over saturated some of the links along the path through adaptive traffic engineering. For this, the SDN controller recalculates a second shortest path in real time and installs new flows to reroute the application's traffic.

### 3.1.3 Application Layer

In the Application layer, a port number based application recognition feature is implemented (such as port 22 is by default for the remote secure shell (SSH)). In our controlled network, the port number can be managed/changed via a separate configuration file that is read by our SDN controller. The modularization of various components provided by different SDN controllers helps the network administrator to control them individually in a manageable way. With regard to our specific use cases, we can resue our foundemental AAN-SDN design and add specific application layer modules:

*Hadoop MapReduce AAN-SDN* It shows the application layer modules for Hadoop MapReduce. It includes HDFS and MapReduce control components are implemented to instruct how to install flows regarding Hadoop file operations and job assignments, accordingly.

*MPEG-DASH AAN-SDN* It shows the application layer modules for MPEG-DASH. With regards to DASH streaming applications, web server and DASH client control components are implemented to instruct how to install flows over forwarding devices. The modularization of various components provided by different SDN controllers helps the network administrator to control them individually in a manageable way. With regarding to recent Immersive video content such as VR/360 streaming over HTTP. MPEG-DASH has such support using Spatial Relationship Descriptor (SRD) in MPD file. The SRD parser module can parse the MPD file and extract the tile coordination based on user's ROI change. Shortest path module can calculate in real-time for all shortest paths between each pair of nodes based on existing network conditions and topology changes. In our test scenarios, higher priority tiles are rerouted using flow reroute module to the non-bottle-necked path to achieve better bandwidth utilization.

We discuss the specific application layer details for each of the mentioned applications, which are the Hadoop MapReduce and MPEG-DASH , in Chapter 4 and 6.

CHAPTER 4

# AAN-SDN FOR HADOOP

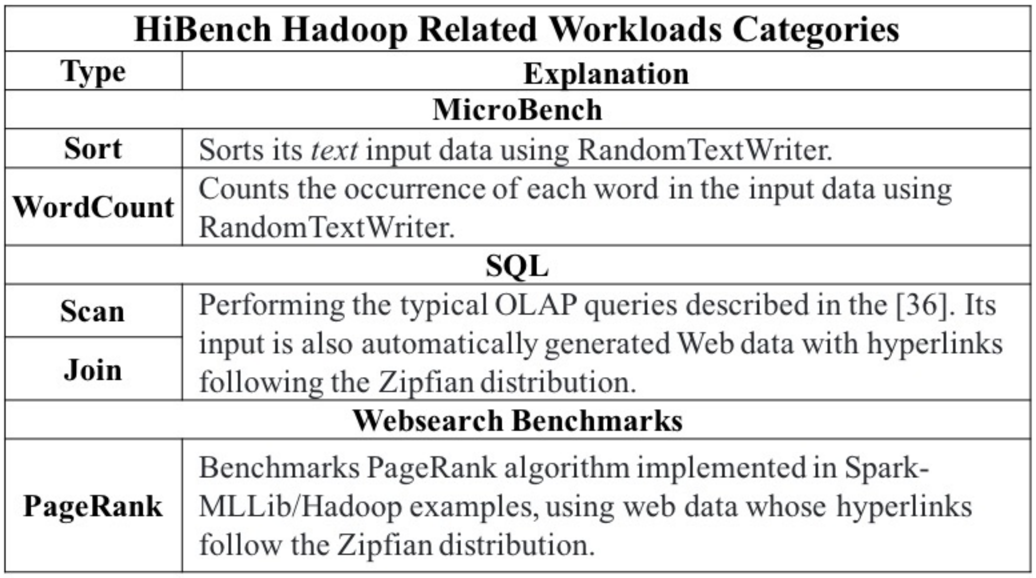
In this section, we first introduce HiBench, which is a Hadoop MapReduce benchmark suite. We ran HiBench using real-world MapReduce applications to understand the depth of MapReduce traffic pattern with various Hadoop configurations such as hardware and network topology. We then discuss our AAN-SDN design for Hadoop MapReduce and evaluate how proposed architecture can improve the Hadoop runtime efficiency

## 4.1 HiBench: Bigdata Micro Benchmark Suite

In this section, we first introduce HiBench, which is a Hadoop MapReduce benchmark suite. We ran HiBench using real-world MapReduce applications to understand the depth of MapReduce traffic pattern with various Hadoop configurations such as hardware and network topology. We then discuss our AAN-SDN design for Hadoop MapReduce and evaluate how proposed architecture can improve the Hadoop runtime efficiency

### 4.1.1 HiBench Summary

HiBench [30] is a big data benchmark suite that helps evaluate different big data frameworks in terms of speed, throughput, and system resource utilization. Hadoop MapReduce workloads in the HiBench benchmark suite include a number of applications such as Sort, WordCount, SQL, and PageRank. Due to our goal of understanding the



| HiBench Hadoop Related Workloads Categories | |
|---|---|
| **Type** | **Explanation** |
| **MicroBench** | |
| **Sort** | Sorts its *text* input data using RandomTextWriter. |
| **WordCount** | Counts the occurrence of each word in the input data using RandomTextWriter. |
| **SQL** | |
| **Scan** | Performing the typical OLAP queries described in the [36]. Its input is also automatically generated Web data with hyperlinks following the Zipfian distribution. |
| **Join** | |
| **Websearch Benchmarks** | |
| **PageRank** | Benchmarks PageRank algorithm implemented in Spark-MLLib/Hadoop examples, using web data whose hyperlinks follow the Zipfian distribution. |

Figure 17: HiBench Hadoop MapReduce Related Workload Summary

resources' usage and shuffle data traffic pattern and to later use this in our AAN-SDN environment, we use the workloads' categories depicted in Fig. 17 based on given hardware resources.

### 4.1.2 HiBench MapReduce: Configuration and Initial Job Runtime Data Collection

| Test Configurations | A | B-1 | B-2 | B-3 | B-4 | C-1 | C-2 | D-1 | E-1 | E-2 | E-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Data Size (MB) | 16 | 48 | | | | 240 | | 304 | 304 | | |
| Slave Allocation | TA | TB-1 | | TB-2 | | TC | | TD-4 | TE | | |
| (# of Mapper,<br># of Reducer) | (1, 1) | (1, 1) | (2, 2) | (1, 1) | (2, 2) | (4, 4) | (8, 8) | (8, 8) | (4, 4) | (8, 4) | (8, 8) |
| MicroBench Sort | | | | | | | | | | | |
| MicroBench WordCount | | | | | | | | | | | |
| SQL Join and Scan [11] | | | | | | | | | | | |
| Join and Scan Configurations | Universities=88000<br>pages=6 | Universities=265000<br>pages=6 | | | | Universities=1325000<br>pages=6 | | Universities=1680000<br>pages=6 | Universities=1680000<br>pages=6 | | |
| WebSearch PageRank | | | | | | | | | | | |
| PageRank Configuration | pages=38200<br>num_interactions=1<br>block=0<br>block_width=16 | pages=110000<br>num_interactions=1<br>block=0<br>block_width=16 | | | | pages=480000<br>num_interactions<br>block=0<br>block_width=16 | | pages=610000<br>num_interactions<br>block=0<br>block_width=16 | pages=610000<br>num_interactions<br>block=0<br>block_width=16 | | |

Figure 18: HiBench MapReduce Workloads Configuration Details

Before considering our AAN-SDN framework, we conducted a set of measurements on the Hadoop HiBench benchmark suite while keeping the topology simple and static to understand traffic patterns. We first discuss HiBench configurations used for this work.

The main considerations for running a MapReduce job include the job input size, the number of running slave nodes, the number of mappers and reducers, and the block size and location of the slaves. The Hadoop file block size was set to $32MB$ due to our limited hardware resources for the experimental platform and the replication factor is set to 2. Additional Hadoop related configurations are displayed in Fig 18. One of our goals in this work is to understand the shuffle traffic of Hadoop M/R. The selected configuration parameters are the key control points for our proposed AAN platform. For the rest, Hadoop default parameter values were used.

The small files problems [26] are avoided by limiting the total given file size. We setup a HiBench configuration based on a Hadoop MapReduce job type and the network topology locations for slave nodes as shown in Fig 18. Consider test configuration "B-1" as an example to explain our setup of HiBench: its input file size is $48MB$. The number of mappers and reducers set to one each. At this configuration setup, we compare the runtime difference with other "B-X" configurations as well as other test configurations using different HiBench settings. Slave locations are depicted in Fig. 19. For a TA type, only one slave is active for the submitted job. A TA case sets up a base comparison with other topology configuration types for the same MapReduce job with respect to job CPU, memory consumption and how long does it take to complete the job. We also ran a

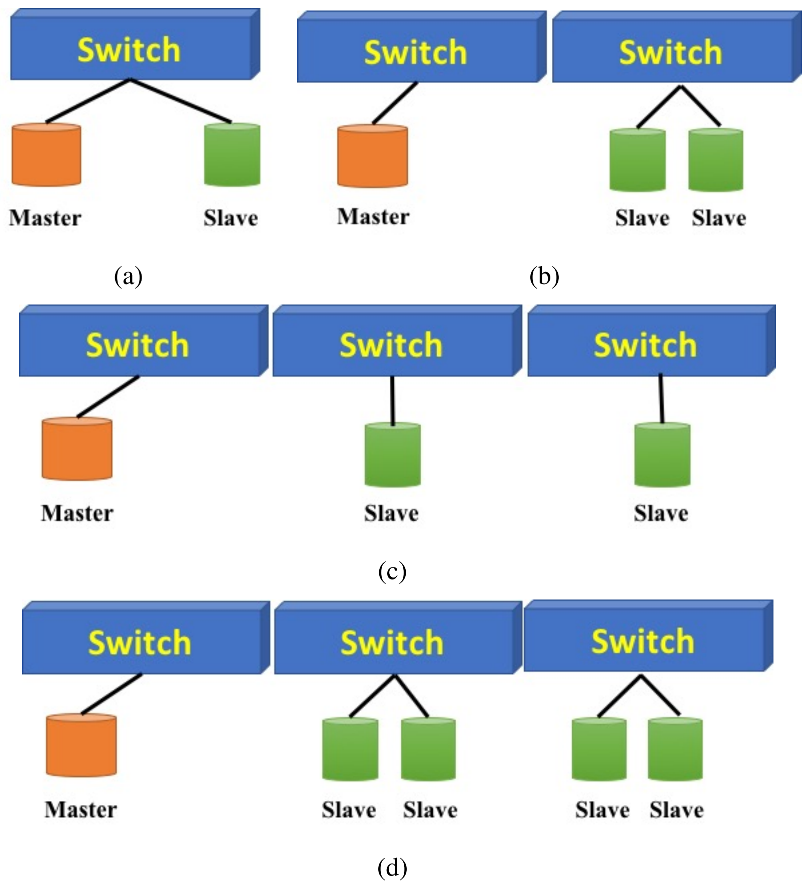


Figure 19: Topology Type (a) TA, (b) TB-1, (c) TB-2, (d) TC, TD

MapReduce job under different slave locations using a various number of slave nodes to understand if the location of the slave could affect the job completion time. For example in the TB-1 and TB-2 configuration setup, two slaves ran under the same topology location in TB-1 while they were in different locations for TB-2. A TE type uses eight slave nodes that are evenly distributed under four forwarding devices as shown in Fig. 29(a).

Some of the predefined MapReduce jobs in HiBench can be assigned a particular data size for input such as its micro workloads, WordCount and Sort, while others use different input parameters such as for Join and Scan [56] from the SQL workload. For example, a Join workload using two parameters to set up the workloads, “number of Universities” and “pages”, instead of specifying the size of the input data. To ensure our input data size is the same, we conducted an initial experiment and derived the following relation based on empirical runs:

*HiBench SQL workload: Join, Sort, Aggregation*

$$Input\_Size(MB) = \frac{Number\ of\ Universities * 1.8}{10^5}. \tag{4.1}$$

*HiBench WebSearch workload: PageRank*

$$Input\_Size(MB) = \frac{Number\ of\ Universities * 2}{6\sqrt{2} * 5000}. \tag{4.2}$$

We determined that only the parameter, “number of Universities”, can change the input data size for workloads such as Join, Scan, and PageRank. By varying the predefined parameters, we keep our test configuration consistent with the same amount of input data size for different workloads.

## 4.2 Data Collection Cases

Hadoop MapReduce hardware resource data collection is conducted for a set of small static topology configurations. The goal is to understand the runtime resource consumption such as CPU, memory, and job completion time for each HiBench workload that is depicted in Fig 18.

Altogether, there were 330 tests. Because of the size of the test cases, was ran each test three times for each configuration to compute the average value; we also calculated the $95\%$ confidence interval $(CI)$ to determine the oscillatory nature of the MapReduce jobs.

Table 4 lists the detailed values for reference. Based on our observations that different HiBench jobs behave differently, we first examined the resource usage separately from each other for each HiBench configuration. We will then present a combined analysis.

#### 4.2.1 Average CPU Usage Summary

Fig. 20 depicts the average CPU usage for various configurations. From the runs, we made the following observations:

- The single slave node uses CPUs most extensively, compared with other configurations. It has the highest CPU usage for a single slave node.
- When the number of mappers and reducers increases, the CPU usage increases slightly using the same input size, such as B-1 vs. B2 and B3 vs. B4. Even though for E-X configurations there is a slight drop for some jobs such as WordCount and Scan, the values are still within the $95\% CI$ range. The reason for such behavior is due to over resource allocations since our testbed had limited hardware.
- When the number of slave nodes increases, the average CPU usage decreases in most our cases; for example, compare between D-1 and E-X.
- The location of a slave node in our setup does not play a significant role in affecting the overall CPU usage with consistent bandwidth allocation on each interconnected link.
- The overall CPU usage has significant oscillation for the same set of test configurations even with no background processes running on the same system. The main

reason is due to the efficiency of the JVM resource allocation and release in the MapReduce platform at the runtime environment.

- PageRank has a higher CPU usage compared to other jobs, while others show similar CPU usage trends.

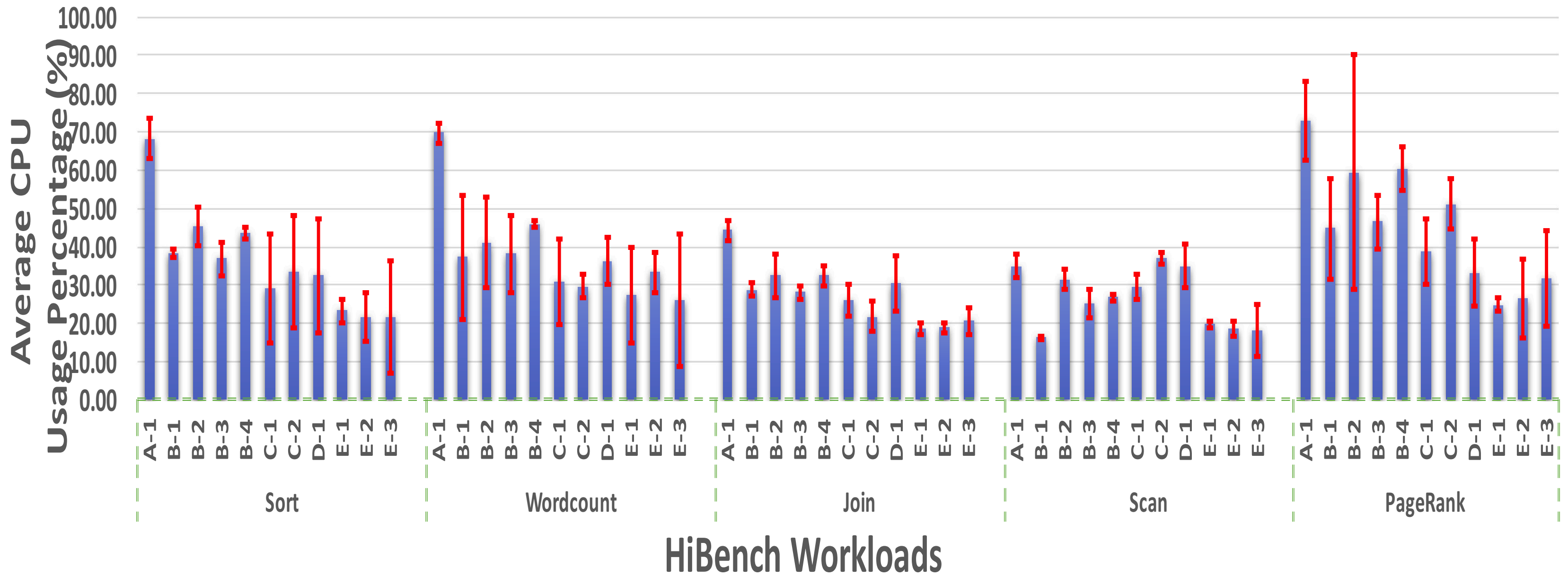


Figure 20: CPU Load Summary

### 4.2.2 Average Memory Usage Summary

Fig. 21 depicts the average memory usage for various HiBench configurations. From the runs, we observed the following:

- A single slave had the lowest memory usage even with the lowest input data size when comparing configuration A-1 with others.
- The average memory usage increases dramatically when the data size increase. For example, the average memory usage increased nearly $500\%(\approx 2500/500)$ compared to six times in data growth$(\approx 304/48)$.
- The memory usage was in direct proportion to the number of allocated mappers and reducers. However, in some cases, we noted opposing results, particularly in the C-2 and E-X cases. Even though they still fell into the calculated $95\% CI$ range, the reason for such behavior was due to over-resource allocations given our hardware's limitations.
- The overall memory usage had less oscillation for the same set of test configurations compared to the CPU usage.
- All of the jobs had the same level of memory usage when compared to the same set of configuration runs.

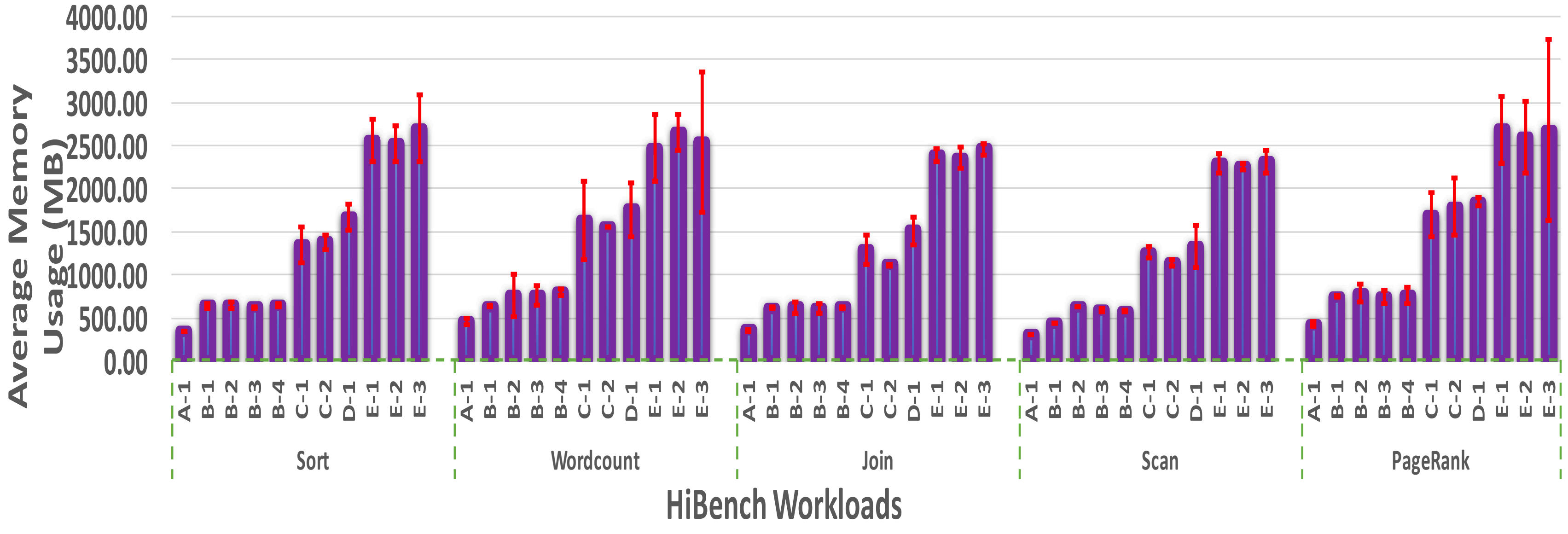


Figure 21: Memory Load Summary

### 4.2.3 Job Completion Time: Comparison

Fig. 22 depicts the average job completion time for various configuration setups. From the runs, we observed the following:

- The input size had the largest impact on the average job completion time. The larger the data size, the longer it took, such as the comparison between C-X and D-1. However, with the same input size, more slave nodes reduced the overall time, such as the comparison between D-1 and E-X.
- The number of mappers and reducers for the same set of HiBench workloads also played an important role in the completion time in most configuration cases except for Scan. However, in some cases, it showed the opposite results such as in PageRank. The slight time increased from E-2 to E-3 was due to over resource allocations on the given hardware's limitation.
- Pagerank has a significant time increase compared with other workloads due to its CPU-intensive nature.

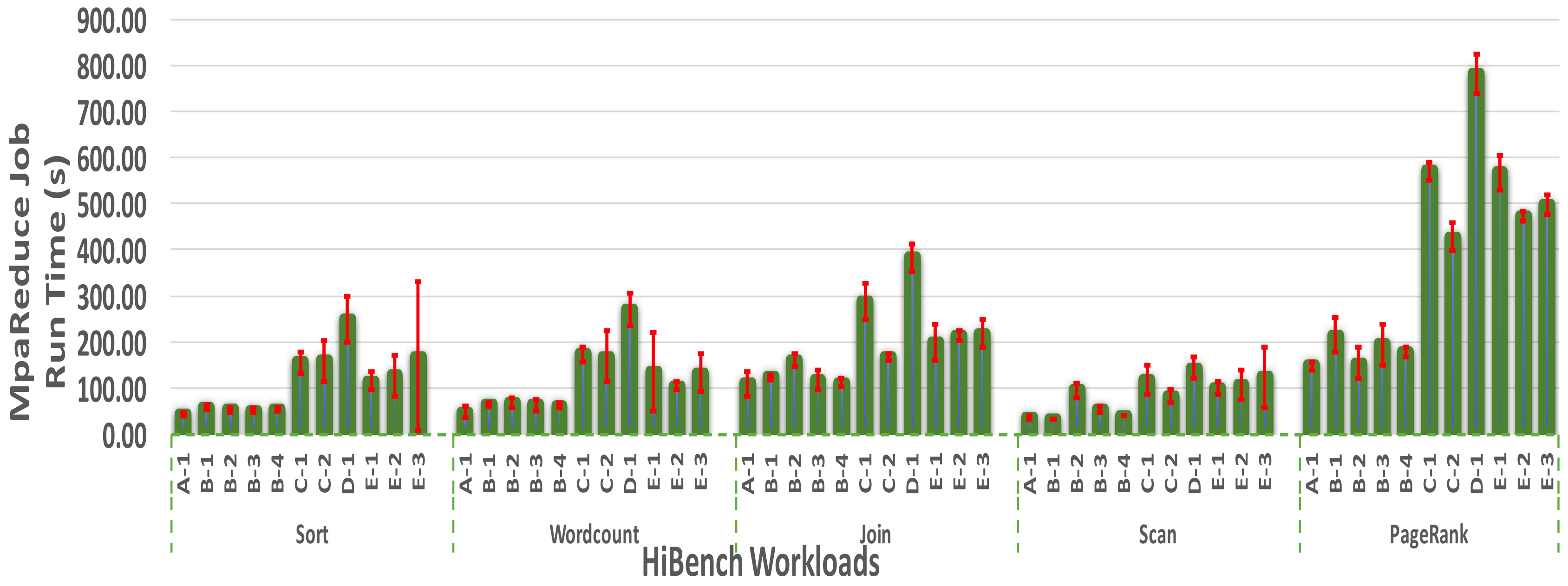


Figure 22: Job Completion time Summary

In summary, each HiBench workload behaved differently even with similar configuration setups. Clearly, understanding the different behaviors of various MapReduce jobs was an essential step for us for our SDN-based AAN environment for MapReduce applications.

