



University of  
Massachusetts  
Amherst

## IMPROVING USER EXPERIENCE BY OPTIMIZING CLOUD SERVICES

Item Type	Dissertation (Open Access)
Authors	Dasgupta, Ishita
DOI	<a href="https://doi.org/10.7275/35011393">10.7275/35011393</a>
Rights	Attribution 4.0 International
Download date	2026-05-14 17:25:32
Item License	<a href="http://creativecommons.org/licenses/by/4.0/">http://creativecommons.org/licenses/by/4.0/</a>
Link to Item	<a href="https://hdl.handle.net/20.500.14394/19175">https://hdl.handle.net/20.500.14394/19175</a>

# IMPROVING USER EXPERIENCE BY OPTIMIZING CLOUD SERVICES

A Dissertation Presented

by

ISHITA DASGUPTA

Submitted to the Graduate School of the  
University of Massachusetts Amherst in partial fulfillment  
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2023

Manning College of Information and Computer Sciences

© Copyright by Ishita Dasgupta 2023

All Rights Reserved

# IMPROVING USER EXPERIENCE BY OPTIMIZING CLOUD SERVICES

A Dissertation Presented

by

ISHITA DASGUPTA

Approved as to style and content by:

---

Michael Zink, Chair

---

Prashant Shenoy, Member

---

Ramesh K. Sitaraman, Member

---

Vishy Swaminathan, Member

---

Susmit Shannigrahi, Member

---

Ramesh K. Sitaraman, Associate Dean for  
Educational Programs and Teaching  
Manning College of Information and Computer  
Sciences

## DEDICATION

*To all my mothers & Amma*

## ACKNOWLEDGMENTS

First and foremost, the last six years of my PhD journey is dedicated to and possible because of my advisor, Professor Michael Zink. His guidance, undying patience, constant positivity and the freedom to allow me to carve my own career path is everything that I am grateful for. It is his style of mentorship that has truly motivated me to do better, be better in the most balanced way possible. I would like to thank him for taking me in as a PhD student six years ago, followed by numerous opportunities to grow in the field of multimedia research. Under his direction, I got the chance to work on wide range of exciting fields from weather research to future networking as well as innovative multimedia technologies. I would also like to thank my extremely curious and hardworking lab-mates through the course of these six years, Cong Wang, Divyashri Bhat, Bhushan Suresh, Rajvardhan Deshmukh, Patrick Lieser, Rhaban Hark, Thiago Teixeira, Siamak Beikzadeh and John Murray. Not only did I learn a lot from interactions with them but their valuable inputs also accelerated and improved my work. Next, I'd like to thank Prof. Shenoy and Prof. Sitaraman whose indispensable courses, knowledge and guidance helped me strengthen the basics required to pursue this PhD journey. I would also like to extend my heartfelt thanks to Prof. Shannigrahi whose instructions and directions taught me ways to deal with deadlock situations in research. His constant guidance in the form of positive feedback through final years of my PhD is something that was always encouraging and motivating. I'm also incredibly thankful to Vishy Swaminathan for teaching me ways around industrial research and for being an excellent mentor with the best words of advice that always pushed me to ask more questions and in the process, learn more.

This milestone would have been impossible to dream, let alone achieve, if it weren't for the two strongest foundational pillars of my life - my father, Dr. Krishnendu Dasgupta and my mother, Shikha Dasgupta. Their blind support and undying faith in my capabilities has been instrumental into shaping my career. My father has always pushed me to do more, learn more, question more and aim more which made me confident enough to pursue my career choices at every step. I'm also thankful for Amma, my first teacher and role model, to whom I owe all my successes. And finally my husband, Sayak Bhowmick who has brought direction, stability and discipline in my life whilst being the most supportive partner, all of which have been majorly instrumental towards the completion of this journey.

Lastly, I'd like to acknowledge my gratitude towards some of my friends across the world who played a pivotal role towards the completion of my PhD journey - Dipa Nandi, Nikita Mehra, Kaushal Sumaria, Meet Vadera, Akanksha Atrey and Hia Ghosh. Without them, this journey would have definitely been harder.