## 4.3 AAN-SDN Hadoop Architecture and Implementation

In this section, we present our proposed AAN-SDN platform design and implementation. For implementation of SDN, we used Ryu [55] and OpenFlow v1.3. We first introduced our layered SDN network architecture then conducted SDN-assisted MapReduce job completion time optimization. Starting from the bottom to the top layer, our proposed architecture (see Fig. 23) segregated our design into three main components:

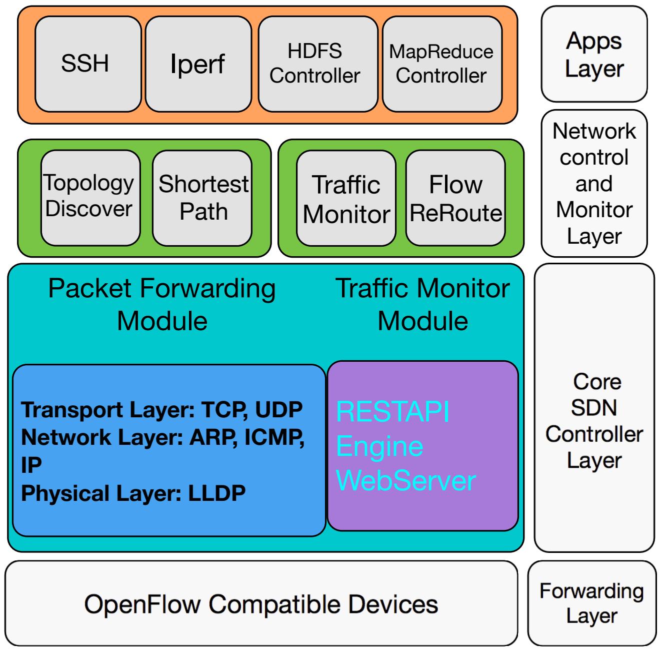


Figure 23: SDN Hadoop Experimental Architecture

### 4.3.1 Application Layer

In the Application layer, a port number based application recognition feature is implemented. In our controlled network, the port number can be managed/changed via a separate configuration file that is read by our SDN controller. With regards to Hadoop applications, HDFS and MapReduce control components are implemented to instruct how to install flows regarding Hadoop file operations and job assignments, accordingly. The modularization of various components provided by different SDN controllers helps the network administrator to control them individually in a manageable way. Four groups of application layer components are deployed:

- SSH is used for remote access control. Hadoop Master node uses SSH to start/end slave nodes' services and other control messages.
- Iperf is implemented as background traffic injections.
- HDFS controller is deployed to control files and write/read in a Hadoop cluster.
- MapReduce controller is used for job assignment and monitoring services.

## 4.4 MapReduce Traffic Optimization Using SDN

In this section, we report on experiments conducted using our proposed AAN-SDN platform to optimize MapReduce job running oversaturated network links.

#### 4.4.1 System Workflow

Fig. 24 explains the workflow of our running system. After the Hadoop and HiBench configuration are completed, the SDN controller starts to run Hadoop MapReduce jobs. While such jobs are running, the SDN controller will install flows based on application types and collect network traffic information accordingly.

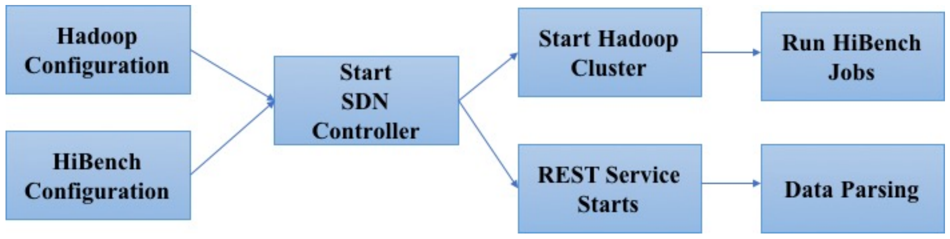


Figure 24: Experimental Network Setup

## 4.5 Performance Evaluation

### 4.5.1 Proof-of-Concept

In this section, we used a small set of M/R job and a relative small ring network topology to test the usability of proposed AAN-SDN architecture.

#### 4.5.1.1 Loop Network Topology

In this experiment, the testbed is created using virtual machines (VMs) in the Global Environment for Network Innovations (GENI) [19] platform. Fig. 25 shows our topology setup. We created a ring topology using five OpenVSwitches (OVSs) as forwarding devices. To illustrate the impact, we consider a Hadoop cluster with one Hadoop master and two data nodes (DN) at two different locations, along with two iPerf traffic generators for background traffic. All of the VMs have one core X5650@2.67GHz, 880MB memory. The bandwidth between the OVSs and host was set to 100 Mbps. The link threshold was set to 50 Mbps between OVS1 and OVS5.

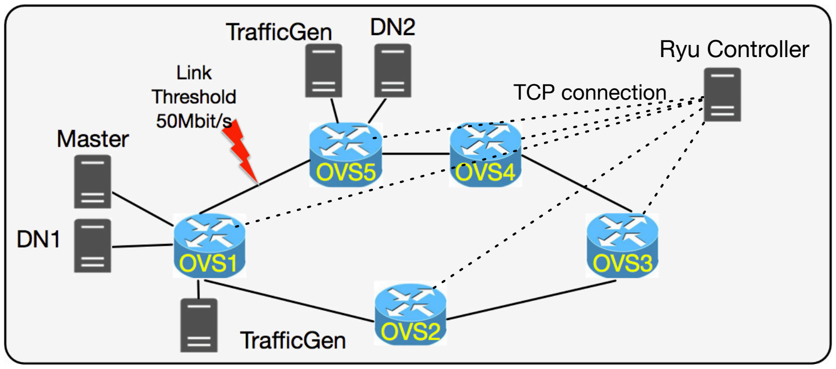


Figure 25: Loop Network Topology Setup

#### 4.5.1.2 Preliminary Results

In this environment, we tested Hadoop M/R performance by running the WordCount program with the following file sizes: 200 MB, 300 MB, and 400 MB. Fig. 26 shows that Hadoop WordCount runtime decreases by an average of 23% when using our AAN platform.

Fig. 27 shows the SDN controller system load based on the number of flows. Before a re-route happens, flows traverse 2 switches (OVS1 and OVS5), and the CPU load increases significantly from 50% to 100% while the flows' number increases from 50 to 100. When the re-route happens, the CPU load is higher using 5 switches compared with using 2 switches. In both cases, the CPU load keeps high occupancy after 100 flows.

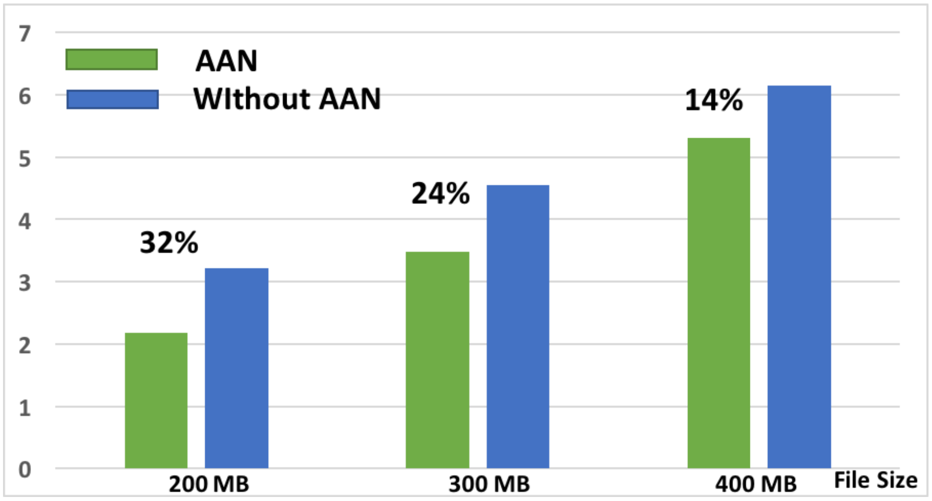


Figure 26: Hadoop M/R runtime Comparison

The CPU load hits the bottleneck around 500 flows before the flow re-route and 100 flows after. However, the system memory use was steady around 6% in both cases.

The SDN controller sends the flow installation and delete instructions through the TCP connection among all connected switches. Fig. 28 shows that the control traffic increases when the number of flows increases. The control traffic size increases go up from 500 Kbps to 800 Kbps when the number of flows changes from 50 to 100; the rate of increase slows down from 100 to 500 flows. It finally hits the transfer bottleneck after 500 flows. Thus, an important factor to address with Hadoop in an AAN environment is to ensure that the SDN controller can handle a sufficiently large number of flow re-routes without hitting the bottleneck.

In this experiment, we presented an application-aware networking environment using SDN for Hadoop M/R applications running in a distributed environment. We used GENI testbed to test the proof-of-concept where flow rerouting aliviates performance

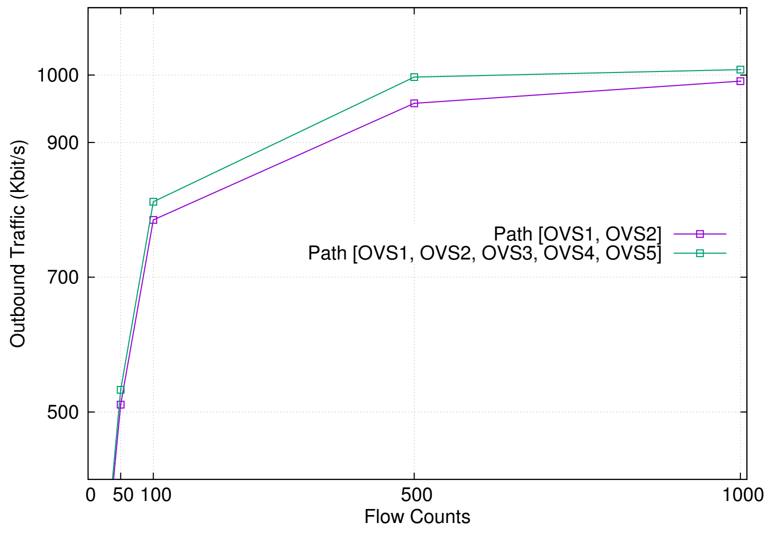


Figure 27: SDN Controller System Load

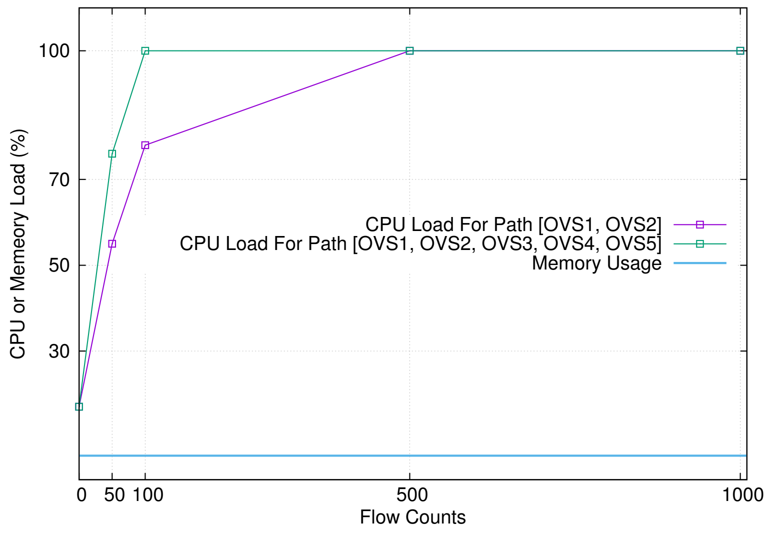


Figure 28: Control Traffic

bottleneck. We conducted a proof-of-concept experiment to show that the AAN approach reduces the compute time. The flow re-route results show that the SDN controller is CPU intensive.

## 4.5.2 Scale up with HiBench and Larger Topology

In this section, we tested the proposed AAN-SDN architecture with HiBench, a larger Hadoop cluster and network topology.

### 4.5.2.1 Data Center Network Topology

We deployed our SDN-based Hadoop and HiBench environment on the GENI testbed [19] platform. We setup a network topology as shown in Fig. 29(a) to emulate a data center network topology with hosts associated with different network switches that correspond to Hadoop nodes. We used Openvswitch (OVS v2.3.1) [55] as our forwarding devices and numbered the DPID in order from '1' to 7'. One Hadoop master node and eight slaves were deployed. The deployed nodes have the same hardware configuration with a single core of Intel(R) Xeon(R) CPU X5650 @ 2.67GHz and 8 GB RAM. Each connected link has $100\ Mbps$ bandwidth allocation.

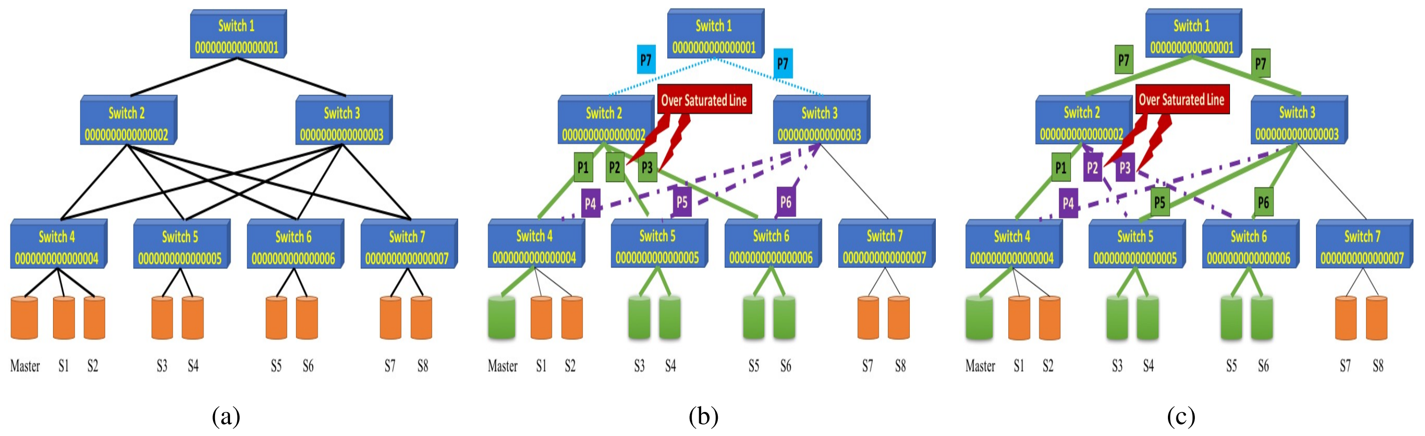


Figure 29: (a) Network Topology, (b) Traffic Path Before Reroute, (c) Traffic Path After Reroute

#### 4.5.2.2 Network Flow Traffic Capture

Network flow traffic is captured when running the HiBench workload using SDN by designed REST-API services, which proactively pulls global network traffic information based on any given port number. Table 2 shows the related port number regarding our MapReduce benchmark tests. For example, data shuffling traffic is captured during the map-reduce shuffle phase using port number 50010.

Table 2: Monitored Port Number

| Traffic Catagory | Explanation | Port Number |
|---|---|---|
| HDFS | Hadoop File System Operation (HDFS) | 50010 |
| | Hadoop Datanode Transfer | 22 |
| | Master Node Http IPC | 54310 |
| YARN | Yarn Resource Scheduler | 8030 |
| | Yarn Resource Tracker | 8031 |
| | Yarn Resource Manager | 8032 |
| Others | Iperf | 5001 |
| | Secure Shell (SSH) | 22 |

Fig. 30 depicts the data shuffling pattern for each workload. Detailed flow traffic is listed in Table 5. The major traffic has been captured from port number 50010, the majority of this being from the MapReduce shuffling phase. We summarize our observations as follows:

1. The shuffling data size is in direct proportion to the input data. Except for the WordCount workload, which has minimal increase, the others have significant data flows. This is to show that WordCount has a minimal shuffling data size. The other

cases have more output data than the input size:

(a) The Sort workload has roughly $110\%$ of the input data size that gets shuffled.

(b) In the Join workload, around $70\%$ of the input data gets shuffled.

(c) In the Scan, around $120\%$ of the input data gets shuffled.

(d) In the PageRank about $240\%$ of the input data gets shuffled.

2. The number of mappers and reducers has no noticeable effect on the shuffling data sizes, such as the comparison between C-1 and C-2. The total shuffle size is the sum of individual traffic among possible slave nodes.

To have a deeper understanding of what composes the shuffle traffic, we capture the data transfer for each pair of slave nodes. Due to the nature of the replication factor we set, even though the sum of the total shuffle traffic is within our calculated $95\%CI$ range, the pairs of slave nodes that generate the shuffle traffic are not fixed for each run. However, we list a network trace in Table 6 to explain the different behaviors of each workload. By identifying the shuffling pattern, we understand what the traffic size is and which network link it takes to transfer the data. From our test results, we summarize our observations as follows:

1. The number of mappers and reducers plays important roles for shuffling pairs. For examples, in the B-1 and B-2 configuration with one and two mappers/reducers, respectively, there is only one pair of shuffling traffic on case B-1 but there might be two pairs on B-2. It mostly depends on the current system load and FIFO resource allocators.

2. The input data size also contributes to the number of shuffling pairs. For example, there is a significant number of pairs that increases from configuration B-X to E-X.

Consider workload Join with the E-3 test case for an example to note the randomness of the flow traffic pattern; the detailed traffic direction is listed in Table 6. There are 53 sets of individual shuffle pairs. The uncertainty of MapReduce traffic is not easy to address by a traditional IP network with less individual traffic flow control.

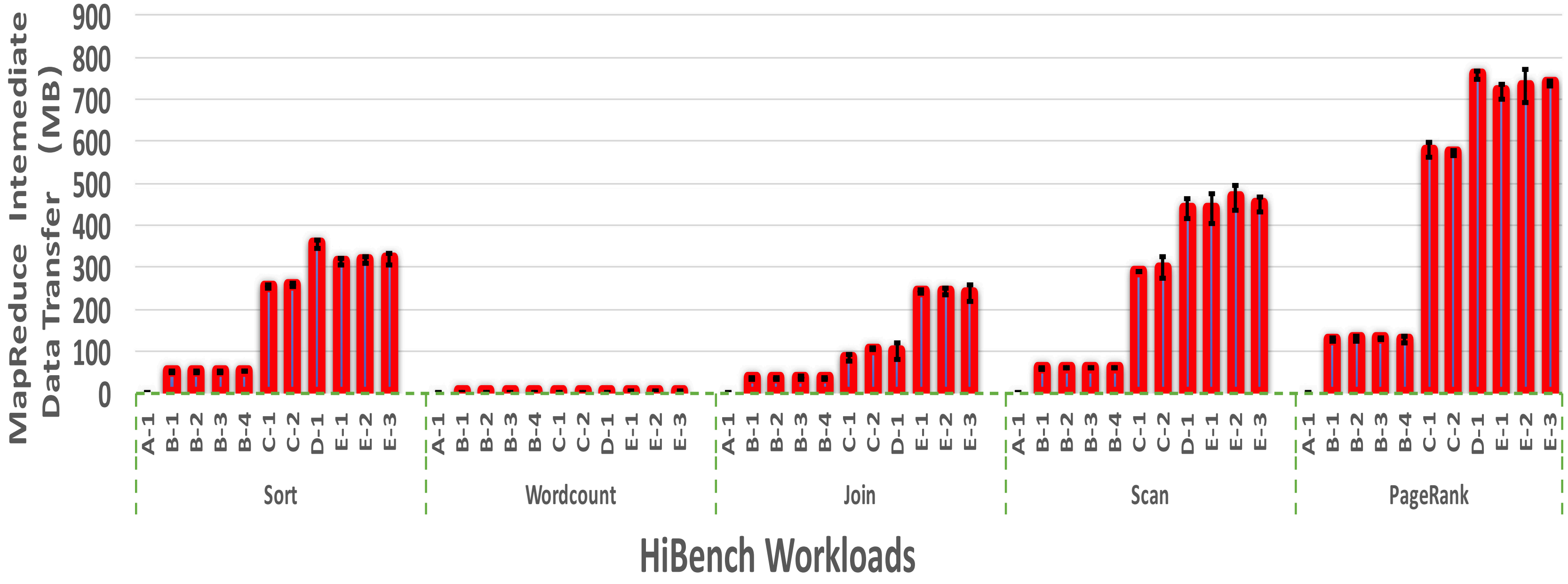


Figure 30: Hadoop MapReduce Intermediate Data Transfer

#### 4.5.2.3 ARP Flooding Avoidance Using ARP Resolver

Our proposed ARP resolver method (see Algorithm 1) provides smooth ARP cache expiration and packet flooding, instead of using a costly STP protocol as would be the case in a traditional IP network environment. We implemented the default ARP packet defined in RFC 826, which is 28 bytes. We then conducted a measurement on how the proposed ARP can avoid flooding issues using the minimum ARP request packets. If we use slave nodes S1 and S7 as an example, the shortest path is S1$\rightarrow$Switch 4$\rightarrow$Switch 2$\rightarrow$Switch 7$\rightarrow$S7. If S1 sends an ARP request, an ARP flooding packet must be sent out if the previous ARP cache has expired. If that is the case, the ARP flooding packet will be forwarded to all network forwarding devices by each other.

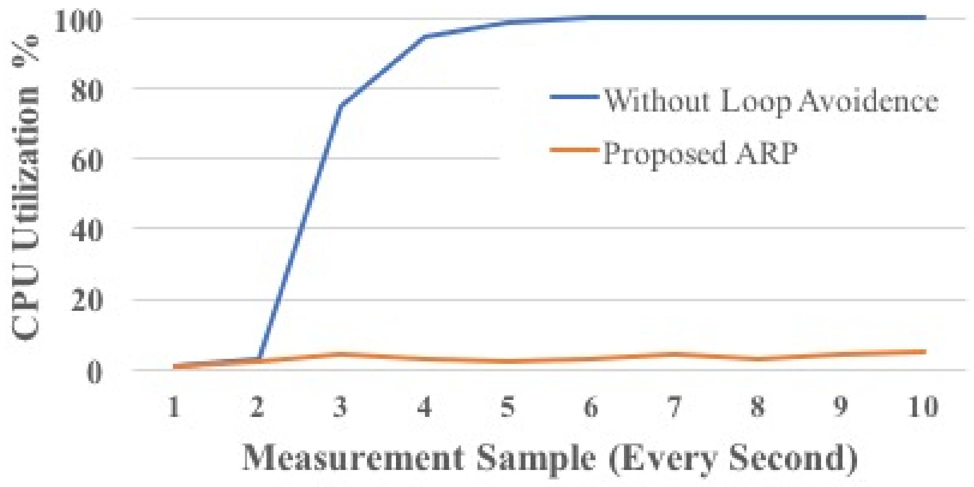


Figure 31: ARP Broadcasting Packet Cause CPU Utilization High in a Loop Topology

Based on our experimental result shown in Fig 31, the forwarding switch's CPU can run almost at $100\%$ utilization without any ARP flooding avoidance methods, which causes the network to stop functioning properly. However, it only takes 20 ARP flooding packets (Each ARP broadcasting packet can only be seen twice by the connecting port) to travel through the network by using our proposed ARP resolver. With the minimum number of flooding packets, the proposed ARP resolver algorithm can minimize the flooding traffic and destination searching time, without overwhelming the forwarding switch's CPU.

#### 4.5.2.4 SDN Traffic Reroute through Adaptive Traffic Engineering

The behavior of the data shuffle phase for an individual MapReduce workload has shown various data sizes and patterns based on our study. One can assume that if such a workload was running on a busy network link, the increase in delay would be expected. A traditional IP network lacks real-time global traffic information updates and the network administrator has less control over a specific network flow with the minimum cost. With regards to Hadoop MapReduce applications, data could shuffle from any pair of running slave nodes. If any delay happens on any of the shuffle phases, the overall job completion time can be prolonged.

Our goal of running Hadoop MapReduce jobs using the proposed AAN-SDN architecture is to investigate how much SDN can alleviate from a busy Hadoop cluster based on the different sizes of input data and the number of slave nodes. Table 3 shows experiment scenarios and test results. Two cases are considered. The first is $48MB$ input size with two slaves nodes and the second is $240MB$ input size with four slave nodes. To simulate a busy cluster situation, we induced background traffic using *iperf* on selected links.

Fig. 29(b) depicts the MapReduce job path utilization before reroute. The running Hadoop nodes are marked in green colors. They are the master node and S3, S4, S5, and S6 are the slave nodes. For reroute test case 1 (shown in Table 3), slave nodes S3 and S5 ran HiBench workloads. The shortest path module from the SDN platform installed flows along the path [**P1, P2, P3**] at the beginning of the system and ran in order to start the Hadoop cluster and start MapReduce jobs. Meanwhile, path [**P1, P7, P5, P6**] was saved as a backup shortest path between the master node and slave nodes S3 and S5.

Table 3: HiBench Workloads Traffic Reroute Using SDN

| Workloads | Sort | | WordCount | | Join | | Scan | | Pagerank | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ReRoute Test 1 | ReRoute Test 2 | ReRoute Test 1 | ReRoute Test 2 | ReRoute Test 1 | ReRoute Test 2 | ReRoute Test 1 | ReRoute Test 2 | ReRoute Test 1 | ReRoute Test 2 |
| Slaves | S3 S5 | S3 S4 S5 S6 | S3 S5 | S3 S4 S5 S6 | S3 S5 | S3 S4 S5 S6 | S3 S5 | S3 S4 S5 S6 | S3 S5 | S3 S4 S5 S6 |
| Input Data Size (MB) | 48 | 240 | 48 | 240 | 48 | 240 | 48 | 240 | 48 | 240 |
| No Background Traffic with path [ P1, P2, P3] | 49±2 | 338±56 | 56±3 | 220±17 | 171±12 | 423±38 | 116±7 | 146±19 | 190±18 | 713±33 |
| Background Traffic with path[ P1, P2, P3] | 66±4 | 483±40 | 75±2 | 393±27 | 216±4 | 616±63 | 151±27 | 208±43 | 256±56 | 1022±224 |
| SDN Reroute Path[ P1, P5, P6, P7] | 55±2 | 360±62 | 61±3 | 234±18 | 187±10 | 452±30 | 130±3 | 167±34 | 212±7 | 745±20 |
| Reroute Time Consumption | 7±1 | 24±10 | 8±5 | 14±27 | 16±13 | 29±15 | 14±5 | 21±14 | 23±16 | 31±15 |
| Improvement | 20.00% | 34.00% | 22.00% | 68.00% | 16.00% | 36.00% | 16.00% | 25.00% | 20.00% | 337.00% |

Iperf background traffic runs on the path [**P2, P3**] consuming $90Mbps$ bandwidth, which is our predefined threshold for any flow reroute scenario. We first disabled the reroute module and forced the MapReduce jobs to run on the oversaturated links between S3 and S5 to emulate a static environment. When the reroute module is enabled, the SDN controller detects there is data flow over an acceptable threshold of background traffic along the first pair of the shortest path. New flows are installed using the backup path [**P1, P7, P5, P6**] to avoid any potential delays as shown in Fig. 29(c).

In summary, Fig. 32 shows that with our adaptive traffic engineering approach through rerouting, the MapReduce Job completion time can be reduced by 16% to 300%

depending on test case configurations. The improvement is varied and it depends on the different behaviors of each MapReduce job. Consider the PageRank for example; its shuffle phase has over 200% more data output compared with the input data size. It also has the most runtime efficacy improvement in terms of job completion time. Even though the improvements for other jobs are not as significant as PageRank, it still shows an increasing trend when the data input size increases.

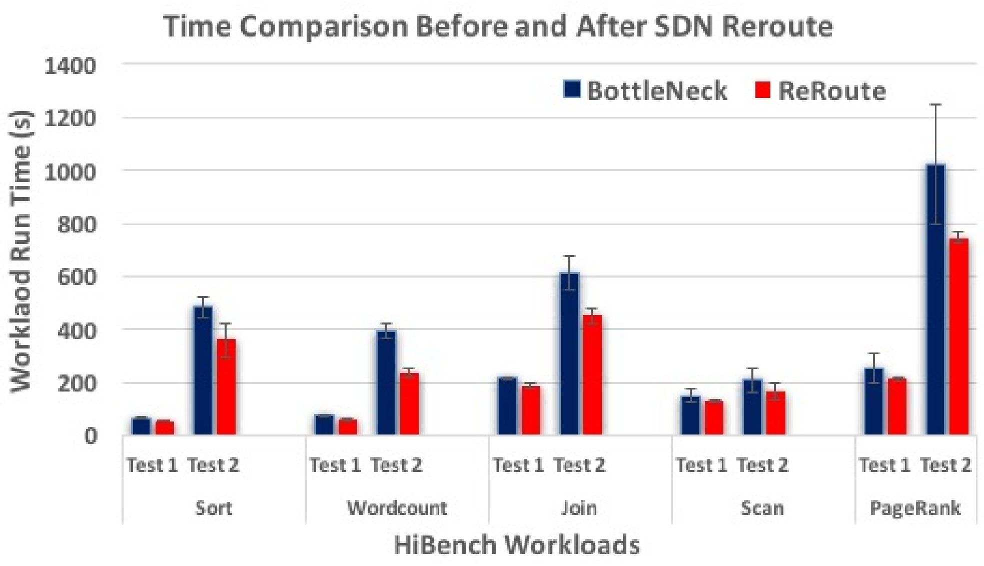


Figure 32: Hadoop Job Runtime Comparison

As our result shows, our AAN platform for Hadoop MapReduce job optimization offers a significant improvement compared to a static, traditional IP network environment. Our design can be extended to other MapReduce jobs and various network topology without much additional complexity.

Table 4: Hibench Workload: Hardware Related Data Collection

| **Configurations** | **A** | | **B-1** | | **B-2** | | **B-3** | | **B-4** | | **C-1** | | **C-2** | | **D-1** | | **E-1** | | **E-2** | | **E-3** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** | **Avg** | **95% CI** |
| **Sort** | | | | | | | | | | | | | | | | | | | | | | |
| Time | 42.00 | **4.30** | 58.67 | **6.25** | 52.67 | **6.25** | 51.67 | **3.79** | 52.33 | **3.79** | 155.00 | **22.77** | 158.67 | **45.83** | 248.33 | **49.33** | 115.67 | **18.97** | 127.00 | **43.53** | 169.00 | **160.09** |
| CPU | 68.17 | **5.17** | 38.27 | **1.23** | 45.33 | **5.09** | 36.80 | **4.35** | 43.70 | **1.51** | 29.10 | **14.33** | 33.53 | **14.62** | 32.37 | **14.90** | 23.17 | **3.11** | 21.67 | **6.48** | 21.67 | **14.69** |
| Memory (MB) | 342.43 | **7.40** | 638.60 | **31.18** | 641.13 | **44.53** | 620.70 | **10.82** | 641.13 | **13.53** | 1338.67 | **205.58** | 1380.97 | **85.03** | 1661.17 | **153.65** | 2556.47 | **254.89** | 2517.37 | **205.55** | 2694.87 | **387.48** |
| **WordCount** | | | | | | | | | | | | | | | | | | | | | | |
| Time | 47.64 | **12.40** | 64.72 | **4.29** | 69.12 | **10.93** | 62.90 | **13.81** | 62.20 | **6.44** | 173.07 | **17.08** | 167.54 | **55.09** | 269.42 | **34.56** | 135.14 | **85.04** | 104.03 | **8.61** | 133.07 | **40.71** |
| CPU | 69.67 | **2.73** | 37.17 | **16.37** | 41.03 | **11.80** | 38.13 | **10.13** | 45.87 | **0.94** | 30.93 | **11.18** | 29.63 | **3.11** | 36.27 | **6.02** | 27.20 | **12.44** | 33.30 | **5.19** | 26.07 | **17.27** |
| Memory (MB) | 457.13 | **33.30** | 631.57 | **3.19** | 761.13 | **249.26** | 760.87 | **110.73** | 797.63 | **38.83** | 1623.07 | **454.34** | 1552.73 | **7.50** | 1753.57 | **309.09** | 2466.47 | **390.76** | 2659.37 | **209.67** | 2533.60 | **812.82** |
| **Join** | | | | | | | | | | | | | | | | | | | | | | |
| Time | 108.97 | 25.89 | 123.22 | 4.38 | 160.40 | 14.21 | 117.85 | 21.37 | 111.69 | 9.53 | 288.49 | 38.53 | 167.75 | 8.13 | 383.02 | 30.65 | 200.23 | 38.76 | 213.25 | 10.19 | 217.93 | 31.11 |
| CPU | 44.20 | 2.59 | 28.70 | 1.72 | 32.47 | 5.72 | 28.03 | 1.80 | 32.37 | 2.74 | 26.00 | 4.24 | 21.80 | 3.76 | 30.33 | 7.35 | 18.33 | 1.52 | 18.87 | 1.37 | 20.67 | 3.49 |
| Memocy (MB) | 352.03 | 6.34 | 613.20 | 10.97 | 615.73 | 64.34 | 599.00 | 56.28 | 616.27 | 11.14 | 1289 | 162.99 | 1115.30 | 8.41 | 1510.03 | 166.02 | 2393.27 | 74.33 | 2357.73 | 122.34 | 2454.17 | 61.67 |
| **Scan** | | | | | | | | | | | | | | | | | | | | | | |
| Time | 37.06 | 3.01 | 33.03 | 1.06 | 94.95 | 15.89 | 53.89 | 6.62 | 38.77 | 0.91 | 117.22 | 32.34 | 81.25 | 13.52 | 143.82 | 22.66 | 99.77 | 14.34 | 106.57 | 31.39 | 123.03 | 66.02 |
| CPU | 34.97 | 3.28 | 16.30 | 0.43 | 31.30 | 2.62 | 25.00 | 3.82 | 26.67 | 1.03 | 29.43 | 3.43 | 36.97 | 1.46 | 34.8 | 5.68 | 19.63 | 0.87 | 18.67 | 1.93 | 18.20 | 6.76 |
| Memocy (MB) | 298.73 | 4.83 | 432.53 | 6.25 | 623.40 | 5.39 | 592.13 | 18.42 | 574.83 | 5.96 | 1258.20 | 70.75 | 1144.07 | 38.60 | 1322.47 | 243.37 | 2293.23 | 110.99 | 2256.8 | 42.53 | 2311.87 | 124.70 |
| **Pagerank** | | | | | | | | | | | | | | | | | | | | | | |
| Time | 148.55 | 9.55 | 214.96 | 36.21 | 153.65 | 33.36 | 194.66 | 44.4 | 177.75 | 9.23 | 571.13 | 19.67 | 427.82 | 31.55 | 781.25 | 41.6 | 567.69 | 37.02 | 473.73 | 11.02 | 496.78 | 20.52 |
| CPU | 72.80 | 10.24 | 44.80 | 13.17 | 59.37 | 30.66 | 46.37 | 6.85 | 60.27 | 5.74 | 38.70 | 8.35 | 51.17 | 6.39 | 33.17 | 8.68 | 24.70 | 1.72 | 26.40 | 10.25 | 31.83 | 12.51 |
| Memocy (MB) | 424.97 | 24.46 | 742.13 | 8.32 | 784.13 | 97.23 | 731.07 | 76.85 | 756.87 | 88.44 | 1693.47 | 261.40 | 1785.07 | 333.00 | 1841.30 | 46.89 | 2681.37 | 390.61 | 2595.93 | 415.55 | 2677.80 | 1044.33 |

Table 5: Mapreduce Workload Traffic Flow Data Summary

| Configurations | | A | | B-1 | | B-2 | | B-3 | | B-4 | | C-1 | | C-2 | | D-1 | | E-1 | | E-2 | | E-3 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Traffic Catagories** | **Port** | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI | Avg | 95%CI |
| **Sort** | | | | | | | | | | | | | | | | | | | | | | | |
| **Slaves To Master** | 54310 | 0.06 | 0.051 | 0.098 | 0.087 | 0.074 | 0.024 | 0.097 | 0.151 | 0.091 | 0.092 | 0.154 | 0.056 | 0.251 | 0.137 | 0.236 | 0.525 | 0.288 | 0.418 | 0.216 | 0.062 | 0.358 | 0.197 |
| | 8031 | 0.037 | 0.043 | 0.093 | 0.121 | 0.049 | 0.011 | 0.071 | 0.183 | 0.053 | 0.103 | 0.154 | 0.105 | 0.317 | 0.246 | 0.304 | 0.667 | 0.341 | 0.742 | 0.239 | 0.176 | 0.416 | 0.15 |
| | 8030 | 0.01 | 0.006 | 0.016 | 0.012 | 0.012 | 0.002 | 0.014 | 0.012 | 0.015 | 0.009 | 0.025 | 0.038 | 0.033 | 0.008 | 0.043 | 0.093 | 0.036 | 0.021 | 0.037 | 0.018 | 0.044 | 0.028 |
| **Slaves To Slaves** | 50010 | 0 | 0 | 50.634 | 0.242 | 50.642 | 0.135 | 50.578 | 0.666 | 51.23 | 0.639 | 253.237 | 3.877 | 256.354 | 4.173 | 355.036 | 10.221 | 314.606 | 7.945 | 318.393 | 7.668 | 319.712 | 14.65 |
| **Others** | 22 | 0.006 | 0.003 | 0.011 | 0.003 | 0 | 0 | 0.003 | 0.014 | 0.004 | 0.017 | 0.011 | 0.049 | 0 | 0 | 0 | 0 | 0.027 | 0.117 | 0 | 0 | 0 | 0 |
| | 8032 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **WordCount** | | | | | | | | | | | | | | | | | | | | | | | |
| **Slaves To Master** | 54310 | 0.060 | 0.054 | 0.092 | 0.099 | 0.096 | 0.046 | 0.082 | 0.119 | 0.079 | 0.018 | 0.161 | 0.042 | 0.191 | 0.141 | 0.298 | 0.104 | 0.266 | 0.222 | 0.245 | 0.236 | 0.311 | 0.182 |
| | 8031 | 0.036 | 0.049 | 0.072 | 0.074 | 0.064 | 0.037 | 0.053 | 0.002 | 0.05 | 0.046 | 0.155 | 0.164 | 0.173 | 0.132 | 0.359 | 0.11 | 0.248 | 0.253 | 0.255 | 0.441 | 0.278 | 0.329 |
| | 8030 | 0.011 | 0.007 | 0.016 | 0.011 | 0.014 | 0.001 | 0.015 | 0.009 | 0.013 | 0.001 | 0.03 | 0.01 | 0.032 | 0.019 | 0.054 | 0.019 | 0.044 | 0.014 | 0.042 | 0.036 | 0.037 | 0.003 |
| **Slaves To Slaves** | 50010 | 0 | 0 | 0.913 | 0.035 | 0.686 | 0.038 | 0.665 | 0.001 | 0.68 | 0.002 | 1.561 | 0.4 | 1.363 | 0.351 | 2.373 | 0.522 | 3.063 | 0.847 | 3.488 | 0.511 | 3.706 | 0.885 |
| **Others** | 22 | 0.005 | 0.002 | 0.002 | 0.008 | 0.016 | 0.018 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 8032 | 0.019 | 0.083 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Join** | | | | | | | | | | | | | | | | | | | | | | | |
| **Slaves To Master** | 54310 | 0.137 | 0.138 | 0.179 | 0.261 | 0.150 | 0.059 | 0.189 | 0.230 | 0.137 | 0.014 | 0.401 | 0.315 | 0.297 | 0.141 | 0.381 | 0.195 | 0.461 | 0.737 | 0.478 | 0.292 | 0.442 | 0.348 |
| | 8031 | 0.081 | 0.066 | 0.113 | 0.19 | 0.129 | 0.031 | 0.148 | 0.125 | 0.119 | 0.035 | 0.395 | 0.069 | 0.271 | 0.219 | 0.461 | 0.02 | 0.517 | 1.326 | 0.532 | 0.567 | 0.646 | 0.387 |
| | 8030 | 0.024 | 0.026 | 0.032 | 0.033 | 0.022 | 0.001 | 0.03 | 0.029 | 0.022 | 0 | 0.058 | 0.029 | 0.06 | 0.026 | 0.066 | 0.02 | 0.061 | 0.036 | 0.049 | 0.017 | 0.056 | 0.03 |
| **Slaves To Slaves** | 50010 | 0 | 0 | 35.886 | 2.624 | 34.774 | 1.765 | 36.278 | 3.007 | 35.182 | 1.745 | 85.577 | 7.607 | 105.285 | 1.128 | 101.337 | 19.675 | 242.593 | 5.054 | 242.128 | 8.268 | 238.858 | 20.447 |
| **Others** | 22 | 0.003 | 0.014 | 0.006 | 0.026 | 0 | 0 | 0 | 0 | 0 | 0 | 0.012 | 0.05 | 0 | 0 | 0 | 0 | 0.028 | 0.12 | 0 | 0 | 0 | 0 |
| | 8032 | 0 | 0 | 0 | 0 | 0 | 0 | 0.006 | 0.027 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Scan** | | | | | | | | | | | | | | | | | | | | | | | |
| **Slaves To Master** | 54310 | 0.077 | 0.151 | 0.151 | 0.248 | 0.062 | 0.006 | 0.125 | 0.241 | 0.066 | 0.011 | 0.311 | 0.410 | 0.200 | 0.097 | 0.363 | 0.458 | 0.353 | 0.228 | 0.474 | 0.430 | 0.269 | 0.126 |
| | 8031 | 0.052 | 0.102 | 0.12 | 0.193 | 0.049 | 0.05 | 0.112 | 0.194 | 0.076 | 0.013 | 0.279 | 0.324 | 0.197 | 0.086 | 0.32 | 0.303 | 0.281 | 0.266 | 0.385 | 0.545 | 0.236 | 0.077 |
| | 8030 | 0.012 | 0.025 | 0.019 | 0.03 | 0.008 | 0.001 | 0.018 | 0.03 | 0.008 | 0.001 | 0.035 | 0.048 | 0.02 | 0.003 | 0.043 | 0.039 | 0.041 | 0.038 | 0.039 | 0.038 | 0.027 | 0.003 |
| | 50010 | 0 | 0 | 59.08 | 1.69 | 59.448 | 0.277 | 59.768 | 0.032 | 59.768 | 0.108 | 289.482 | 1.568 | 298.707 | 26.64 | 439.974 | 24.981 | 439.016 | 35.091 | 465.416 | 28.026 | 451.201 | 17.702 |
| **Others** | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 103.318 | 444.542 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 8032 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Pagerank** | | | | | | | | | | | | | | | | | | | | | | | |
| **Slaves To Master** | 54310 | 0.138 | 0.132 | 0.216 | 0.204 | 0.186 | 0.088 | 0.232 | 0.133 | 0.238 | 0.291 | 0.587 | 0.440 | 0.471 | 0.081 | 0.841 | 0.214 | 0.662 | 0.811 | 0.683 | 0.161 | 0.748 | 0.197 |
| | 8031 | 0.09 | 0.059 | 0.198 | 0.182 | 0.169 | 0.12 | 0.199 | 0.076 | 0.202 | 0.188 | 0.522 | 0.061 | 0.533 | 0.157 | 0.817 | 0.132 | 0.615 | 0.955 | 0.81 | 0.2 | 0.926 | 0.172 |
| | 8030 | 0.021 | 0.045 | 0.056 | 0.021 | 0.049 | 0.011 | 0.057 | 0.028 | 0.063 | 0.07 | 0.121 | 0.023 | 0.1 | 0.016 | 0.195 | 0.07 | 0.134 | 0.048 | 0.123 | 0.025 | 0.114 | 0.069 |
| | 50010 | 0 | 0 | 127.878 | 4.058 | 129.949 | 5.251 | 130.879 | 2.618 | 127.304 | 6.722 | 579.303 | 18.069 | 571.967 | 6.101 | 757.69 | 9.565 | 718.863 | 16.586 | 732.344 | 39.643 | 738.479 | 5.218 |
| **Others** | 22 | 0.003 | 0.014 | 0.006 | 0.027 | 0 | 0 | 0.005 | 0.022 | 0 | 0 | 0.012 | 0.053 | 0 | 0 | 0 | 0 | 0.063 | 0.271 | 0 | 0 | 0 | 0 |
| | 8032 | 0.015 | 0.065 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 6: Example of MapReducde Shuffle Traffic For Each Flow