## **ABSTRACT**

# **IMPROVING USER EXPERIENCE BY OPTIMIZING CLOUD SERVICES**

MAY 2023

ISHITA DASGUPTA

B.Tech, WEST BENGAL UNIVERSITY OF TECHNOLOGY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Michael Zink

Today, cloud services offer myriads of applications, tailor made for different users in the field of weather, health, finance, entertainment, etc. These services fulfill varying genres of user demands over the Internet. For example, these services can be live (live weather radar, ESPN Live) or on-demand services (weather forecasting, Netflix). While these applications cater to different customer requirements, it is necessary for these services to be efficient with respect to latency, scalability, robustness and quality of experience. These systems need to constantly evolve to provide the best user experience and meet the most current demands of the customer. For instance, weather forecasting systems need to make accurate predictions close to real-time in order to alert customers of an imminent disaster. Similarly, video-streaming systems consistently strive towards providing best viewing experience to the customer. As weather predictions become more fine-grained with high-resolution observations

and video streaming platforms evolve with higher resolution content, the underlying system needs to improve to deliver optimum performance.

Even though weather sensing and forecasting has become increasingly accurate in the last decade thanks to high-resolution radars, efficient computational algorithms, and high-performance computing facilities, there are existing challenges that could result in sub-optimal weather prediction. Flood-forecasting models are dependent on high-performance computing clusters for processing high-resolution rainfall data and their unavailability can become a bottleneck. Also, weather radars are critical infrastructure, but are often located in remote areas with poor network connectivity. Data retrieved from these radars are often delayed or lost, or even lack proper synchronization, resulting in less than ideal weather predictions. Additionally, as more and more users have access to the cloud-based video-streaming platforms, their viewing experience is often adversely affected by network conditions (congestion, limited bandwidth) and limitations of address-based Internet architectures.

In this dissertation, we try to address these existing challenges by designing and implementing systems on the cloud that improve end-user experience. We optimize two cloud services: In the field of safety we improve upon weather forecasting systems and in the field of entertainment, quality of experience in video-streaming systems. We utilize, investigate and incorporate benefits of different system technologies such as parallelization, virtualization, software defined networking and information centric networking in the existing system workflow with the goal of optimizing the quality of experience. Firstly, we relieve the dependency on high-performance computing clusters for a flood-forecasting model by using cloud resources as an alternative. We then optimize the performance of this model on the cloud by parallelly distributing its existing workflow, which leads to 34% reduction in rainfall processing time. The observed improvement in runtime is not only consistent with varying rainfall conditions but could also lead to faster forecasts on the cloud and potentially be extended to

different weather forecast models. Secondly, we improve data-retrieval and synchronization challenges in weather sensing radars by applying Named Data Networking (NDN) for efficient content addressing and retrieval. We identify weather data based on a hierarchical naming scheme to explicitly access desired data and demonstrate that compared to a time-based windowing mechanism for radar data retrieval on top of TCP/IP, an NDN based mechanism improves data quality, reduces uncertainty, and enhances weather prediction. Lastly, we improve quality of experience in live video streaming by designing a seamless and efficient distribution system that is flexible enough to choose from multiple Internet architectures best suited for live video streaming. We highlight the benefits of such a hybrid system for live video streaming as well as present a detailed analysis with the goal to provide a high quality of experience (QoE) for the viewer. For our hybrid architecture, video streaming is supported simultaneously over TCP/IP and Named Data Networking (NDN)-based architecture via operating system and networking virtualization techniques to design a flexible system that utilizes the benefits of these varying Internet architectures. Based on a prototype we have designed and implemented maintaining efficient use of network resources, we demonstrate that in the case of live streaming, NDN achieves better QoE per client than IP and can also utilize higher than the available bottleneck bandwidth through in-network caching. Even without caching, as opposed to IP-only, our hybrid setup achieves better average bitrate and better perceived visual quality over live video streaming services. Furthermore, we present detailed analysis on ways adaptive video streaming with NDN can be further improved with respect to QoE. In this context, we identify the phenomenon of oscillation effects that adversely affects QoE in adaptive video streaming and propose solutions to reduce them.

# TABLE OF CONTENTS

	Page
<b>ACKNOWLEDGMENTS</b> .....	v
<b>ABSTRACT</b> .....	vii
<b>LIST OF TABLES</b> .....	xiv
<b>LIST OF FIGURES</b> .....	xv
 <b>CHAPTER</b>	
<b>1. INTRODUCTION</b> .....	<b>1</b>
1.1 Weather Systems .....	2
1.2 Live video streaming systems .....	4
1.3 Dissertation Outline .....	5
<b>2. BACKGROUND AND RELATED WORK</b> .....	<b>8</b>
2.1 CloudLab .....	8
2.2 CASA .....	9
2.3 Flood-forecasting hydrologic Model .....	10
2.4 Weather sensing .....	11
2.4.1 Current Dataflow Paradigm .....	11
2.4.2 Asynchronous Data .....	12
2.5 Adaptive Bitrate Streaming and DASH .....	13
2.6 NDN .....	14
2.6.1 Related Work .....	15
<b>3. OPTIMIZING FLOOD FORECASTING ON THE CLOUD</b> .....	<b>17</b>
3.1 Hydrology characteristics in the DFW region .....	17
3.2 Methodology .....	19

3.2.1	Parallel Processing of headwater basins . . . . .	20
3.2.2	Interleaved processing of headwater and downstream basins . . . . .	22
3.3	Experimental Setup . . . . .	25
3.4	Results & Discussion . . . . .	29
3.4.1	Comparison of models with respect to rainfall processing runtime . . . . .	29
3.4.2	Effect of data-ingest intervals on rainfall processing runtime . . . . .	32
3.4.3	Comparison of different degrees of parallelization in HWB processing with respect to overall rainfall processing runtime . . . . .	34
3.4.4	Performance study under different rainfall conditions . . . . .	35
3.5	Conclusion & Future Work . . . . .	37
<b>4.</b>	<b>IMPROVING WEATHER SENSING WITH NAMED DATA NETWORKING . . . . .</b>	<b>38</b>
4.1	Weather Sensing with NDN . . . . .	39
4.2	Methodology/Design . . . . .	40
4.2.1	Scenario . . . . .	40
4.2.2	Data Retrieval in Rounds . . . . .	43
4.2.3	A Naming Scheme for Periodic Radar Data . . . . .	44
4.2.4	Piggybacking . . . . .	45
4.3	Evaluation . . . . .	45
4.3.1	Experiment Setup . . . . .	46
4.3.2	Results . . . . .	48
4.4	Conclusion and Future Work . . . . .	50
<b>5.</b>	<b>OPTIMIZING VIDEO STREAMING ON THE CLOUD . . . . .</b>	<b>52</b>
5.1	NDN and Live Streaming . . . . .	53
5.2	SDN and NFV support for Parallel Live Video Streaming . . . . .	54
5.3	Evaluation Setup . . . . .	56
5.3.1	Experimental Setup . . . . .	56
5.3.2	Live Streaming . . . . .	61
5.3.3	Metrics . . . . .	61
5.4	Results . . . . .	63

5.4.1	Hybrid Streaming Architecture Analysis .....	65
5.4.2	Live Streaming with NDN vs. IP .....	66
5.4.2.1	Higher bitrate & bandwidth utilization by NDN .....	66
5.4.2.2	Comparison of Perceived video quality .....	68
5.4.2.3	Effect of caching .....	71
5.4.2.4	Effect of Oscillation effect on Rebuffering, #QS & Spectrum: Causes & Solution .....	73
5.4.2.5	NDN vs. IP with client-side bottleneck.....	78
5.5	Discussion .....	80
5.5.1	CDNs and NDN .....	80
5.5.2	SDN, NFV, and NDN.....	81
5.6	Conclusion & Future Work.....	84
<b>6.</b>	<b>HANDLING OSCILLATION EFFECT DUE TO CACHING TOWARDS IMPROVED QOE.....</b>	<b>85</b>
6.1	Oscillation effect and its disadvantages .....	86
6.2	Methodology .....	87
6.2.1	Proposed dynamic ABR algorithm to reduce oscillation effect .....	88
6.2.1.1	Identifying oscillation .....	88
6.2.1.2	Reducing oscillations .....	89
6.2.2	$OR_s$ and $OR_d$ : Static and Dynamic Oscillation effect Reduction .....	90
6.3	Evaluation setup .....	92
6.3.1	Experimental Setup .....	92
6.3.2	Design Choice .....	92
6.4	Evaluation Results .....	93
6.4.1	QoE Analysis of clients live video streaming with and without oscillation reduction .....	94
6.4.1.1	QoE With and without Oscillation detection and reduction: .....	95
6.4.1.2	Static vs. Dynamic Oscillation detection and reduction: .....	96

6.4.2	Oscillation reduction in live video streaming clients with variable bandwidth connectivity . . . . .	99
6.4.3	Comparison of QoE with changing network traffic . . . . .	100
6.4.4	Effect of oscillation count and detection window size on overall QoE . . . . .	101
6.4.5	Effect of proposed bitrate qualities on overall QoE . . . . .	103
6.5	Discussion . . . . .	105
6.5.1	Deciding on number of bitrates in Bitrate proposals by $OR_d$ . . . . .	105
6.5.2	Alternative to reduced bitrates for oscillation reduction . . . . .	106
6.6	Conclusion and Future Work . . . . .	107
<b>7.</b>	<b>CONCLUSION &amp; FUTURE SCOPE . . . . .</b>	<b>109</b>
7.1	Conclusion . . . . .	109
7.1.1	Optimizing Flood Forecasting Hydrologic Models . . . . .	109
7.1.2	Optimizing Weather Sensing using NDN . . . . .	110
7.1.3	Optimizing Video Streaming using NDN . . . . .	111
7.1.4	Optimizing ABR streaming with In-network caching . . . . .	111
7.2	Future Scope . . . . .	112
	<b>BIBLIOGRAPHY . . . . .</b>	<b>113</b>