| Workloads | Configuration | Number of Shuffle Pairs | Total Data Size (MB) | % Of Flow#1 | % Of Flow#2 | % Of Flow#3 | % Of Flow#4 | % Of Flow#5 | % Of Flow#6 | % Of Flow#7 | % Of Flow#8 | % Of Flow#9 | Others |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sort | **A-1** | 0 | - | - | - | - | - | - | - | - | - | - | - |
| | **B-1** | 1 | 50.54 | 98.75% | 1.25% | - | - | - | - | - | - | - | - |
| | **B-2** | 2 | 50.63 | 50.50% | 49.50% | - | - | - | - | - | - | - | - |
| | **B-3** | 2 | 50.70 | 99.48% | 0.52% | - | - | - | - | - | - | - | - |
| | **B-4** | 2 | 50.99 | 100.00% | 0.00% | - | - | - | - | - | - | - | - |
| | **C-1** | 9 | 252.20 | 24.92% | 24.92% | 13.06% | 12.98% | 12.01% | 11.93% | 0.10% | 0.08% | 0.00% | 0.00% |
| | **C-2** | 12 | 257.53 | 24.88% | 12.93% | 12.51% | 12.45% | 12.34% | 12.32% | 12.31% | 0.10% | 0.10% | 0.06% |
| | **D-1** | 12 | 353.21 | 29.87% | 20.69% | 13.24% | 11.36% | 9.38% | 9.37% | 4.04% | 1.97% | 0.08% | 0.00% |
| | **E-1** | 33 | 302.05 | 10.85% | 10.82% | 10.82% | 10.81% | 10.77% | 10.75% | 9.09% | 6.40% | 4.82% | 14.87% |
| | **E-2** | 28 | 321.83 | 14.63% | 14.43% | 10.29% | 10.28% | 10.19% | 10.18% | 10.18% | 10.17% | 4.56% | 5.08% |
| | **E-3** | 30 | 324.83 | 20.27% | 12.49% | 12.29% | 10.31% | 10.24% | 10.01% | 9.98% | 4.61% | 2.28% | 7.52% |
| | **Traffic Node Pairs For E-3** | | | (S3→S8) | (S7→S2) | (S8→S4) | (S7→S1) | (S8→S5) | (S6→S8) | (S5→S6) | (S3→S8) | (S6→S7) | - |
| | | | | | | | | | | | | | |
| Wordcount | **A-1** | 0 | - | - | - | - | - | - | - | - | - | - | - |
| | **B-1** | 2 | 0.93 | 68.60% | 31.40% | - | - | - | - | - | - | - | - |
| | **B-2** | 2 | 0.70 | 86.05% | 13.95% | - | - | - | - | - | - | - | - |
| | **B-3** | 2 | 0.66 | 56.84% | 43.16% | - | - | - | - | - | - | - | - |
| | **B-4** | 2 | 0.68 | 59.47% | 40.53% | - | - | - | - | - | - | - | - |
| | **C-1** | 10 | 1.74 | 35.21% | 21.13% | 18.09% | 15.61% | 7.63% | 1.01% | 0.43% | 0.35% | 0.10% | 0.43% |
| | **C-2** | 10 | 1.25 | 27.30% | 27.06% | 25.48% | 11.61% | 7.04% | 0.92% | 0.28% | 0.16% | 0.13% | 0.04% |
| | **D-1** | 11 | 2.15 | 21.57% | 19.03% | 14.41% | 12.35% | 12.22% | 7.29% | 4.59% | 4.37% | 3.84% | 0.34% |
| | **E-1** | 31 | 2.70 | 15.98% | 9.92% | 9.83% | 9.76% | 9.72% | 9.53% | 6.08% | 5.88% | 4.53% | 18.78% |
| | **E-2** | 27 | 3.58 | 36.31% | 10.48% | 9.41% | 7.65% | 7.56% | 7.37% | 5.20% | 4.27% | 3.76% | 7.99% |
| | **E-3** | 33 | 3.88 | 20.95% | 14.40% | 13.33% | 9.27% | 7.19% | 6.86% | 6.81% | 4.10% | 4.06% | 13.02% |
| | **Traffic Node Pairs For E-3** | | | (S5→S3) | (S5→S7) | (S5→S4) | (S3→S7) | (S6→S3) | (S5→S3) | (S7→S8) | (S4→S7) | (S8→S4) | - |
| | | | | | | | | | | | | | |
| Join | **A-1** | 0 | - | - | - | - | - | - | - | - | - | - | - |
| | **B-1** | 2 | 35.28 | 98.88% | 1.12% | - | - | - | - | - | - | - | - |
| | **B-2** | 2 | 35.53 | 95.84% | 4.16% | - | - | - | - | - | - | - | - |
| | **B-3** | 2 | 35.60 | 98.09% | 1.91% | - | - | - | - | - | - | - | - |
| | **B-4** | 2 | 35.57 | 97.68% | 2.32% | - | - | - | - | - | - | - | - |
| | **C-1** | 12 | 87.79 | 21.11% | 19.67% | 19.58% | 19.52% | 19.21% | 0.34% | 0.25% | 0.22% | 0.07% | 0.04% |
| | **C-2** | 12 | 105.27 | 32.20% | 16.41% | 16.38% | 16.29% | 16.03% | 1.56% | 0.40% | 0.24% | 0.17% | 0.31% |
| | **D-1** | 12 | 105.91 | 32.25% | 16.55% | 16.28% | 16.09% | 15.90% | 1.79% | 0.56% | 0.19% | 0.19% | 0.20% |
| | **E-1** | 39 | 243.42 | 14.07% | 7.58% | 7.06% | 7.04% | 7.03% | 7.02% | 6.99% | 6.98% | 6.97% | 29.26% |
| | **E-2** | 37 | 243.45 | 7.25% | 7.25% | 7.09% | 7.07% | 7.05% | 7.03% | 7.01% | 7.01% | 6.99% | 36.26% |
| | **E-3** | 53 | 243.88 | 14.05% | 7.09% | 7.07% | 7.04% | 7.04% | 7.01% | 7.00% | 6.99% | 6.99% | 29.72% |
| | **Traffic Node Pairs For E-3** | | | (S4→S8) | (S2→S6) | (S7→S1) | (S7→S3) | (S1→S2) | (S5→S2) | (S3→S4) | (S6→S7) | (S2→S5) | - |
| | | | | | | | | | | | | | |
| Scan | **A-1** | 0 | - | - | - | - | - | - | - | - | - | - | - |
| | **B-1** | 2 | 59 | 99.99% | 0.01% | - | - | - | - | - | - | - | - |
| | **B-2** | 2 | 59 | 64.99% | 35.01% | - | - | - | - | - | - | - | - |
| | **B-3** | 2 | 59.76 | 100.00% | 0.00% | - | - | - | - | - | - | - | - |
| | **B-4** | 2 | 59.66 | 64.17% | 35.83% | - | - | - | - | - | - | - | - |
| | **C-1** | 11 | 289.02 | 25.42% | 23.45% | 21.67% | 17.85% | 5.90% | 5.50% | 0.14% | 0.06% | 0.00% | 0.00% |
| | **C-2** | 11 | 297.57 | 25.32% | 15.74% | 15.66% | 10.70% | 9.97% | 7.26% | 6.42% | 5.73% | 3.17% | 0.03% |
| | **D-1** | 9 | 451.50 | 29.20% | 17.09% | 15.85% | 12.93% | 12.32% | 7.81% | 4.80% | 0.00% | 0.00% | 0.00% |
| | **E-1** | 39 | 445.73 | 14.65% | 13.32% | 11.14% | 9.75% | 7.86% | 6.77% | 5.45% | 5.29% | 4.27% | 21.50% |
| | **E-2** | 33 | 458.39 | 14.81% | 11.54% | 11.00% | 7.70% | 7.25% | 7.14% | 6.72% | 5.85% | 5.24% | 22.76% |
| | **E-3** | 37 | 447.17 | 17.03% | 11.25% | 7.80% | 7.40% | 6.74% | 5.98% | 5.10% | 4.94% | 4.40% | 29.36% |
| | **Traffic Node Pairs For E-3** | | | (S7→S3) | (S4→S7) | (S5→S8) | (S4→S8) | (S5→S2) | (S4→S1) | (S8→S6) | (S8→S4) | (S7→S2) | - |
| | | | | | | | | | | | | | |
| PageRank | **A-1** | 0 | - | - | - | - | - | - | - | - | - | - | - |
| | **B-1** | 2 | 126.15 | 99.49% | 0.51% | - | - | - | - | - | - | - | - |
| | **B-2** | 2 | 128.15 | 50.41% | 49.59% | - | - | - | - | - | - | - | - |
| | **B-3** | 2 | 130.24 | 97.05% | 2.95% | - | - | - | - | - | - | - | - |
| | **B-4** | 2 | 130.28 | 51.32% | 48.68% | - | - | - | - | - | - | - | - |
| | **C-1** | 12 | 587 | 14.34% | 12.34% | 11.26% | 11.20% | 11.12% | 9.24% | 6.42% | 6.30% | 5.71% | 12.07% |
| | **C-2** | 12 | 574.44 | 18.43% | 18.18% | 13.93% | 12.56% | 11.73% | 6.48% | 6.42% | 6.38% | 5.72% | 0.17% |
| | **D-1** | 12 | 757.74 | 27.41% | 18.65% | 14.42% | 14.31% | 12.97% | 4.36% | 3.13% | 3.03% | 0.60% | 1.13% |
| | **E-1** | 47 | 726.34 | 9.09% | 9.04% | 8.92% | 7.61% | 5.91% | 4.61% | 4.58% | 4.56% | 4.54% | 41.14% |
| | **E-2** | 40 | 738.04 | 13.23% | 8.87% | 8.86% | 8.82% | 8.81% | 5.82% | 4.54% | 4.52% | 4.48% | 32.06% |
| | **E-3** | 39 | 737.60 | 16.28% | 11.78% | 8.91% | 8.87% | 7.66% | 7.37% | 7.35% | 7.34% | 7.34% | 17.11% |
| | **Traffic Node Pairs For E-3** | | | (S8→S2) | (S8→S7) | (S3→S8) | (S6→S7) | (S1→S8) | (S4→S8) | (S6→S3) | (S8→S5) | (S7→S4) | - |

# CHAPTER 5

# STUDY OF USER QOE IMPROVEMENT FOR DYNAMIC ADAPTIVE STREAMING OVER HTTP (MPEG-DASH)

In this chapter, we first show the study of the basic MPEG-DASH QoE metric design and propose the moving averaging algorithm for client and use one of the Andorid-based DASH player, ExoPlayer [20]. The purpose of that is to display how an MPEG-DASH based player performs and how client's side fetching algorithm can affect the user perceived QoEs based on pre-defined QoE Metrics. We then discuss our proposed AAN-SDN platform to demonstrate how proposed AAN-SDN can benefit MPEG-DASH based streaming without modification of application itself.

## 5.1 Composite Video Streaming QoE Metrics

Users' perspectives of a high quality of experience can vary by each individual. However, in a nutshell it can be quantified by a combination of many metrics. For example, rebuffer is the most undesirable case based on [40]. A high bitrate will provide the user a better streaming quality. However, if the bitrate changes frequently from a higher bitrate to a lower one, a sudden bitrate improvement can not represent a smooth experience. However, if the bitrate switches gradually inside one quality category, such as between a standard definition range or a high definition range, it might not cause a noticeable difference for the user. Similar findings are in [17, 35–37, 64].

Our approach toward a composite QoE focuses on both the DASH server and client sides. We consider the situation where a client fetches video segments from a DASH server using the HTTP-GET protocol. The communication channel between client and server uses a configurable network environment. The user's QoE is measured by a set of defined variables; see Table 7.

Table 7: Composite QoE Metrics

| **QoE Metrics Name** | | **Definition** |
|---|---|---|
| **Bitrate Switch Count** | $\rho$, Avg. Change Frequency | $\sum(N_{not}/N)$ |
| | m, Avg. Change Magnitude | $\sum(M_i/N)$, $i = 1, 2, 3..N$ |
| **rebuffer Count** | $T_{fi}$, Count | $T_{fi} \in 0, 1, 2, ...$ |
| | $T_R$, Duration | $T_R = \sum T_{fi}$, $i \in 0, 1, 2, ...$ |
| **Estimated Bitrate** | r, Single Bitrate | $r_i, i \in 0, 1, 2, ...$ |
| | $\gamma$, Avg. Bitrate | $\sum r_i/N$ |
| **Video Quality** | $Q_{sd}$, Avg. SD | $\sum(Q_{si}/N)$ |
| | $Q_{hd}$, Avg. HD | $\sum(Q_{hi}/N)$ |
| | $Q_{total}$, Avg. Total | $Q_{sd} + Q_{hd}$ |
| **Buffer Status** | $T_B$, Buffered Time | $T_q * t_i$,$t_i \in 1, 2, .4..$ |
| | $T_q$, Buffered Queue | $T_q \in 1, 2, 3, ..$ |

*DASH client.* The DASH client is responsible for fetching the proper video bitrate based on current network discrete bandwidth $r$ and captures QoE metrics when streaming a video over the channel. We define $N$ as the total downloaded video segments. The bandwidth moving average $\gamma$ is refreshed for each download. The playback video segment duration $t_i(s) \in 1, 2, 4, ...T$. Each downloaded segment falls into a bitrate quality category that either belongs to $Q_{sd}$ or $Q_{hd}$, where $Q_{sd}$ represents the average of the standard definition video count and $Q_{hd}$ represents the average of the high definition video count. We define $Q_{sd}$ equals $\sum(Q_{si}/N)$, where $Q_{si}$ is the total standard definition video count at download number i ($\in 0, 1, ...N$). $Q_{total}$ is the average of the total video segment quality. The bitrate switchover is captured in two levels: average change frequency $\rho = \sum(N_{not}/N)$, where $N_{not}$ is the number of unchanged bitrates, and the average change magnitude $m = \sum(M_i/N)$, where $M_i$ is the average magnitude after each download. Both $\rho$ and $m$ represent the smoothness of the video playback. The client buffer status is represented by $T_B$, which is the buffered time, and $T_q$, which is the buffered queue size. The total buffered time equals $\sum(T_q * t_i)$, where $t_i$ represents the video segment length.

The rebuffer is measured by the rebuffer frequency $T_{fi}$ and the rebuffer duration $T_R$ for each $T_{fi}$, where $i \in \mathbb{Z}^+$. The system buffered time $T_B$ and queue size $T_q$ are important indicators of rebuffer occurance. Given an available bandwidth, the QoE optimization problem can be expressed as follows:

$$Minimize \begin{cases} BitrateSwitchover : \rho, m \\ rebuffer : T_f, T_R \end{cases}$$

and

$$Maximize \begin{cases} Buffer : T_B, T_q \\ Quality : Q_{total} \end{cases}$$

*Network Profile.* We design various network profiles based on available bandwidth to simulate different network on-off patterns. The bandwidth $r$ is simulated by the increasing and decreasing percentage $P_i$ where $i \in (1, 2, ..., N_p)$, and $N_p$ is the bandwidth change frequency. The bandwidth changing magnitude $P_{diff} = (P_i - P_{i-1})$ represents the stable of the given available bandwidth at a specific time $T$. The combination of $N_p$ and $P_{diff}$ represents a network profile.

## 5.2 Proposed Dynamic Composite QoE Algorithm

Our dynamic composite QoE adaptive algorithm takes a bandwidth based approach. The estimated bandwidth is captured by the weighted sliding window based bandwidth estimator: Sliding Percentile (SP). A overview of how SP runs is elaborated in Algorithm 2. The performance of SP shows a slow convergence when $P_{diff}$ and $N_p$ are relatively large and frequent. The percentile $p$ for each captured bandwidth $r$ and recycle bin size $B$ can be altered to be suited for an unstable network.

Our proposed algorithm is in Algorithm 3. In order to smooth the estimated bandwidth and allow fast convergence in the case of dramatic network changes when using SP as the bandwidth estimator, we add bandwidth history $R_{his}$ to keep track of bandwidth change. $R_{avg}$ is the moving average of the estimated bandwidth. Immediate bandwidth change $\alpha = R_{his}[-2]/R_{his}[-1]$ is utilized in order to enable and accurately detect network changes and avoid false positive bandwidth estimations. Together $\Delta = R_{avg}[-2]/R_{avg}[-1]$ and $\alpha$ will decide how the bandwidth changes as well as the changing magnitude. To mitigate the SP slow convergence problem, compensators: $\omega$ and $\epsilon$ are being added to the SP algorithm.

Buffered time based threshold indicators, $T_{in}$ and $T_{de}$, are also used for returning the final bitrate $R_{next}$ for downloading where $T_{in}$ and $T_{de}$ represent the threshold for

**Algorithm 2:** SlidingPercentile

```
input  : MaxWeight $W$, percentile $p$, SampleSet $Set$, SampleSize $Set_s$, Recycle Bin $B$,
         BinSize $B_s$
output: A weighted Bandwidth $r$
Download ByteSize $D_s$
// Save data into $Set_s$ and Keep it under $W$
for $i \subset D_s$ do
    if $B_s > 0$ then
        $Set$.add$\Big($B[-1]$\Big)$ ;
    else
        $Set$.add$\Big(\sqrt{i}\Big)$ ;
    while $Set_s >= W$ do
        $excessWeight \leftarrow Set_s - W$;
        if $excessWeight >= Set[0]$ then
            $Set_s - = Set[0]$;
            $Set_s$.remove(0);
            for $j \leftarrow B_s$ do
                $B$.add($Set$[0]);
        else
            $W - = excessWeight$
// Return Weighhted Bandwidth $r$
for $i \leftarrow Set$ do
    if $\sum i >= Set_s * p$ then
        return $r_i$;
```

bitrate upgrades and downgrades, respectively. By allowing $R_{next}$ to follow $R_{avg}$ and with constraints of buffer threshold indicators, it can mitigate the rebuffering occurance and improve composite QoE.

**Algorithm 3:** Proposed Dynamic Composite QoE Algorithm

```
input : Estimated Bandwidth $r$, Buffer Time $T_B$, BitRate Increase Threshold $T_{in}$, BitRate
        Decrease Threshold $T_{de}$ Aviable Bitrate $R_{mpd}$, current Bitrate $R_{current}$, percentile
        $p$, Recycle Bin $B$, BinSize $B_s$, Bandwidth History $R_{his}$, Immediate Bandwidth
        Change $\alpha = R_{his}[-2]/R_{his}[-1]$, Bandwidth Moving Average $R_{avg}$,
        BandwidthChangePencentage $\Delta = R_{avg}[-2]/R_{avg}[-1]$, BandwidthState $R_{state}$,
        Bandwidth factor $\zeta$, Percentile factor $\omega$, Bin factor $\epsilon$
output: Next Bitrate $R_{next}$

System Initialization;
for each $r$ evaltion cycle do
    Recalculate $R_{avg}, R_{his}, \alpha, \Delta$;
    // Calculate $R_{next}$
    ;
    if $\Delta > 1$ and $\alpha > 1 \pm \zeta$ then
        $R_{state}$ is in decreasing mode;
        $p = 0.1 \pm \omega$, $B_s = 2 \pm \epsilon$;
        for $i \leftarrow R_{mpd}$ do
            if $R_{current} <= i$ then
                Return $R_{next} = i \pm 1 \sim 2$;
    else if $\left(\Delta > 1 \text{ and } \alpha < 1 \pm \zeta\right)$ or $\left(\Delta < 1 \text{ and } \alpha < 1 \pm \zeta\right)$ then
        $R_{state}$ is in increasing mode;
        $p = 0.5 \pm \omega$, $B_s = 5 \pm \epsilon$;
        Return $R_{next} <= R_{avg}[-1]$;
    // Double Check if $R_{next} = i$ is The proper One Based on $T_B$
    if $R_{next} > R_{current}$ then
        if $T_B >= T_{in}$ then
            Return $R_{next}$
        else
            Return $R_{current}$
    else if $R_{next} < R_{current}$ then
        if $T_B >= T_{de}$ then
            Return $R_{next}$
        else
            Return $R_{current}$
    else
        return $R_{next}$
```

## 5.3 Implementation and Empirical Evaluation

In this section, we conduct our empirical network traces using our dynamic algorithm to evaluate the QoE metrics. Google ExoPlayer [20], as the first Android-based mobile DASH player, is being used as our reference player. We compare our algorithm performance with ExoPlayer's reference bitrate adaptive algorithm.

*TestBed Setup.* We run our network traces in a controlled network environment. Video sources are stored in an Apache Server running Ubuntu 14.04 LTS. A network shaper is also deployed at the server side to simulate different network profiles. Fig 33 shows the reference network profile recommended by Chrome's [21] web browser. The ExoPlayer will download video segments from the server while the network shaper tries to control the server side bandwidth throughput.

| **Network Profile** | **Delay (ms)** | **Download (mbps)** | **Upload (kbps)** |
|---|---|---|---|
| **2G** | 150 | 0.45 | 0.15 |
| **3G Regular (3GR)** | 100 | 0.75 | 0.25 |
| **3G Good (3GG)** | 40 | 1.5 | 0.75 |
| **Wi-Fi BenchMark (BM)** | 2 | 5 | 5 |

Figure 33: Network Profile for Testing Environment [21]

The video source used in our trace is from ”Big Buck Bunny” [32] and has 20 video representations, see Table 8. The duration of each video segment is 4s. The quality of each downloaded video segment is grouped into standard ($Q_{sd}$) and high ($Q_{hd}$)

definitions based on the segment bitrate and resolution. In our definition, $Q_{sd}$ includes a segment that has a bitrate less than $0.783mbps$ and the resolution is less than $1280 * 720(720p)$. $Q_{hd}$ includes a segment that has a bitrate greater than and equal to $0.783mbps$ and a resolution that is greater than and equal to $720p$. We argue that high $Q_{hd}$ represents one important factor of a user's QoE.