## LIST OF TABLES

Table	Page
3.1 Model Descriptions .....	25
5.1 Average QoE for IP-only and NDN/IP case.....	66
5.2 Average QoE metric results from streaming sessions accross different experiment setup .....	70
5.3 Comparison of the theoretical available average bandwidth for each client with the average playback bitrate observed by each client.....	71
5.4 Avg. QoE and hit-miss ratio at respective caches for Livestreaming on NDN with 10 qualities vs. 4 qualities .....	74
5.5 QoE for NDN and IP clients streaming with full available dataset with intermediate caches present for both IP and NDN .....	81
6.1 Average QoE metric results from streaming sessions across different experiment setup. 20 NDN clients live streaming with 500MB in-network caches .....	96
6.2 Oscillation reduction for clients with variable bandwidth traffic.....	99
6.3 Oscillation reduction with cross traffic. ....	101
6.4 QoE for clients streaming with OR-d with varying oscillation count as threshold and detection window .....	103
6.5 QoE for clients streaming with no oscillation reduction $OR_o$ & dynamic oscillation reduction with varying bitrate proposals for oscillation reduction in the form of $OR_d$ , $OR'_d$ and $OR''_d$ .....	104
6.6 QoE for clients streaming with $OR_d$ with varying bitrate proposal size) .....	106

## LIST OF FIGURES

Figure	Page
3.1 Headwater and Downstream basins (in blue and white respectively) within the black border that represents Dallas-Fort Worth metroplex .....	18
3.2 Division of Headwater basins of our DFW area for 4-fold parallelization. Each group has comparable geographical areas and is processed separately and parallelly. ....	20
3.3 Parallel processing of HWBs for a given range of timesteps ( $t_1 - t_n$ ) 21	
3.4 interleaved processing of HWBs and DSBs for timestep range ( $t_1 - t_n$ ) 22	
3.5 MapReduce solution for parallelized downstream basin processing within interleaved processing .....	24
3.6 Timeline of receiving and processing unit timestep data ( $t_i, t_{i+1}$ and so on) , with a data ingest interval of 2 minutes. ....	27
3.7 CASA Radar images: Top Left: non-tornadic heavy rain ; Top Right: squall line forming; Bottom: widespread stratiform rainfall .....	28
3.8 Comparison of all models based on processing runtime with group-based parallelization on HWBs under varying incoming rainfall data speed. ....	29
3.9 Comparison of all models on processing runtime with full parallelization on HWBs under varying incoming rainfall data speed .....	30
3.10 Explanation of how Model B varies with Model D when range of timestep varies. ....	31
3.11 Summarization of our best case scenario .....	32

3.12	Effect of different data-ingest intervals on runtime of each model (4-fold group-based parallelization) . . . . .	33
3.13	Effect of different data-ingest intervals on runtime (full parallelization) of each model . . . . .	33
3.14	Effect of full parallelization on a 2.00 GHz, 56-core cloudlab machine when all-data present . . . . .	34
3.15	Performance of our model under different rainfall conditions . . . . .	36
4.1	Illustration of the input for the merging algorithm to create a mosaic from data generated by two heterogeneous radars. Top figure shows the preferred sceanrios while the bottom figure shows worst-case scenarios. . . . .	41
4.2	NDN based Interest/Data Exchange . . . . .	44
4.3	Experiment topology. The site names, bandwidth and delays are obtained from the actual CASA deployment. . . . .	46
4.4	TOP: (a) Time required for downloading datasets from radars using NDN. All datasets in this figure are needed for the weather prediction workflow. The observed RTTs from emulation are slightly higher than simulation; MIDDLE:(b) Effect of increased RTT on file download times; BOTTOM:(c) Effect of increased loss on file download times . . . . .	47
4.5	Current method vs NDN based file retrieval. The left bars show the number of files from each individual radar that are included in mosaics on a percentage basis. The blue bars show the cumulative files used for a mosaic. The red bar shows NDN's improvement, always the ideal number of files. One radar was offline and was excluded from the simulation. . . . .	49
4.6	Merged radar data from the same sever weather event, in the case of ideal data ingest (top) and the ingest of too many files (bottom). . . . .	50
5.1	SDN-NFV setup in an intermediate router . . . . .	55
5.2	Topology for our Evaluation setup . . . . .	56
5.3	Execution environment for IP (left) and NDN (right) ABR streaming clients. . . . .	58