Table 8: Test Video Bitrate Index

| **Index** | **Bitrate (mbps)** | **Resolution** | **Index** | **Bitrate (mbps)** | **Resolution** |
|---|---|---|---|---|---|
| 0.1 | 0.045 | 320x240 | 1.1 | 0.783 | 1280x720 |
| 0.2 | 0.089 | 320x240 | 1.2 | 1.0 | 1280x720 |
| 0.3 | 0.129 | 320x240 | 1.3 | 1.2 | 1280x720 |
| 0.4 | 0.177 | 480x360 | 1.4 | 1.5 | 1280x720 |
| 0.5 | 0.218 | 480x360 | 1.5 | 2.1 | 1920x1080 |
| 0.6 | 0.256 | 480x360 | 1.6 | 2.4 | 1920x1080 |
| 0.7 | 0.323 | 480x360 | 1.7 | 2.9 | 1920x1080 |
| 0.8 | 0.378 | 480x360 | 1.8 | 3.3 | 1920x1080 |
| 0.9 | 0.509 | 854x480 | 1.9 | 3.6 | 1920x1080 |
| 1.0 | 0.578 | 854x480 | 2.0 | 3.9 | 1920x1080 |

*Understand QoE Metrics Collection with an Example.*

QoE metrics collection while playback is presented here by running a network trace example. We use a sample video source that is 150s long. Since we desire to test our dynamic algorithm within a relatively unstable network condition, we simulated the bandwidth in a steep on-off pattern. Fig. 34 shows that the available bandwidth starts with 5mbps for 10s, then drops to 2G for 35s, and continues a similar pattern until the playback stops. By keeping a low available bandwidth for a relative longer time, we try to create rebuffer cases.

Fig. 35 (a) shows the system buffer detail. Both the buffered time and queue size

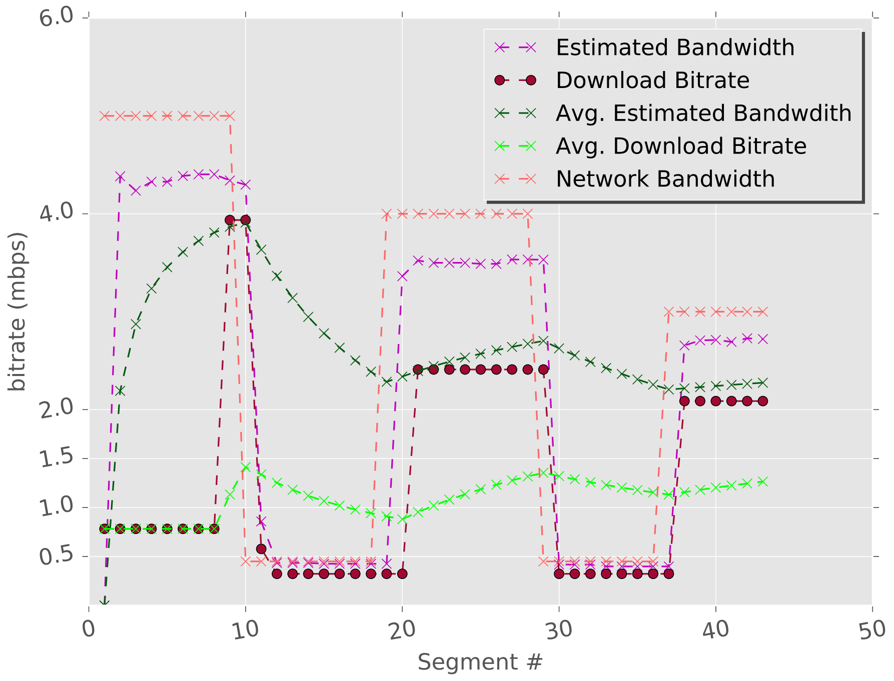


Figure 34: A Sample Run of Network Traces Using Our Dynamic Algorithm

show the state of the client. If either the buffered time or queue size drops to and near 0, the player stops playing and rebuffering happens. For each downloaded segment, the video quality is stored in a buffered queue in bitrate. Fig. 35 (b) captures the bitrate as $Q_{sd}$ and $Q_{hd}$. In this run, $T_{in}$ is set to 10s and $T_{de}$ is 25s, which means the next downloadable bitrate will not: (1) increase to a higher bitrate if $T_{in}$ is less than 10s (2) decrease to a lower bitrate if $T_{de}$ is greater than 25s.

*Empirical Result.* We compared our dynamic algorithm with 's reference adaptive algorithm. We implemented our proposed algorithm in . Our approach for achieving a high composite QoE is described in Section 5.1. A benchmark scenario (Fig. 36) is created for the purpose of giving a best case scenario for a playback. In the benchmark use case,

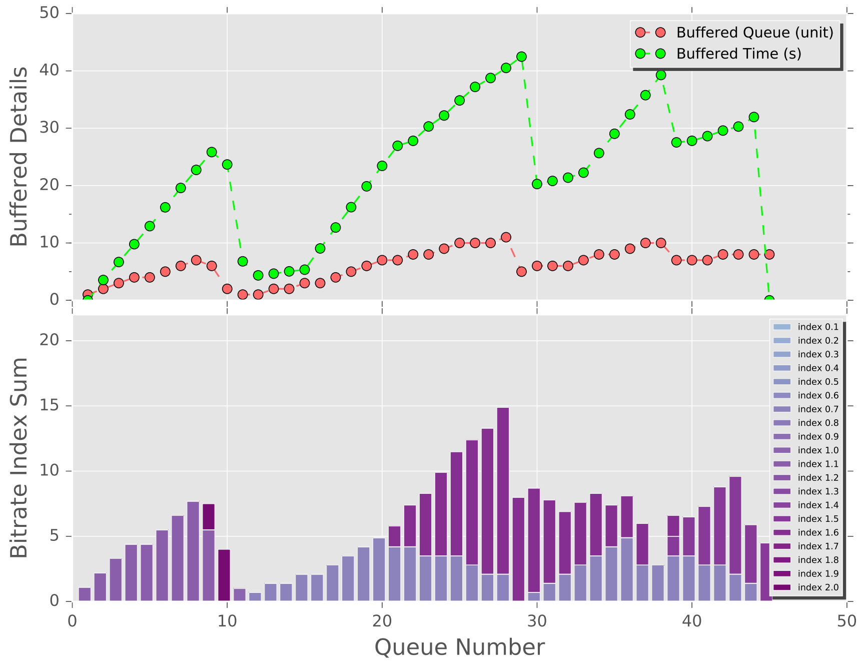


Figure 35: System Buffered Details: (a) Buffered Time and Queue Size (b) Buffered Bitrate Quality and Queue Number

the network shaper simulates a constant 5mbps bandwidth using the same video source in Table 8. The achieved QoE is expected to be higher compared with our simulated network profiles.

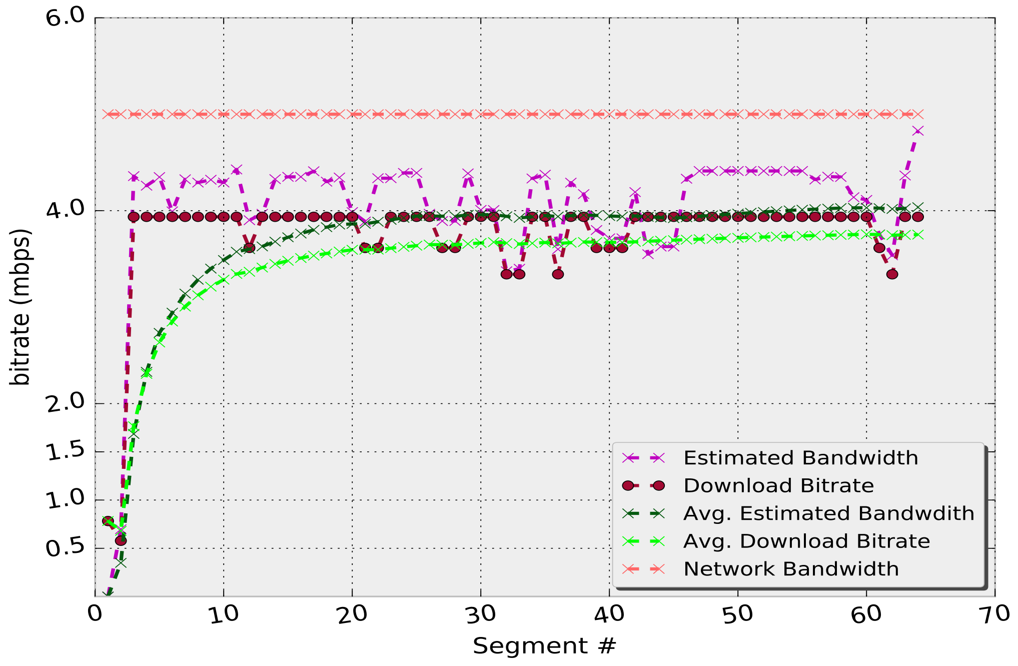
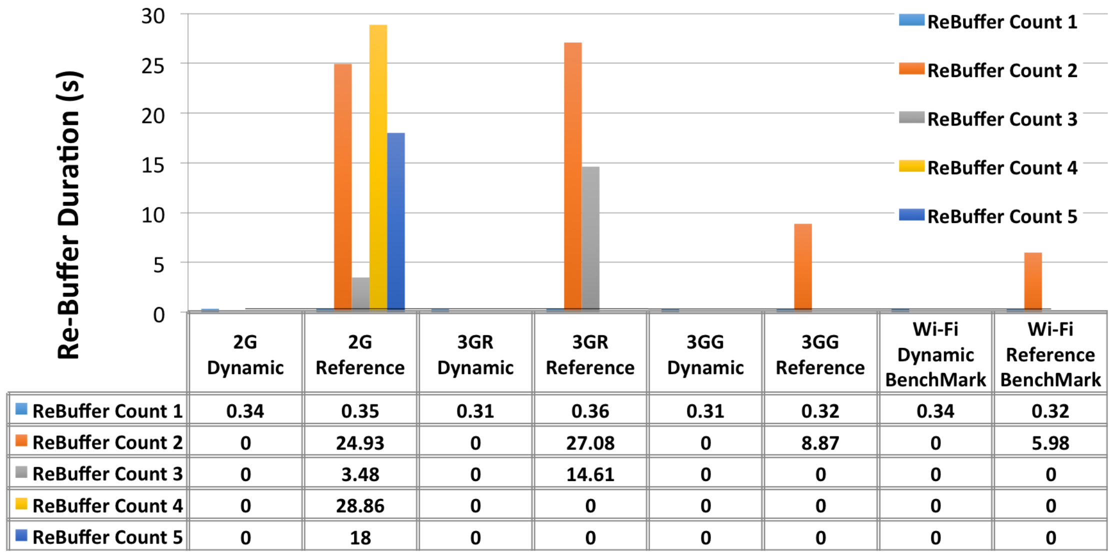


Figure 36: BenchMark Sample Run

After a 150s playback, a rebuffer metric comparison is shown in Fig 37. With our dynamic algorithm, no rebuffering occurs in any network condition except in the initial buffering stage ($\sim 0.35s$) that happens on each case. The worst case happened in a 2G network profile using 's reference adaptive algorithm. Rebuffer $T_f$ occurred 5 times and the total duration was 76s, represented by $T_R = \sum T_{fi}$, where $i \in \{1, 2, 3, 4, 5\}$. In the same network profile, our dynamic algorithm only had an initital buffer. When we improved the network condtion from 2G to 3G, the reference algorithm reduced the $T_f$ and $T_R$, accordingly. However, rebuffering still occurred for each case. Even under the benchmark test, the reference algorithm still remained in two rebuffer cases. That happened because even though the network condition was stable, the reference player greedily downloaded the highest bitrate with no concern for TCP protocol's on-off nature (see in Fig 36). Our dynamic algorithm kept a moving average approach and gradually increased or decreased to the next downloadable bitrate and remained a smooth playback.

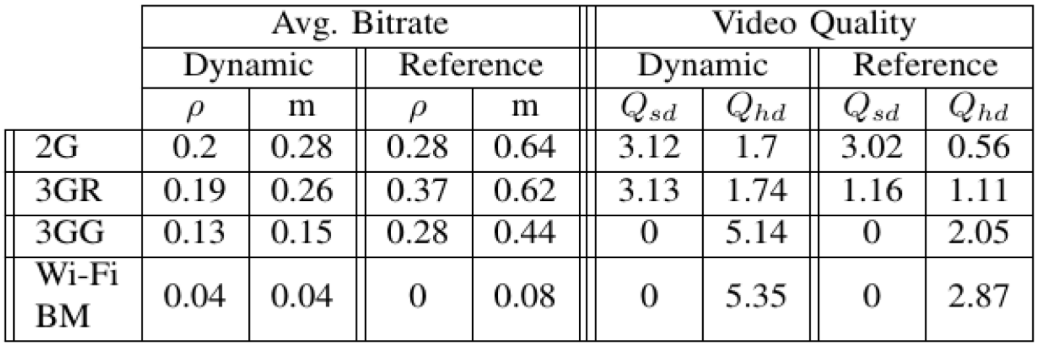


| | 2G Dynamic | 2G Reference | 3GR Dynamic | 3GR Reference | 3GG Dynamic | 3GG Reference | Wi-Fi Dynamic BenchMark | Wi-Fi Reference BenchMark |
|---|---|---|---|---|---|---|---|---|
| ReBuffer Count 1 | 0.34 | 0.35 | 0.31 | 0.36 | 0.31 | 0.32 | 0.34 | 0.32 |
| ReBuffer Count 2 | 0 | 24.93 | 0 | 27.08 | 0 | 8.87 | 0 | 5.98 |
| ReBuffer Count 3 | 0 | 3.48 | 0 | 14.61 | 0 | 0 | 0 | 0 |
| ReBuffer Count 4 | 0 | 28.86 | 0 | 0 | 0 | 0 | 0 | 0 |
| ReBuffer Count 5 | 0 | 18 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 37: Rebuffer Comparison for Various Network Profiles

The bitrate changing metric is shown in Fig 38. The reference algorithm always has a higher bitrate switchover frequence $\rho$ and change magnitude $m$ compared with our dynamic one in the same network profile. For example, in a 2G network profile, dynamic $\rho = 0.2 <$ reference $\rho = 0.28$ and dynamic $m = 0.28 <$ reference $m = 0.64$.

The average downloaded high definition video quality $Q_{hd}$ also keeps a higher number compared with the referenced algorithm. The video quality increases more slowly when the network profile changes from 2G to 3GR because the bandwidth changes (in a relative small range) from 0.45mbps to 0.75mbps. Video quality quickly improves when the network profile changes to 3GG since bandwidth increases to 1.5mbps. But in any case, our dynamic algorithm keeps a higher video quality in terms of the average number of high definition video segments ($Q_{hd}$), lower average bitrate switchover rate ($\rho$) and change magnitude ($m$).



| | Avg. Bitrate | | | | Video Quality | | | |
|---|---|---|---|---|---|---|---|---|
| | Dynamic | | Reference | | Dynamic | | Reference | |
| | $\rho$ | m | $\rho$ | m | $Q_{sd}$ | $Q_{hd}$ | $Q_{sd}$ | $Q_{hd}$ |
| 2G | 0.2 | 0.28 | 0.28 | 0.64 | 3.12 | 1.7 | 3.02 | 0.56 |
| 3GR | 0.19 | 0.26 | 0.37 | 0.62 | 3.13 | 1.74 | 1.16 | 1.11 |
| 3GG | 0.13 | 0.15 | 0.28 | 0.44 | 0 | 5.14 | 0 | 2.05 |
| Wi-Fi BM | 0.04 | 0.04 | 0 | 0.08 | 0 | 5.35 | 0 | 2.87 |

Figure 38: Comparison in Video Bitrate Switchover Frequence, Magnitude and Video Quality Between Proposed Dynamic and ExoPlayer Reference Algorithm

In summar, we proposed a generic dynamic bitrate adaptive algorithm that can be utilized in both bandwidth and buffer based approachs and Investigate the composite QoE approach for improving DASH performance under various network profiles.

# CHAPTER 6

# AAN-SDN FOR MPEG-DASH

In this section, we present our proposed AAN-SDN platform design and implementation for video streaming using MPEG-DASH for both regular 2D and immersive/360-degree videos. We discuss AAN-SDN applicaiton layer design separately based on the video contents.

## 6.1 Regular MPEG-DASH Video Streaming QoE Improvement

In this section, we report on experiments conducted using our proposed AAN-SDN platform to optimize DASH running oversaturated network links for regular flat 2D video contents. Starting from the bottom to the top layer, our proposed architecture (see Fig. 39) segregates our design into three main components as discussed in chapter 3.

### 6.1.1 AAN-SDN Application Layer Design for Regular 2D Video Streaming

In the Application layer, a port number based application recognition feature is also implemented. With regards to DASH streaming applications, web server and DASH client control components are implemented to instruct how to install flows over forwarding devices. The modularization of various components provided by different SDN controllers helps the network administrator to control them individually in a manageable way. Three applications layer components are deployed:

- Iperf is used for background traffic injection purpose.

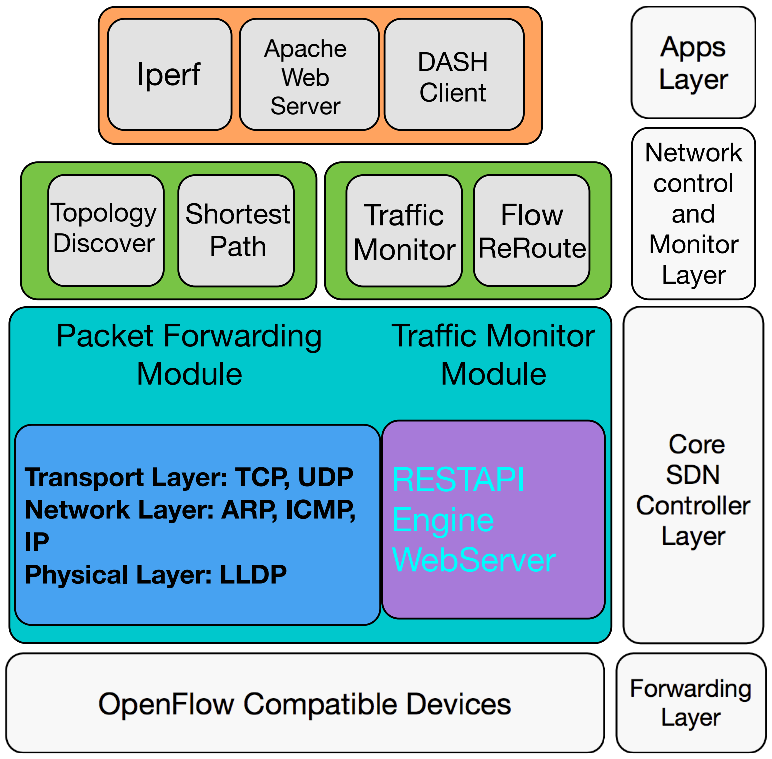


Figure 39: AAN-SDN DASH Experimental Architecture

- One MPEG-DASH compliant Apache web server is deployed for serving DASH video segments.
- One DASH compliant video player is implemented to fetch video segment from server.

### 6.1.2 Performance Evaluation

#### 6.1.2.1 Video Dataset And Media Player

We used MPEG-DASH encoded video datasets to evaluate our models. To test the efficiency of our model, we selected different types of DASH dataset (Table. 10) to ensure each dataset had variations in encoding details.

Table 9: MPEG-DASH Video Dataset Characteristics

| Name | Codec | Source Quality | Genre |
|---|---|---|---|
| BigBuck Bunny | H.264/AVC | 1080p | Animation |
| The Swiss Account | H.264/AVC | 1080p | Sport |
| Valkaama | H.264/AVC | 1080p | Movie |
| Of Forest and Men | H.264/AVC | SD | Movie |

Astream Media player [4] is a python based command line tool. Its an emulated video player which could be used to evaluate performance of the DASH bit-rate adaptation schemes. It can request segments from any multimedia server using the MPD file provided to it during the start of the video streaming session. it typically doesn't provide any GUI for the user to watch the video.

During the video play back, the media player provides logs like Buffer logs and Playback Logs. Buffer logs provide information about Epoch time, current playback time, current buffer size, and current playback state. Playback logs provide information about epoch time, playback time, segment number, segment size, playback bitrate, segment duration, and weighted harmonic mean average download bitrate.

#### 6.1.2.2 Network Topology

We deployed our SDN-based DASH streaming system on the GENI testbed [19] platform. We setup a network topology as shown in Fig. 47. We used Openvswitch (OVS v2.3.1) as our forwarding devices and numbered the DPID in order from '0000000000000001' to '0000000000000006'. One DASH web server and DASH client were deployed. Iperf based deployed nodes had the same hardware configuration with a single core of Intel(R)

Xeon(R) CPU X5650 @ 2.67GHz and 8 GB RAM. Each connected link had default 100 Mbps bandwidth allocation.

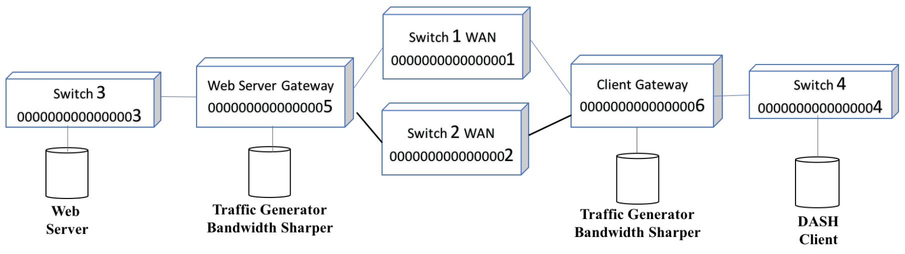


Figure 40: GENI Topology Setup

#### 6.1.2.3 SDN Traffic Reroute Through Adaptive Traffic Engineering

The proposed SDN-based dynamic path selection model is depicted in Fig 41. Assuming this is the first time streaming from the client's side, the completed video segment fetching starts from the ARP request packets, which triggered the shortest path algorithm discussed in Chapter 3 implemented with Dijkstra's algorithm. Once the initial path is selected, MPEG-DASH TCP based flows were installed along the path. Adaptive path selection occurred at time $t2$ when cross traffic was taking most of bandwidth out of the previous DASH streaming link, and the second most shortest path was re-selected. The DASH client and server used the new path to continue the streaming experience.

#### 6.1.2.4 Multiple Session Tests Using Mininet

Mininet is a well-known SDN emulator. A multi-session test of the proposed AAN architecture was conducted using the Mininet. The topology is shown in Fig.47 was setup with 10 Mbps link bandwidth and a 2 ms delay. We used 5 DASH clients. In this test,

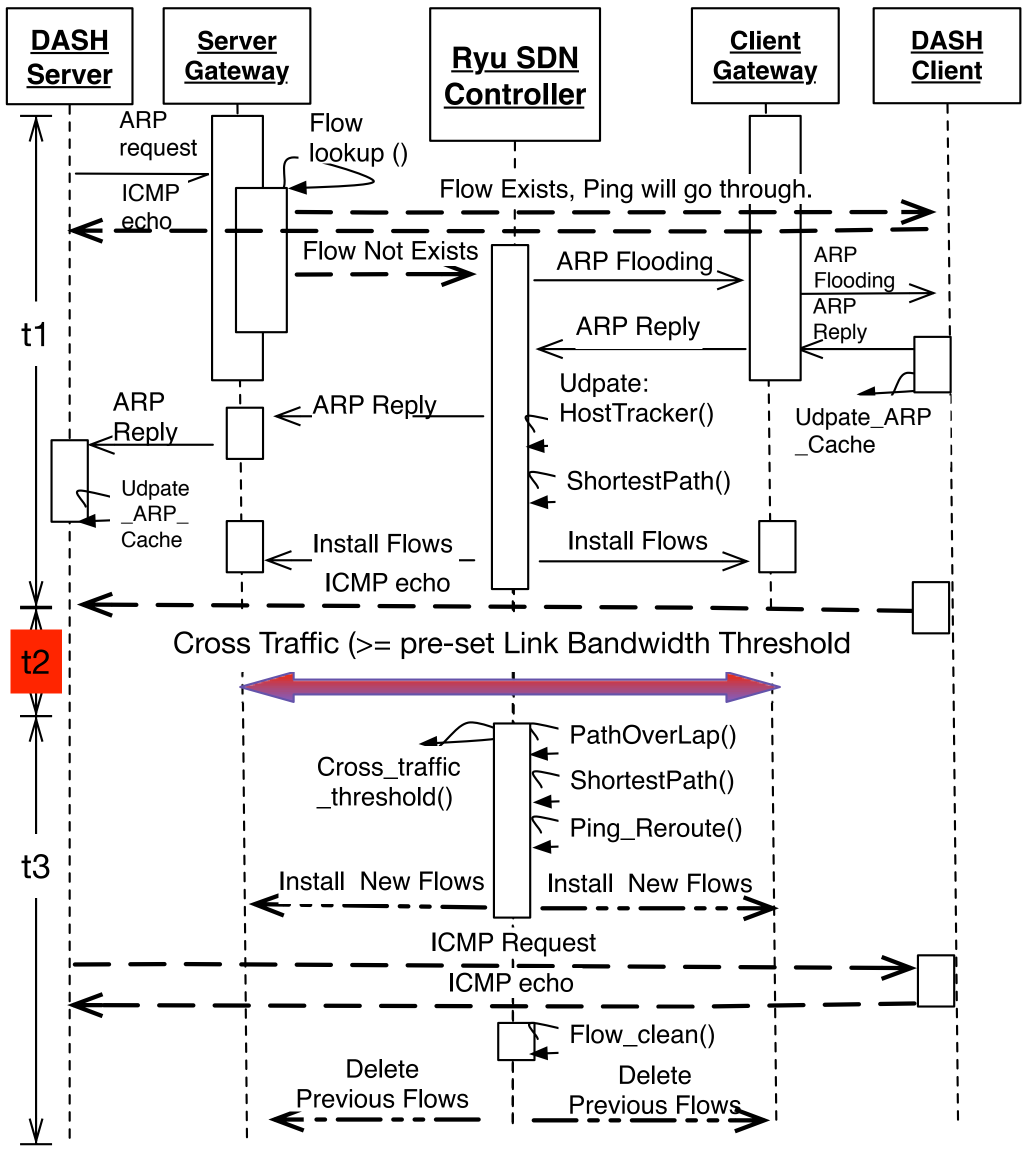


Figure 41: SDN Adaptive Path Selection For DASH Traffic

we started all five clients at the same time to fetch the BigBuckBunny video from a video server.

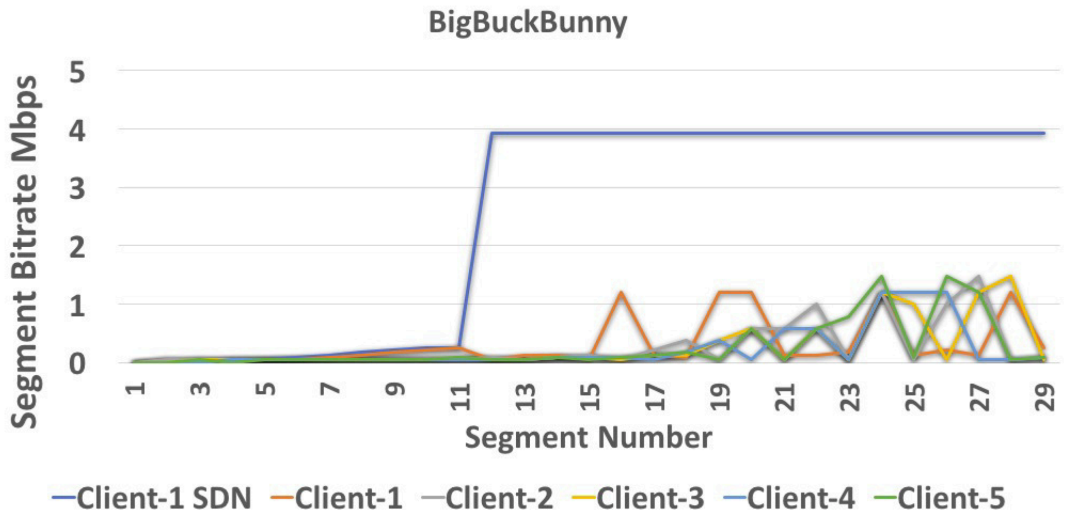


Figure 42: Multiple Clients Fetch Video Segment at The Same Time Using SDN and Non-SDN

Fig. 42 shows the downloaded segment bitrate quality with and without SDN. Without SDN, all of the DASH clients fetched segments using the default path. All of the clients could not get an optimal streaming experience because bandwidth in the congested link was shared by all the clients. With SDN, we rerouted client-1's DASH traffic to the unused second path after the 15th segment to demonstrate how available network link resources were fully exploited using SDN for this DASH client, thereby improving the user's streaming experience significantly. This also shows that through selective traffic engineering, our AAN architecture can be used for giving priority to some customers.

## 6.2 Immersive/360-degree MPEG-DASH Video Streaming QoE Improvement

In this section, we report on experiments conducted using our proposed AAN-SDN platform to optimize DASH running oversaturated network links for Immersive/360-degree video contents. We first discuss state-of-art literature overview regarding how to use tile-based ideas for immersive/360-degree video streamings. Then we discuss the MPEG-DASH Spatial Relationship Description support for immersive videos. We design our experiments based on the movement of user's Region of Interests for a immersive/360-degree video. We then discuss the AAN-SDN application layer design and experiment result for QoE improvement.

### 6.2.1 Tile-Based Concepts for Immersive Video Streaming

Video tilling is considered as a spatial relationship of a video where those tiles are independently decodable video bitstreams [53]. A typical classic video bitstream is considered to be a 1x1 tiling. While the approach of tiling nicely removes all dependencies between tiles, the drawbacks of this approach are that synchronization between tiles must be ensured and that a ROI might require more than one tile to be accessed for reconstructing the view.

Video titles can be grouped based on defined priorities discussed in [42], such as uniform, top-bottom, row-based, center-based and middle-column based. Tiles can be delivered in various strategies discussed in [22] such as full or partial delivery. The chosen delivery strategies shall be based on user's region of interests or the movement of HMD. In [22], the workflow of immersive/360-degree video streaming is illustrated.

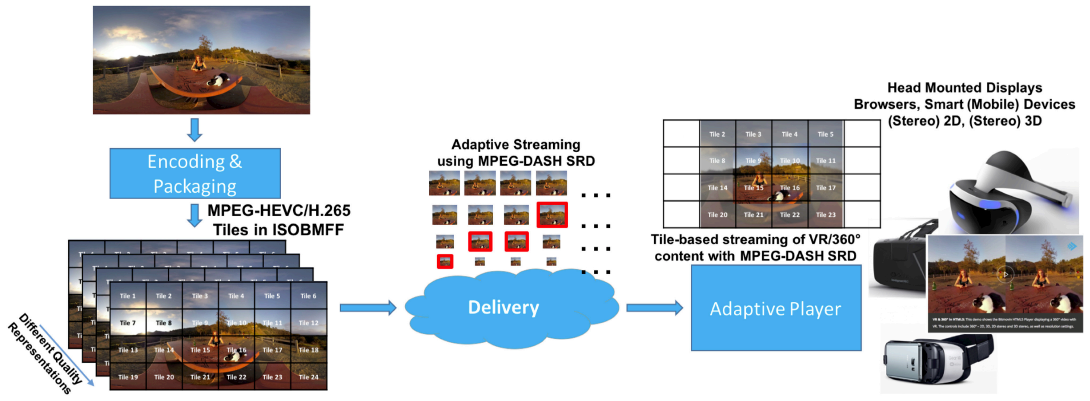
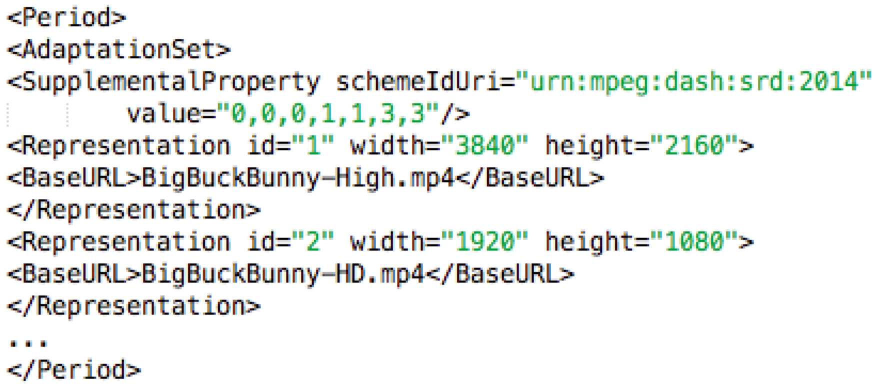

Encoding & Packaging
MPEG-HEVC/H.265 Tiles in ISOBMFF
Different Quality Representations
Adaptive Streaming using MPEG-DASH SRD
Delivery
Tile-based streaming of VR/360° content with MPEG-DASH SRD
Adaptive Player
Head Mounted Displays Browsers, Smart (Mobile) Devices (Stereo) 2D, (Stereo) 3D

Figure 43: Tile-based Streaming Workflow [22]

In Fig 43, the tiling process is performed within a multi-resolution scheme such as using MPEG-DASH , where can contain multiple bitrates for each tile. Using multi-resolution tiling allows for increasing the user perceived qualities by increasing the bitrates of desired tiles and lower the bitrate for others at the same time. For example, when one region of interest (ROI) is defined by user, the video server will provide the highest resolution tiles to constitute the requested region. However, the work of reconstructing one ROI is also more complex and might lead to visual quality drops at the border between high resolution tiles and concealed pixel regions. Tiles synchronization can also decrease user's QoE.

### 6.2.2 MPEG-DASH Spatial Relationship Description (SRD)

DASH is the MPEG standard for describing dynamic adaptive streaming content [31]. The DASH standard allows associating non-timed related information to MPD elements, such as the role of a media asset (e.g. main video or alternate video, subtitle representation, or audio description). The MPD uses so-called Descriptor elements to associate such information. Prior to the definition of SRD, there was no descriptor to associate spatial information with media assets. The existing Viewpoint descriptor was too restricted and mainly oriented toward 3D. It was not possible to describe that two videos were representing spatially related parts of a same scene. The SRD feature solves this problem. Fig 44 depicts one XML example of an MPEG-DASH with SRD support. In this XML, it describes one horizontal 360-degree video made of 9 tiles; each tile has two altenate resolutions. Different resolution can be choosed adaptively by client side segment fetching algorithms.



```
<Period>
<AdaptationSet>
<SupplementalProperty schemeIdUri="urn:mpeg:dash:srd:2014"
        value="0,0,0,1,1,3,3"/>
<Representation id="1" width="3840" height="2160">
<BaseURL>BigBuckBunny-High.mp4</BaseURL>
</Representation>
<Representation id="2" width="1920" height="1080">
<BaseURL>BigBuckBunny-HD.mp4</BaseURL>
</Representation>
...
</Period>
```

Figure 44: DASH SRD XML Example

To make a tiled video, one can resort to either multiple source camera setup, or to partition a single video into multiple frames of smaller resolution. Here, tiles are defined as a spatial segmentation of the video content into a regular grid of independent videos. In addition to supporting segmented video streaming, the DASH standard also allows associating non-timed related information to MPD elements. One of such syntax calls Spatial Relationship Description (SRD) to represent spatial relation for various parts of the same scene. In SRD, the relationship is represented using a scheme URI (@schemeIdUri attribute) and a value (@value attribute). In the value field, one can specify how video tiles are spatially allocated. Figure 45 depicts a tile space for value = $\langle 0, 0, 0, 1, 1, 3, 3 \rangle$. This is a $3 \times 3$ tiled coordination with tile's height and length equaling 1. Starting from the lower left corner, tiles' coordinates are numbered.

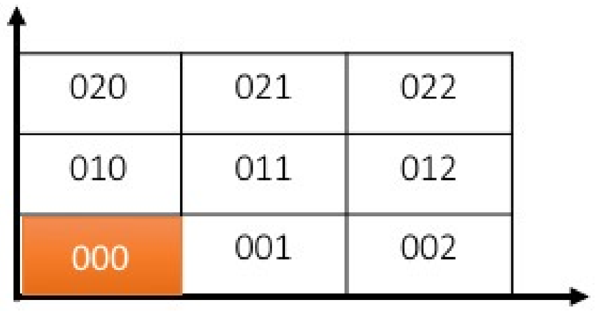


Figure 45: DASH SRD Coordinate Example ($3 \times 3$ tiles)

### 6.2.3 Region of Interests (ROIs) vs Video Tiles

When viewing immersive contents, the ROIs are the current viewport or the viewer's focus. For efficient streaming experience, ideally, one ROI is covered by just one or less than one video tile, which can reduce data size by streaming lower bitrate for non-ROIs tiles and increase, otherwise. However, in the real world use case, a user's ROI falls more like Figure 46, which shows a possible ROI locations for a $3 \times 3$ tiled scheme. In such scenarios, it is better for tiles with the number in [ 010, 020, 011, 021 ] to get higher bitrates to increase the view quality while the rest of the tiles can be transmitted with lower bitrates. Simply by doing such assessments, the transferred data size can be cut by half with increased QoE significantly.

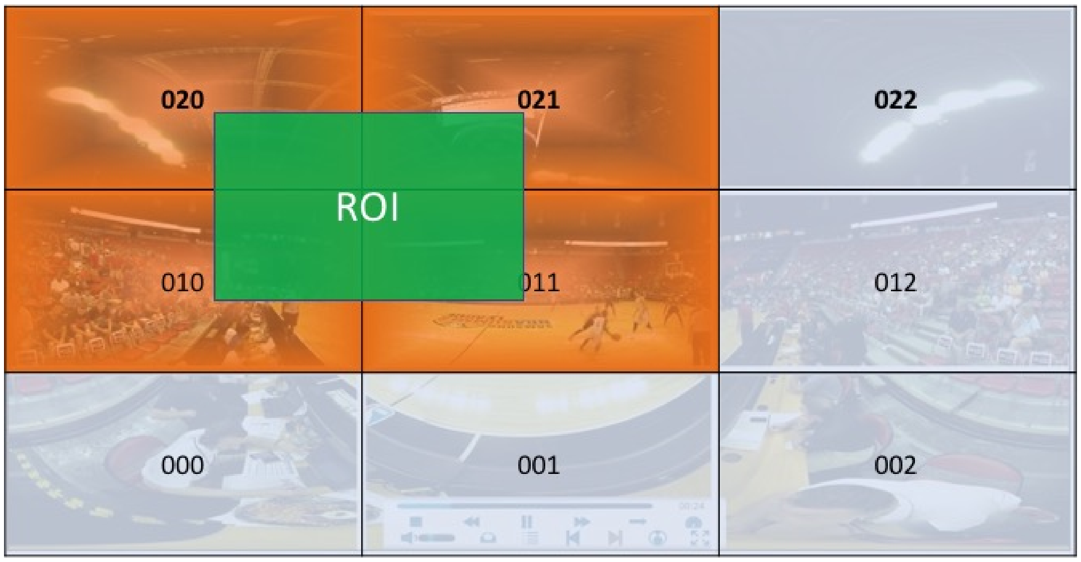


Figure 46: Tile-based Video Playback Example

On the other hand, the current immersive contents are streamed either by downloading the whole content into player's devices or without any optimization with the user's ROIs. Instead, significant bandwidths are wasted while transferring high bitrate segments that is not necessary. Also, the current IP network does not fully support multipath traffic loading balancing and cannot dynamically adapt network flows based on real-time network information, which can lead to suboptimal user's quality of experience and waste of bandwidth. Especially, with the humongous immersive media's creation, new approach are investigated in the next section to explore how SDN can be utilized in streaming immersive media.

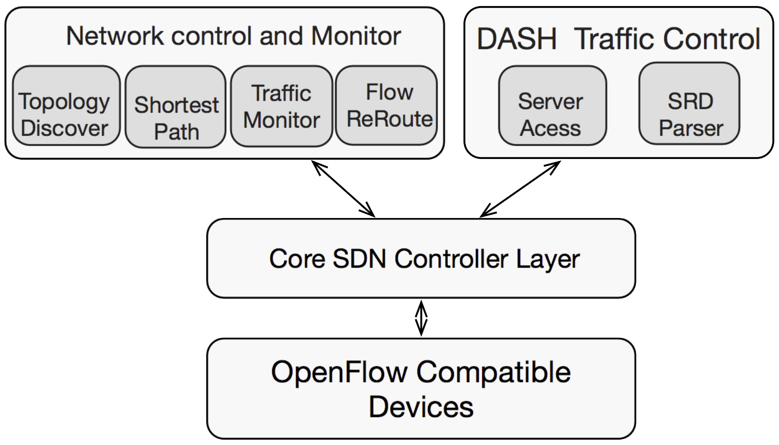


Figure 47: SDN DASH Experimental Architecture

### 6.2.4 AAN-SDN Application Layer Design for Immersive/360-degree Video Streaming

Fig 47 depicts the abstract function design for each SDN layers, which includes the SDN control and Application layer. In the network control and monitoring layer, the global network topology is discovered where we take an adaptive traffic engineering approach by feeding into a shortest path algorithm module to calculate a path for each pair of network node/hosts on an on-demand basis. The traffic monitor component using REST-APIs' services deployed at the core SDN controller layer proactively pulls network traffic information from the network. If there is any pre-defined traffic priority violation, a traffic reroutes using flow reroute component might happen. From the beginning, for ARP messages for a network request such as Ping, SSH, or other applications, it first looks at the flow table and passes the traffic if there is a current matching flow or checks if it is a ARP broadcasting message otherwise.

Port number based application recognition feature is implemented (such as port 80 is by default for the Apache Web Service). In our controlled network, the port number can be managed/changed via a separate configuration file that is read by our SDN controller. With regards to DASH streaming applications, web server and DASH client control components are implemented to instruct how to install flows over forwarding devices. The modularization of various components provided by different SDN controllers helps the network administrator to control them individually in a manageable way.

The SRD parser module can parse the MPD file and extract the tile coordination based on user's ROI change. Shortest path module can calculate in real-time for all shortest paths between each pair of nodes based on existing network conditions and topology changes. In our test scenarios, higher priority tiles are rerouted using flow reroute module to the non-bottle-necked path to achieve better bandwidth utilization.

#### 6.2.5 Network Topology

The topology is as shown in Fig.47 that is setup with 10 Mbps link bandwidth and 2 ms delay in Mininet. We set up a network topology as illustrated in Fig. 48. We use Openvswitch (OVS v2.3.1) as our forwarding devices. One DASH Apache web server and DASH client were deployed. Each connected link has default 10 Mbps bandwidth allocation. For the implementation of SDN, we used Ryu and OpenFlow v1.3 as southbound APIs.

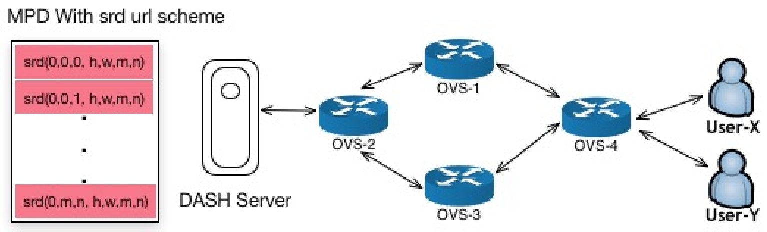


Figure 48: Mininet SDN Topology Setup

### 6.2.6 Immersive Video Dataset and Media Player

We use MPEG-DASH encoded video datasets to evaluate our models. To test the efficiency of our model, we selected different types of DASH datasets (Table. 10) ensuring that each dataset has variations in encoding details.

Table 10: MPEG-DASH Video Dataset Characteristics

| Name | Codec | Source Quality | Genre |
|---|---|---|---|
| BigBuck Bunny | H.264/AVC | 1080p | Animation |

*AStream Media player* used in [35] is a python based command line tool. It is an emulated video player, which could be used to evaluate the performance of the DASH bitrate adaptation schemes. It can request segments from any multimedia server using MPD file provided to it during the start of the video streaming session. It typical does not provide any GUI for the user to watch the video.

During the video playback, the media player provides logs like Buffer logs and Playback Logs. Buffer logs provide information about Epoch time, Current playback time, current buffer size, current playback state. Playback logs provide information about epoch time, playback time, segment number, segment size, playback bitrate, segment duration and weighted harmonic mean average download bitrate.

### 6.2.7 SDN Traffic Engineering for ROI(s) Traffic Optimization

In this section, we evaluate user's QoE based given ROI bitrates and buffer size. Figure 49 shows that two different ROI regions on a given $2X2$ tile-based video segments. The MPEG-DASH SRD represents the spatial locations, such as tile $\#12$ is represented by SRD syntax value field $\langle 1, 2, 1, 1, 2, 2\rangle$. In our test scenario, ROI-A is covered only by tile $\#12$, while ROI-B is covered by both tile $\#11$ and $\#12$. The number of tiles covering the specific ROI(s) is the number of segments that user's focus at a specific time. At the specific time $t$, the user will change its ROI, such as from ROI-A to ROI-A' and from ROI-B to ROI-B'. We track the segment bitrate changes for each tile and evaluate how proposed SDN framework can adaptively optimize user's QoE by varying the segments' bitrate based on ROIs' coordination and movement.

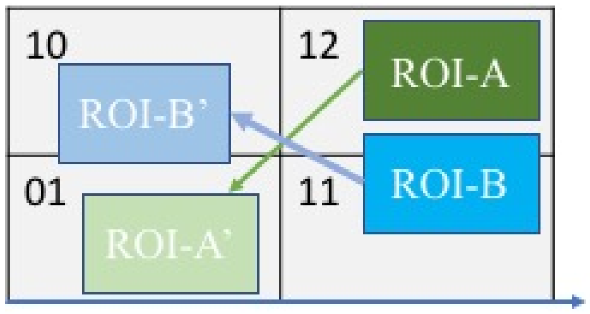
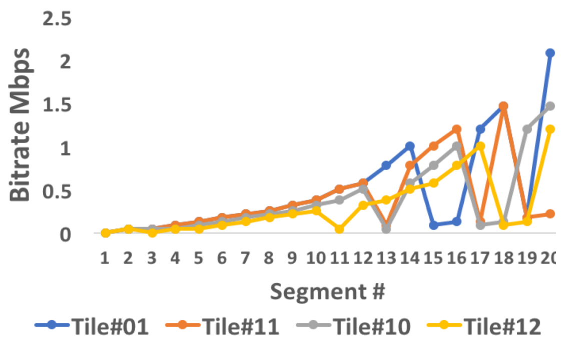


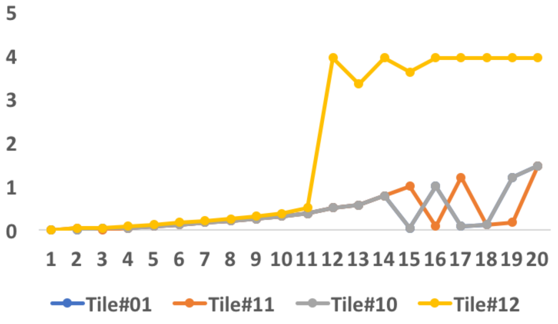
Figure 49: ROI Movement for A $2X2$ Tile-based Video Segments

### 6.2.8 Benchmark with Non-SDN Deployment

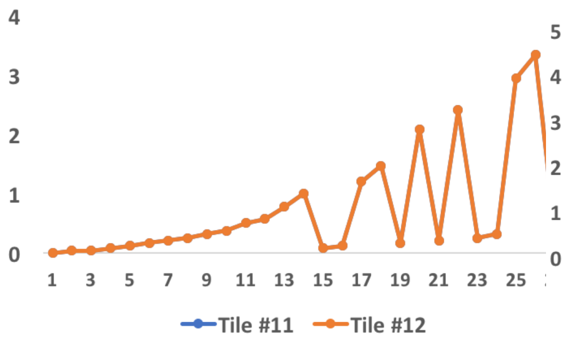
To establish the benchmark for comparison purpose, streaming over the traditional network is conducted first. Without any traffic engineering to optimize ROI's bitrate,



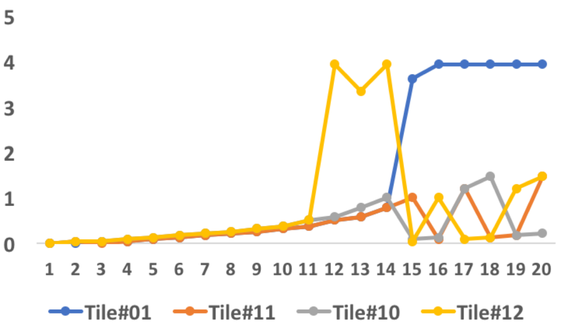
(a) BenchMark Bitrate With No SDN



(b) SDN: One ROI Bitrate



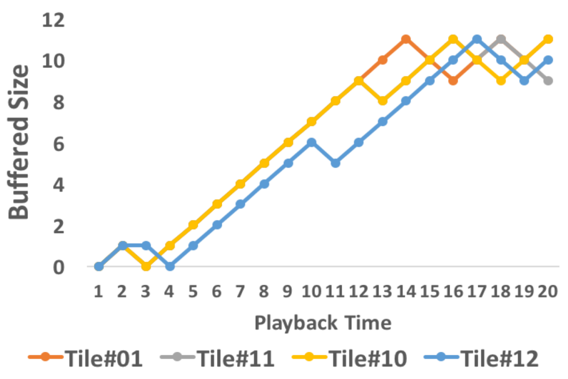
(c) SDN: Two ROIs Bitrates



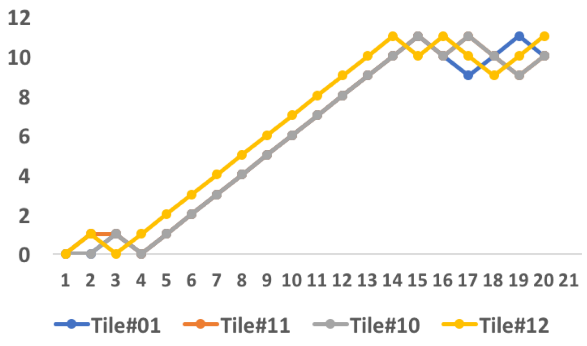
(d) SDN: One ROI Adaptive Bitrate

Figure 50: Bitrate for Both Non-SDN and SDN deployment

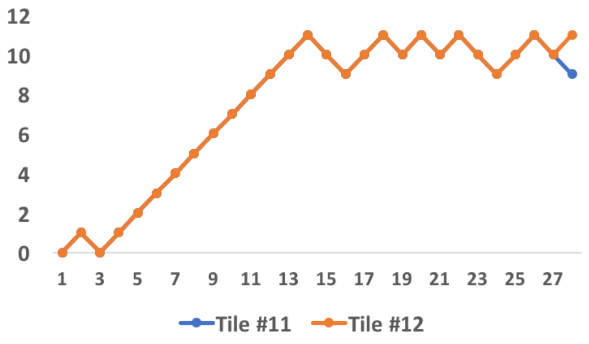
all the tiles are streaming over one single link. Figure 50(a) and 51(a) depict the bitrate and buffer information for each tile flow. The traffic pattern displays a typical TCP bandwidth allocations among all traffic. The average downloaded bitrate for each tile is $[0.5, 0.4, 0.4, 0.3]$, with an overall average bitrate of 0.4 Mbps. The average buffer size is $[6.7, 6.5, 6.5, 5.7]$, with an overall average buffer of 6.3 units.



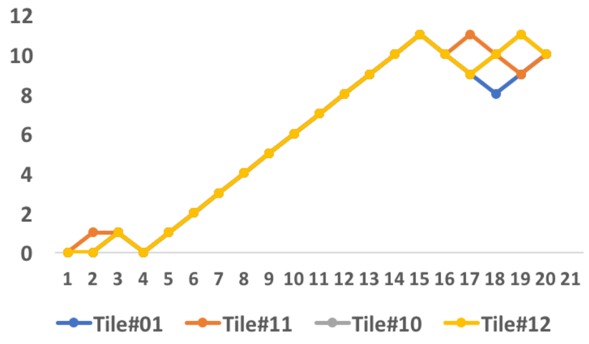
(a) BenchMark Buffer With No SDN



(b) SDN: One ROI Buffer



(c) SDN: Two ROIs Buffer



(d) SDN: One ROI Adaptive

Figure 51: Buffered Size for Both Non-SDN and SDN deployment

### 6.2.9 ROI Bitrate Optimization With SDN Deployed

In this section, we first assume viewer's ROI is on the tile $\#12$, shown in Figure 49. The SDN controller's SRD parser module parses the feedback from the viewer and installs flows on the path [ovs-2, ovs-3, ovs3-4] in Figure 48 for that particular flow. The rest of tiles stayed with the original path [ovs-2, ovs-1, ovs-4]. Figure 50(b) shows that the bitrate rate increases after client's initial setup, which is around ten segments. The average downloaded bitrate for each tile is $[0.4, 0.4, 0.4, 2.0]$, with an overall average bitrate of 0.8 Mbps. The highest bitrate $1.93$ is for tile $\#12$, which increases viewer's quality of experience by increasing the bitrate for his/her ROI. Average buffer size in Figure 51(b) is $[6.2, 6.2, 6.2, 6.7]$, with an overall average buffer of 6.3 units. Even though the average buffer is approximately equal to the previous test case, the higher buffer size for ROI tile $\#12$ is increased by 1 buffer unit.

Then we assume viewer's ROI is covered by tile $\#12$ and $\#11$.In this case, ROI and Non-ROI tiles are split equally by two paths. Figure 50(c) shows the average bitrate for ROI tiles' average bitrate $[0.7, 0.9]$, with an overall average bitrate of 0.8 Mbps. The Average buffer size in Figure 51(c) is $[7.7, 7.7]$, with an overall average buffer of 7.7 units. In both metrics, they are better than cases with Non-SDN deployed.

### 6.2.10 ROI Switchover with SDN Deployed

In this test case, we adaptively change viewer's ROI from ROI-A to another ROI-A' shown in Figure 49. The new ROI is still covered by one tile. At the beginning of playback, segments are split by two paths to increase initial ROI's ($\#12$) bitrate. We assume that ROI switches to $\#01$ right after finish streaming the 15-th segment. Figure 50(d) shows the average downloaded bitrate for each tile as $[1.4, 0.4, 0.4, 0.9]$, with an overall average bitrate of 0.8 Mbps. The highest bitrate $1.41$ Mbps is for new ROI with tile $\#01$. Previous ROI tile $\#12$ has average bitrate 0.9 Mbps. Average buffer size in Figure 51(d) is $[5.9, 6.2, 6.1, 6.1]$, with an overall average buffer of 6.1 units. Slightly low buffer size compared with one ROI optimization test case is negligible.

In summary, from Figure 52, with the SDN-assisted framework, user's QoE can be increased by increasing average ROI's bitrate in both fixed ROI and ROI switchover cases. In our test cases, the increased bitrate is around $100\%$ and buffer size $6\%$.

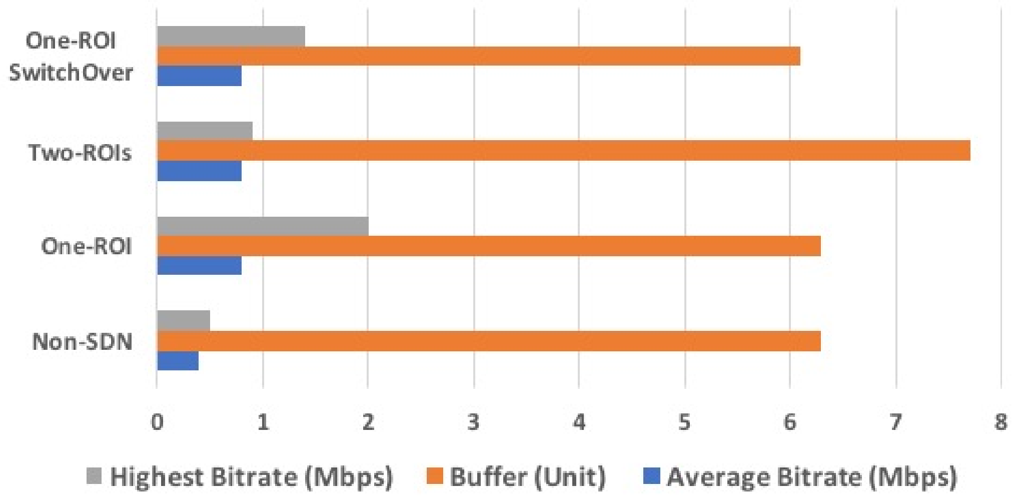


Figure 52: QoE Improvement Comparison

# CHAPTER 7

# CONCLUSION

Our approach in this work was to develop an Application-aware networking (AAN) environment with modularized components and fine-grained control network behavior to improve application's performance without changing the underlying design of the application itself. We presented a software-defined network based system architecture and over-the-top (OTT) software applications for the Hadoop MapReduce and MPEG-DASH based video streaming data flow control. We proposed a general application interface regarding a traffic flow alternation mechanism.

With Regarding to Hadoop MapReduce applications, we identified major traffic patterns of various MapReduce workloads based on HiBench benchmark suites using different configurations, which is the key to understanding the problem caused by data shuffle. A primary goal of this work was to demonstrate a concept of AAN using SDN implementation and MapReduce performance improvement without alternating the existing Hadoop configuration.

To assist flat 2D video streaming over the internet, we presented a software-defined network based system architecture to improve regular 2d flat DASH 's-based video streaming performance. A primary goal of this work was to demonstrate a concept of AAN using SDN implementation and DASH streaming performance improvement without altering of the existing MPEG-DASH client's configuration.

The presented AAN architecture performed better when compared to the traditional end-client adaptation model. Our model was evaluated and tested on two client adaptation schemes. We also found that SARA was better at adapting to change while minimizing switching events compared to buffer based adaptation when the originally designated path was overloaded. In all tests, we observed a significant reduction of bitrate switchover events that signified improvement of QoE of video streaming.

We also summarize the difference in immersive content streaming between traditional- and SDN-based network and introduce a SDN-based framework to assist tile-based immersive content streaming. Our goal was to develop a SDN-assisted framework to improve user QoEs for streaming new 3D immersive media such as Virtual Reality and 360-degree video using DASH without changing the underlying design of DASH client itself. We proposed a general application interface regarding a traffic flow alteration mechanism. By optimizing bitrate for viewer's ROI, our proposed framework reduces the overall transferred data size and increase bitrates for ROI to improve the overall viewing experience. The presented framework outperformed compared to the traditional end-client adaptation model. Our model is evaluated and tested on a number of ROI schemes. In all tests, we observed a significant ROI bitrate and buffer increase which signifies improvement of QoE of video streaming.

As future work, we plan to integrate tile-based streaming related issues such as a synchronization problem for different tiles and also increase the number of tiles with higher resolutions. Because each tile is essentially a separate TCP connection, each tile might have different playback timeline due to package delay in a congested network. We are also planning to include video traffic classification using data signature based dynamic machine learning models and explore additional traffic engineering (TE) methods as well as larger platforms for our approach. New transport protocols such as WebSocket can also be investigated for multiple ROIs switchover at the same time in a single duplex network flow for better network efficiency.

APPENDIX A

## MININET TOPOLOGY SOURCE CODE FOR DASH SETUP

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.log import lg, setLogLevel
from mininet.cli import CLI
from mininet.node import RemoteController
from mininet.node import Host
from functools import partial
from mininet.link import TCLink
CORES = {
    'gw1': {'dpid': '000000000000010%s'},
    'gw2': {'dpid': '000000000000020%s'},
}
# number of nodes
FANOUT = 10
class I2Topo(Topo):
    def __init__(self, enable_all=True):
        "Create␣Smart␣Home␣topology."
        # Add default members to class.
        super(I2Topo, self).__init__()

        # Add core switches
        self.cores = {}
        self.Host_count = 0
        for switch in CORES:
            self.cores[switch] = self.addSwitch(
                switch, dpid=(CORES[switch]['dpid'] % '0'))
        # Add hosts and connect them to their core switch
        for switch in CORES:
            if switch == "gw1":
                FANOUT = 1
            else:
                FANOUT = 10
            for count in xrange(1, FANOUT + 1):
                self.Host_count = self.Host_count + 1
                # Add hosts
                host = 'h_%s_%s' % (switch, count)
                ip = '172.16.0.%s' % self.Host_count
                print "Switch Name: ", switch
                print "     Host IP address: ", ip
                mac = CORES[switch]['dpid'][4:] % count
                h = self.addHost(host, ip=ip, mac=mac)
                # Connect hosts to core switches
                self.addLink(h, self.cores[switch])
        # Connect core switches
        self.addLink(self.cores['gw1'], self.cores['gw2'])
if __name__ == '__main__':
    topo = I2Topo()
    ip = '10.0.0.224'
    # ip = '192.122.236.105'
    port = 6633
    c = RemoteController('c', ip=ip, port=port)
    # add private DIR for each host
```

```
privateDirs = [('/var/log', '/tmp/%(name)s/var/log'),
               ('/var/run', '/tmp/%(name)s/var/run'),
               '/var/mn']
host = partial(Host, privateDirs=privateDirs)
link = partial( TCLink, delay='2ms', bw=10 )
net = Mininet(topo=topo, host=host, autoSetMacs=True,
              xterms=False, controller=None, link=link)
net.addController(c)
net.addNAT().configDefault()
net.start()
CLI(net)
net.stop()
```

APPENDIX B

HADOOP INSTALLATION SOURCE CODE

```
#!/bin/bash
HadoopUserLogin="sudo␣su␣-␣hadoopuser"
echo -n "#####␣For␣all␣the␣Hadoop␣Nodes␣#####"
echo "Are␣you␣installing␣for␣Hadoop␣Master␣mode␣or␣slave␣(M/S)?"
echo -n "Enter␣M␣or␣S␣[ENTER]:␣"
read mode

echo $mode
if [ $mode == 'M' ] ; then
echo "Installing␣hadoop␣Master"
elif [ $mode == 'S' ] ; then
echo "Installation␣hadoop␣slave"
else
echo "wrong␣input"
exit
fi

echo -n "add␣java␣repo"
sudo apt-get update
sudo apt-get install -y software-properties-common
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java7-installer -y
sudo apt-get install openssh-client openssh-server -y
```

```
# Updata Java runtime
sudo update-java-alternatives -s java-7-oracle
sud apt-get install -y vim
# Disable IPv6 (Skip this step if you are not using IPv6)
sudo sed -i 's/net.ipv6.bindv6only\ =\ 1/net.ipv6.bindv6only\
=\ 0/' \ /etc/sysctl.d/bindv6only.conf
&& sudo invoke-rc.d procps restart
echo -n "Setting up a Hadoop User"
sudo addgroup hadoopgroup
sudo adduser hadoopuser
sudo adduser hadoopuser hadoopgroup
# sudo delgroup hadoopgroup
# sudo deluser hadoopuser

echo -n "#####For Master node only #####"
if [ $mode == 'M' ]; then
echo -n "Login as hadoopuser and Generate ssh key "
$HadoopUserLogin -c "whoami"

$HadoopUserLogin -c "ssh-keygen -t rsa -P ''"
#Authorize the key to enable password less ssh
$HadoopUserLogin -c
"cat /home/hadoopuser/.ssh/id_rsa.pub >>
  /home/hadoopuser/.ssh/authorized_keys "
$HadoopUserLogin -c "chmod 600 ~/.ssh/authorized_keys"
echo -n "You need to copy id_ras.pub to slaves authorized_keys"
echo -n "Also add hosts's IP in your hadoop slaves files"
#Copy this key to slave-1 to enable password less ssh
#$ ssh-copy-id -i ~/.ssh/id_rsa.pub slaves/IPaddress
```

```
#Make sure you can do a password
#less ssh using following command.
#$ ssh slaves/IPaddress
fi

echo -n "#####␣For␣all␣nodes␣#####"

$HadoopUserLogin -c "wget
␣␣http://apache.mirrors.ionfish.org/hadoop/
␣␣␣␣common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
tar␣xzvf␣hadoop-2.6.0.tar.gz"
# change hadoop source folder to hadoop
# (not necessary, just for easy remember purpose)
$HadoopUserLogin -c "mv␣hadoop-2.6.0␣hadoop"
# set up environment variables for Master and slaves
$HadoopUserLogin -c "cat␣>>␣~/.bashrc␣<<␣EOF
#␣Set␣HADOOP_HOME
export␣HADOOP_HOME=/home/hadoopuser/hadoop
#␣Set␣JAVA_HOME
export␣JAVA_HOME=/usr/lib/jvm/java-7-oracle
#␣Add␣Hadoop␣bin␣and␣sbin␣directory␣to␣PATH
export␣PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
EOF
"
$HadoopUserLogin -c  "source␣~/.bashrc"

$HadoopUserLogin -c
  'echo -n "␣need␣to␣mannualy␣update␣hadoop-env" '
$HadoopUserLogin -c
```

```
  'echo -n "hadoop-env.sh
  # export JAVA_HOME=/usr/lib/jvm/java-7-oracle"'

echo -n "udpate core-site.xml for all nodes"
$HadoopUserLogin -c
  'cat > /home/hadoopuser/hadoop/etc/hadoop/core-site.xml
    << EOF
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/hadoopuser/tmp</value>
<description>Temporary Directory.</description>
</property>

<property>
<name>fs.defaultFS</name>
<value>hdfs://192.168.0.1:54310</value>
<description>Use HDFS as file storage engine</description>
</property>
</configuration>
EOF
'

echo -n "##### Update Master only #####"

if [ $mode == 'M' ]; then
echo -n "update master mapred-site.xml"
```

```
$HadoopUserLogin -c
  'cat > /home/hadoopuser/hadoop/etc/hadoop/mapred-site.xml
    << EOF
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>mapreduce.jobtracker.address</name>
<value>http://192.168.0.1:54311</value>
<description>The host and port that the MapReduce
job tracker runs at. If â localâ , then jobs are run
in-process as a single map and reduce task.
</description>
</property>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
<description>
  The framework for running mapreduce jobs
  </description>
</property>
</configuration>
EOF
'
fi

echo -n "#####␣For␣all␣nodes␣#####"

echo -n "update␣hdfs-site.xml␣for␣all␣nodes"
```

```
$HadoopUserLogin -c
  'cat > /home/hadoopuser/hadoop/etc/hadoop/hdfs-site.xml
    << EOF
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>dfs.replication </name>
<value>2</value>
<description>Default block replication.
The actual number of replications can be
specified when the file is created.
The default is used if replication is
not specified in create time.
</description>
</property>
<property>
<name>dfs.namenode.name.dir </name>
<value>/home/hadoopuser/hdfs/namenode</value>
<description>Determines where on the local
filesystem the DFS name node should store
the name table(fsimage). If this is a comma-delimited
list of directories then the name table is replicated
in all of the directories, for redundancy.
</description>
</property>
<property>
<name>dfs.datanode.data.dir </name>
```

```
<value>/home/hadoopuser/hdfs/datanode</value>
<description>Determines where on the local filesystem
an DFS data node should store its blocks. If this is a c
omma-delimited list of directories, then data will
be stored in all named directories, typically on
different devices. Directories that do
not exist are ignored.
</description>
</property>
<property>
<name> dfs.block.size </name>
<value>33554432</value>
<description>File block size, default on Hadoop
2.6 is 128 MB. I changed to 32MB =  32 * 1024 * 1024
  = 33554432 bytes
</description>
</property>
</configuration>
EOF
'
echo -n "update yarn-site.xml"

$HadoopUserLogin -c
  'cat > /home/hadoopuser/hadoop/etc/hadoop/yarn-site.xml
    << EOF
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
```

```
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
 </property>
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>192.168.0.1:8030</value>
 </property>
<property>
  <name>yarn.resourcemanager.address</name>
  <value>192.168.0.1:8032</value>
 </property>
<property>
   <name>yarn.resourcemanager.webapp.address</name>
   <value>0.0.0.0:8088</value>
 </property>
<property>
   <name>
       yarn.resourcemanager.resource-tracker.address
          </name>
   <value>192.168.0.1:8031</value>
 </property>
<property>
   <name>yarn.resourcemanager.admin.address</name>
   <value>192.168.0.1:8033</value>
 </property>
 </configuration>
EOF
'
```

```
echo -n "update␣bashrc␣file"

$HadoopUserLogin -c
'cat >> /home/hadoopuser/.bashrc << EOF
export
  PATH=
     $PATH:/home/hadoopuser/hadoop/bin:\
     /home/hadoopuser/hadoop/sbin:/sbin
EOF
'
$HadoopUserLogin -c 'source /home/hadoopuser/.bashrc'

echo -n "done␣installation␣and␣configure..NEXT...."

echo -n "add␣hadoopuser␣to␣root␣group...."
sudo adduser hadoopuser root

echo -e "To␣Begin␣using␣Hadoop:
su␣-␣hadoopuser\n
1.␣you␣need␣to␣edit␣master␣slaves␣file␣\n
2.␣start␣your␣master␣node␣$␣hdfs␣namenode␣-format␣\n
3.␣need␣to␣mannualy␣update␣hadoop-env.xml␣change
␣␣at␣hadoop-env.sh␣\n
␣␣#␣export␣JAVA_HOME=/usr/lib/jvm/java-7-oracle␣\n
4.␣modify␣/etc/hosts␣files␣for␣all␣the␣nodes\n␣\n
5␣copy␣master's␣id_ras.pub␣to␣slaves␣authorized_keys␣␣\n
6.␣$␣start-dfs.sh␣\n6.␣start-yarn.sh␣"
exit
```

# APPENDIX C

## OPENVSWITCH INSTALLATION SOURCE CODE

```
#!/bin/bash
echo "make␣sure␣\nOS:␣Ubuntu␣14.04.1␣Server␣LTS␣X86_64
Kernel␣version:␣3.13.0-34-generic"
sudo apt-get update
sudo aptitude install dh-autoreconf libssl-dev openssl
echo "compile␣OVS␣2.3.1,␣only␣support␣Linux␣kernel␣<=␣3.13"
cd ~
wget http://openvswitch.org/releases/openvswitch-2.3.1.tar.gz
tar zxvf openvswitch-2.3.1.tar.gz && cd openvswitch-2.3.1
./boot.sh
./configure --with-linux=/lib/modules/`uname -r`/build
make -j && sudo make install
sudo make modules_install
sudo modprobe gre
sudo modprobe openvswitch
sudo modprobe libcrc32c
sudo lsmod |grep openvswitch
sudo ovsdb-tool \
        create /usr/local/etc/openvswitch/conf.db \
        /usr/local/share/openvswitch/vswitch.ovsschema \
echo "setup␣ovsdb-server"
sudo ovsdb-server \
        --remote=punix:/usr/local/var/run/openvswitch/db.sock \
        --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
        --pidfile --detach --log-file
echo "check OVSdb log"
cat /usr/local/var/log/openvswitch/ovsdb-server.log
echo "open ovs-vsctl"
sudo ovs-vsctl --no-wait init
echo "open ovs-vswitchd"
sudo ovs-vswitchd --pidfile --detach --log-file
echo "Auto open when reboot , Enter root mode"
su
echo "openvswitch " >> /etc/modules
echo "gre" >> /etc/modules
echo "libcrc32c" >> /etc/modules
cat > /etc/init.d/openvswitch << EOF
#!/bin/sh
start-stop-daemon -q -S -x \
        /usr/local/sbin/ovsdb-server -- \
        --remote=punix:/usr/local/var/run/openvswitch/db.sock \
        --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
        --pidfile --detach --log-file
sleep 3 # waiting ovsdb-server
start-stop-daemon -q -S -x /usr/local/bin/ovs-vsctl \
        -- --no-wait init
start-stop-daemon -q -S -x /usr/local/sbin/ovs-vswitchd \
        -- --pidfile --detach --log-file
EOF
chmod +x /etc/init.d/openvswitch
update-rc.d -f openvswitch defaults
exit
sudo ovs-vsctl show
```

APPENDIX D

HIBENCH INSTALLATION GUIDE

```
#!/bin/bash

echo "see details at http://github.co/inter-hadoop/HiBench"
cd ~
git clone https://github.com/intel-hadoop/HiBench.git
cd HiBench

Build HiBench
echo "install maven"
sudo apt-get install maven
mvn -Dscala=2.11 clean package
mvn -Dscala=2.10 clean package

Basic HiBench Configuration Setup:
hibench.hadoop.home       /home/hadoopuser/hadoop
# The path of hadoop executable
hibench.hadoop.executable     ${hibench.hadoop.home}/bin/hadoop
# Hadoop configrautionn directory
hibench.hadoop.configure.dir  ${hibench.hadoop.home}/etc/hadoop
# The root HDFS path to store HiBench data
hibench.hdfs.master          hdfs://192.168.0.1:54310
# Hadoop release provider. Supported value: apache, cdh5, hdp
hibench.hadoop.release      apache
```

```
Change Data Input Profile:
cd conf/
vim hibench.conf
hibench.scale.profile  tiny/small/large/huge/gigantic/bigdata

Example: Change Data Size for each workload Profile
Take wordcount as one example in
        /HiBench/conf/workloads/micro/wordcount.conf
I change to 16MB, 48MB, 80MB, 112MB, 240Mb, 304MB
hibench.wordcount.tiny.datasize   16777216
hibench.wordcount.small.datasize  50331648
hibench.wordcount.large.datasize  83886080
hibench.wordcount.huge.datasize  117440512
hibench.wordcount.gigantic.datasize     251658240
hibench.wordcount.bigdata.datasize      318767104

Tuning:
cd conf/
vim hibench.conf
hibench.default.map.parallelism          Mapper number in hadoop
hibench.default.shuffle.parallelism  Reducer number in hadoop
```

# REFERENCE LIST

[1] Abhishek, R., Zhao, S., and Medhi, D. SPArTaCuS: Service priority adaptiveness for emergency traffic in smart cities using software-defined networking. In *Smart Cities Conference (ISC2), 2016 IEEE International* (2016), IEEE, pp. 1–4.

[2] Abhishek, R., Zhao, S., Song, S., Choi, B.-Y., Zhu, H., and Medhi, D. BuDDI: Bug detection, debugging, and isolation middlebox for software-defined network controllers. In *Network and Service Management (CNSM), 2016 12th International Conference on* (2016), IEEE, pp. 307–311.

[3] Abhishek, R., Zhao, S., Tipper, D., and Medhi, D. SeSAMe: Software defined smart home alert management system for smart communities. In *Local and Metropolitan Area Networks (LANMAN), 2017 IEEE International Symposium on* (2017), IEEE, pp. 1–6.

[4] Astream. A DASH segment size aware rate adaptation model for DASH. https://github.com/pari685/AStream. Access Date: Dec 2016.

[5] Babu, S. Towards automatic optimization of MapReduce programs. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (2010), ACM, pp. 137–142.

[6] Becerra, Y., Beltran, V., Carrera, D., González, M., Torres, J., and Ayguadé, E. Speeding up distributed MapReduce applications using hardware accelerators. In

*Parallel Processing, 2009. ICPP'09. International Conference on* (2009), IEEE, pp. 42–49.

[7] Borthakur, D. The hadoop distributed file system: Architecture and design. https://svn.apache.org/repos/asf/hadoop/common/tags/release-0.12.0, 2008.

[8] Chakraborti, A., Rajaraman, V., Zhao, S., Azgin, A., Ravindran, R., and Wang, G. Demo overview: multi-party conference over virtual service edge router (vser) platform. In *Proceedings of the 1st International Conference on Information-centric Networking* (2014), ACM, pp. 205–206.

[9] Chen, Y., Griffit, R., Zats, D., and Katz, R. H. Understanding TCP Incast and Its Implications for Big Data Workloads. Tech. Rep. UCB/EECS-2012-40, University of California at Berkeley, Apr 2012.

[10] Cheung, G., Liu, Z., Ma, Z., and Tan, J. Z. Multi-Stream Switching for Interactive Virtual Reality Video Streaming. *arXiv preprint arXiv:1703.09090* (2017).

[11] Chowdhury, M., and Stoica, I. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks* (2012), ACM, pp. 31–36.

[12] Chowdhury, M., Zaharia, M., Ma, J., Jordan, M. I., and Stoica, I. Managing data transfers in computer clusters with orchestra. In *ACM SIGCOMM Computer Communication Review* (2011), vol. 41, ACM, pp. 98–109.

[13] Cisco. Cisco visual networking index: Forecast and methodology. *CISCO White paper: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.htm* (2012), 2011–2016.

[14] Concolato, C., Le Feuvre, J., Denoual, F., Maze, F., Ouedraogo, N., and Taquet, J. Adaptive Streaming of HEVC Tiled Videos using MPEG-DASH. *IEEE Transactions on Circuits and Systems for Video Technology* (2017).

[15] Costa, P., Donnelly, A., Rowstron, A., and O'Shea, G. Camdoop: Exploiting in-network aggregation for big data applications. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (2012), USENIX Association, pp. 3–3.

[16] De Cicco, L., Caldaralo, V., Palmisano, V., and Mascolo, S. Elastic: A client-side controller for dynamic adaptive streaming over http (dash). In *Packet Video Workshop (PV), 2013 20th International* (2013), IEEE, pp. 1–8.

[17] Dobrian, F., Sekar, V., Awan, A., Stoica, I., Joseph, D., Ganjam, A., Zhan, J., and Zhang, H. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM Computer Communication Review* (2011), vol. 41, ACM, pp. 362–373.

[18] El-Ganainy, T., and Hefeeda, M. Streaming virtual reality content. *arXiv preprint arXiv:1612.08350* (2016).

[19] Elliott, C. GENI-global environment for network innovations. In *LCN* (2008), p. 8.

[20] Google. ExoPlayer: An extensible media player for Android. https://github.com/google/ExoPlayer. Access Date: 07/2016.

[21] Google. The hrome Web Browser. http://google.com/chrome, 2008.

[22] Graf, M., Timmerer, C., and Mueller, C. Towards bandwidth efficient adaptive streaming of omnidirectional video over HTTP: Design, implementation, and evaluation. In *Proceedings of the 8th ACM on Multimedia Systems Conference* (2017), ACM, pp. 261–271.

[23] Gu, R., Yang, X., Yan, J., Sun, Y., Wang, B., Yuan, C., and Huang, Y. SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters. *Journal of Parallel and Distributed Computing Vol. 74* (2014), 2166–2179.

[24] Hammoud, M., and Sakr, M. F. Locality-aware reduce task scheduling for MapReduce. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on* (2011), IEEE, pp. 570–576.

[25] He, C., Lu, Y., and Swanson, D. Matchmaking: A new MapReduce scheduling technique. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on* (2011), IEEE, pp. 40–47.

[26] Holmes, A. *Hadoop in Practice*. Manning Publications Co., 2012.

[27] Holmes, G., Donkin, A., and Witten, I. H. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on* (1994), IEEE, pp. 357–361.

[28] Hosseini, M. View-aware tile-based adaptations in 360 virtual reality video streaming. In *Virtual Reality (VR), 2017 IEEE* (2017), IEEE, pp. 423–424.

[29] Hosseini, M., and Swaminathan, V. Adaptive 360 VR video streaming based on MPEG-DASH SRD. In *Multimedia (ISM), 2016 IEEE International Symposium on* (2016), IEEE, pp. 407–408.

[30] Huang, S., Huang, J., Dai, J., Xie, T., and Huang, B. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on* (2010), IEEE, pp. 41–51.

[31] ISO/IEC. 23009-1:2014/Amd 2:2015, Spatial relationship description, generalized URL parameters and other extensions. https://www.iso.org/standard/65274.html.

[32] ITEC. Big Buck Bunny Movie. http://www-itec.uni-klu.ac.at/ftp/, 2015. Access Date: Dec 2016.

[33] Jiang, J., Sekar, V., and Zhang, H. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (2012), ACM, pp. 97–108.

[34] Jin, J., Luo, J., Song, A., Dong, F., and Xiong, R. Bar: An efficient data locality driven task scheduling algorithm for cloud computing. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2011), IEEE Computer Society, pp. 295–304.

[35] Juluri, P., Tamarapalli, V., and Medhi, D. SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *Communication Workshop (ICCW), 2015 IEEE International Conference on* (2015), IEEE, pp. 1765–1770.

[36] Juluri, P., Tamarapalli, V., and Medhi, D. Measurement of quality of experience of video-on-demand services: A survey. *IEEE Communications Surveys & Tutorials Vol. 18* (2016), 401–418.

[37] Juluri, P., Tamarapalli, V., and Medhi, D. QoE management in DASH systems using the segment aware rate adaptation algorithm. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP* (2016), IEEE, pp. 129–136.

[38] Khiem, N. Q. M., Ravindra, G., and Ooi, W. T. Adaptive encoding of zoomable video streams based on user access pattern. *Signal Processing: Image Communication 27*, 4 (2012), 360–377.

[39] Kreutz, D., Ramos, F. M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., and Uhlig, S. Software-defined networking: A comprehensive survey. *proceedings of the IEEE Vol. 103* (2015), 14–76.

[40] Krishnan, S. S., and Sitaraman, R. K. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking Vol. 21* (2013), 2001–2014.

[41] Lam, C. *Hadoop in Action*. Manning Publications Co., 2010.

[42] Le Feuvre, J., and Concolato, C. Tiled-based adaptive streaming using MPEG-DASH. In *Proceedings of the 7th International Conference on Multimedia Systems* (2016), ACM, p. 41.

[43] Li, B., Mazur, E., Diao, Y., McGregor, A., and Shenoy, P. A platform for scalable one-pass analytics using MapReduce. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (2011), ACM, pp. 985–996.

[44] Li, Z., and I.Bouazizi. FF: Temporal Quality Signalling in ISO Based Media File Format. ISO/IEC/JTC1/MPEG2014/m33239, 2014.

[45] Li, Z., Zhao, S., Medhi, D., and Bouazizi, I. Wireless video traffic bottleneck coordination with a DASH SAND framework. In *Visual Communications and Image Processing (VCIP), 2016* (2016), IEEE, pp. 1–4.

[46] Li, Z., Zhu, X., Gahm, J., Pan, R., Hu, H., Begen, A. C., and Oran, D. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications Vol. 32* (2014), 719–733.

[47] Lim, S. Y., Seok, J. M., Seo, J., and Kim, T. G. Tiled panoramic video transmission system based on MPEG-DASH. In *Information and Communication Technology Convergence (ICTC), 2015 International Conference on* (2015), IEEE, pp. 719–721.

[48] Mao, H., Hu, S., Zhang, Z., Xiao, L., and Ruan, L. A load-driven task scheduler with adaptive DSC for MapReduce. In *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on* (2011), IEEE, pp. 28–33.

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

[50] Min, W., Hannu, H., Pettersson, J., and Timner, Y. Optimization of fairness for HTTP adaptive streaming with network assistance in LTE mobile systems. In *Vehicular Technology Conference (VTC Fall), 2014 IEEE 80th* (2014), IEEE, pp. 1–5.

[51] Mueller, C. MPEG-DASH in a Nutshell. https://bitmovin.com/dynamic-adaptive-streaming-http-mpeg-dash/. Access Date: 08/2016.

[52] Neves, M. V., De Rose, C. A., Katrinis, K., and Franke, H. Pythia: Faster big data in motion through predictive software-defined network optimization at runtime. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International* (2014), IEEE, pp. 82–90.

[53] Niamut, O., Prins, M., Brandenburg, R. v., and Havekes, A. Spatial tiling and streaming in an immersive media delivery network. *9th European Interactive TV Conference, EuroITV'11, Lisbon* (2011).

[54] Niamut, O. A., Thomas, E., D'Acunto, L., Concolato, C., Denoual, F., and Lim, S. Y. MPEG DASH SRD: Spatial relationship description. In *Proceedings of the 7th International Conference on Multimedia Systems* (2016), ACM, p. 5.

[55] OSRG. Framework community: Ryu sdn controller. https://osrg.github.io/ryu/, 2016.

[56] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., and Stonebraker, M. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data* (2009), ACM, pp. 165–178.

[57] Pióro, M., and Medhi, D. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Elsevier, 2004.

[58] Prins, M., Niamut, O., van Brandenburg, R., Macq, J.-F., Rondao Alface, P., and Verzijp, N. A hybrid architecture for delivery of panoramic video. In *Proceedings of the 11th European Conference on Interactive TV and Video* (2013), ACM, pp. 99–106.

[59] Qian, F., Ji, L., Han, B., and Gopalakrishnan, V. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges* (2016), ACM, pp. 1–6.

[60] Romirer-Maierhofer, P., Ricciato, F., DÕAlconzo, A., Franzan, R., and Karner, W. Network-wide measurements of TCP RTT in 3G. *Traffic Monitoring and Analysis* (2009), 17–25.

[61] Rmi Houdaille, C. T. DASH/CESAND: Cooperative parameters. ISO/IEC JTC1/SC29/WG11 MPEG 111/m36033, 2015.

[62] Seo, S., Jang, I., Woo, K., Kim, I., Kim, J.-S., and Maeng, S. HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment. In *Cluster Computing and Workshops* (2009), IEEE, pp. 1–8.

[63] Sharma, B., Prabhakar, R., Lim, S.-H., Kandemir, M. T., and Das, C. R. Mrorchestrator: A fine-grained resource orchestration framework for MapReduce clusters. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on* (2012), IEEE, pp. 1–8.

[64] Sitaraman, R. K. Network performance: Does it really matter to users and by how much? In *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on* (2013), IEEE, pp. 1–10.

[65] Spiteri, K., Urgaonkar, R., and Sitaraman, R. K. BOLA: Near-optimal bitrate adaptation for online videos. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on* (2016), IEEE, pp. 1–9.

[66] Stockhammer, T. Dynamic adaptive streaming over HTTP: standards and design principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems* (2011), ACM, pp. 133–144.

[67] TaghaviNasrabadi, A., Mahzari, A., Beshay, J. D., and Prakash, R. Adaptive 360-degree video streaming using layered video coding. In *Virtual Reality (VR), 2017 IEEE* (2017), IEEE, pp. 347–348.

[68] Tan, J., Meng, X., and Zhang, L. Coupling task progress for MapReduce resource-aware scheduling. In *INFOCOM, 2013 Proceedings IEEE* (2013), IEEE, pp. 1618–1626.

[69] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (2014), ACM, pp. 147–156.

[70] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (2013), ACM, p. 5.

[71] Vernica, R., Balmin, A., Beyer, K. S., and Ercegovac, V. Adaptive MapReduce using situation-aware mappers. In *Proceedings of the 15th International Conference on Extending Database Technology* (2012), ACM, pp. 420–431.

[72] White, T. *Hadoop: The Definition Guide*. OReilly Media, Inc, 2009.

[73] Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., and Qin, X. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on* (2010), IEEE, pp. 1–9.

[74] Xin, M., and Li, H. An implementation of GPU accelerated MapReduce: Using Hadoop with OpenCL for data-and compute-intensive jobs. In *Service Sciences (IJCSS), 2012 International Joint Conference on* (2012), IEEE, pp. 6–11.

[75] Yao, J., Kanhere, S. S., and Hassan, M. An empirical study of bandwidth predictability in mobile computing. In *Proceedings of the third ACM international workshop on Wireless network testbeds, experimental evaluation and characterization* (2008), ACM, pp. 11–18.

[76] You, H.-H., Yang, C.-C., and Huang, J.-L. A load-aware scheduler for MapReduce framework in heterogeneous cloud environments. In *Proceedings of the 2011 ACM Symposium on Applied Computing* (2011), ACM, pp. 127–132.

[77] Yu, W., Wang, Y., and Que, X. Design and evaluation of network-levitated merge for Hadoop acceleration. *IEEE Transactions on Parallel and Distributed Systems 25*, 3 (2014), 602–611.

[78] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems* (2010), ACM, pp. 265–278.

[79] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., and Stoica, I. Improving MapReduce performance in heterogeneous environments. In *Osdi* (2008), vol. 8, p. 7.

[80] Zhao, S., Leftwich, K., Owens, M., Magrone, F., Schonemann, J., Anderson, B., and Medhi, D. I-can-mama: Integrated campus network monitoring and management. In *Network Operations and Management Symposium (NOMS), 2014 IEEE* (2014), IEEE, pp. 1–7.

[81] Zhao, S., Li, Z., and Medhi, D. Low delay MPEG DASH streaming over the WebRTC data channel. In *Multimedia & Expo Workshops (ICMEW), 2016 IEEE International Conference on* (2016), IEEE, pp. 1–6.

[82] Zhao, S., Li, Z., and Medhi, D. Low delay MPEG DASH streaming over the WebRTC data channel. In *Multimedia & Expo Workshops (ICMEW), 2016 IEEE International Conference on* (2016), IEEE, pp. 1–6.

[83] Zhao, S., Li, Z., and Medhi, D. Low delay streaming of DASH content with WebRTC data channel. In *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on* (2016), IEEE, pp. 1–2.

[84] Zhao, S., Li, Z., Medhi, D., Lai, P., and Liu, S. Study of user QoE improvement for dynamic adaptive streaming over HTTP (MPEG-DASH). In *Computing, Networking and Communications (ICNC), 2017 International Conference on* (2017), IEEE, pp. 566–570.

[85] Zhao, S., and Medhi, D. Application-Aware network design for Hadoop MapReduce optimization using software-defined networking. *IEEE Transactions on Network and Service Management* (2017).

[86] Zhao, S., and Medhi, D. Tile-based streaming with MPEG-DASH using Software-defined Network. In *The 3rd IEEE Conference on Network Functions Virtualization and Software Defined Networking (IEEE NFV-SDN 2017)* (2017), IEEE.

[87] Zhao, S., Sydney, A., and Medhi, D. Building Application-Aware Network Environments using SDN for Optimizing Hadoop Applications. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference (poster paper)* (2016), ACM, pp. 583–584.

# VITA

Shuai Zhao was born on Dec 15th, 1986 in Qingdao, China. He received B.Sc in Applied Mathematics from Heze University, China and M.S. in Computer Science from the University of Missouri - Kansas City, USA.

Mr. Zhao currently works as a senior engineer at Mediatek USA Inc. He is an IEEE member and serves (or served) as a Technical Committee Member for IEEE NOMS, ICNC, NFV-SDN, ICIIP, Globecom, ANTS, NetCom, NoF, ICCC, WPMC, ICUMT, CyberNeticsCom, IPCCC, CNSM and designated reviewer for IEEE ICC, VCIP and ACM MMSys.

Mr. Zhao interned with Lenovo (2013), FutureWei (2014), Raython BBN (2015) and Mediatek Inc USA (2016). He has published over 15 papers. His research interests are Application-Awareness Networking, Software-Defined Networking, network design, big data, deep learning, MPEG-DASH and MPEG-I Standards, and VR/360 video streaming.