5.4	Effect of increasing clients on CPU Load and Rebuffering percentage. ....	59
5.5	Cumulative Distribution Functions for QoE metrics for the case of 4 NDN and 4 IP clients. ....	67
5.6	Cumulative Distribution Functions for QoE metrics of 20 NDN client cases with varying cache sizes. Case A: NDN cache size 0MB; Case B: NDN cache size 250MB; Case C: NDN cache size 500MB. ....	69
5.7	Cumulative Distribution Functions for QoE metrics for the case of 40 NDN clients and 20 IP clients. ....	72
5.8	Distinct qualities requested per DASH video segment by an NDN client when #qualities available at the server are 10 vs. 4. Total DASH segments in our streaming video are 298. Maximum possible unique qualities requested in both cases would be 10 and 4 respectively and minimum would be 1. ....	75
5.9	Cumulative Distribution Functions for QoE metrics for the case of limited playback bitrates. Case A: NDN chooses from 4 qualities & IP from 10. Case B: Both NDN & IP choose from 10 qualities. ....	76
5.10	Cumulative Distribution Functions for QoE metrics with client-side bottleneck. Case A: NDN chooses from 4 qualities & IP from 10. Case B: Both NDN & IP choose from 10 qualities. Case C: Case B with 1% packet loss. ....	79
5.11	Cumulative Distribution Functions for QoE metrics of 20 NDN and 20 IP client cases with 500MB cache setup for both. ....	82
6.1	Cumulative Distribution Functions for QoE metrics for the case of oscillation reductions. $OR_o$ : No oscillation reduction, $OR_s$ : Static oscillation reduction, $OR_d$ : Dynamic oscillation reduction. ....	97
6.2	Client A that reported 117 quality switches and 27% Rebuf %. ....	103
6.3	Client B reported 46 quality switches and 6.25% Rebuf %. ....	103

# CHAPTER 1

## INTRODUCTION

The efficiency of any system or cloud service model can be defined by a combination of metrics such as latency, scalability, robustness, quality of experience, etc. For instance, live streaming supported by a cloud platform would prioritize latency and visual quality as the "liveness" of a good-quality content being delivered to the user is a key requirement. Similarly to optimize any model or system, a combination of metrics needs to be identified that best quantifies the efficiency of the system. Latest system technologies can be utilized to optimize these metrics, which as a result, enhances system performance. It is our goal to improve different use-cases of cloud services such as weather systems with respect to safety and live video-streaming with respect to entertainment. They are common in the sense that they both require minimum delay in delivering information to the end-user. However, these cloud services have their own set of existing challenges and may also differ in other end requirements - while weather systems need to provide accurate and fast-accessible information to the end-user, video streaming systems might prioritize video quality for end-users. In this thesis, we investigated and utilized the benefits of system technologies such as virtualization, parallelization, software defined networking and information centric networking to improve processing runtime and data retrieval from radars in weather forecasting systems and overall quality of experience in video-streaming systems.

## 1.1 Weather Systems

Flooding is one of those environmental calamities that has a significant adverse impact on the human population. To be able to predict and monitor flood events at a high spatiotemporal resolution in real-time, we need weather radars and high-resolution distributed hydrologic modeling systems which require high-performance computing clusters (HPC) for processing high-resolution rainfall data. The dependency on HPC clusters for real-time weather forecasting can sometimes be a resource bottleneck if they are unavailable or unaffordable. Hence, a cheaper and more accessible alternative is required that can also generate close to real-time flood forecasts. Moreover, the remotely placed radars involved in weather sensing and forecasts face significant network connectivity challenges. First, the available bandwidth varies considerably across these radars. Next, during worse weather, these radars produce more data and uncoordinated requests from these radars (e.g., requesting all available files from a radar at once) lead to congestion and delay in data retrieval which, in turn, delays short term weather forecasts. Eventually, data from all radars are combined into a mosaicked product at a merging site before being processed by the weather application. The files from these radars arrive at the merging site at different times due to the different data generation rates of each radar or network conditions. As a result, these merged files are not temporally synchronized and may reduce the quality of the merged product and affect weather prediction. In this dissertation, we address these problems in weather systems to improve end-user experience.

To resolve the first problem of running flood forecasting models independent of HPC clusters, we use cloud resources to process rainfall data since virtual machine (VMs) instances on the cloud are more readily available. We then modify the flood forecasting hydrologic model by parallelly distributing its workflow and test if the parallelization techniques makes predictions more scalable, robust and can be extended in real-time or close to real-time and potentially substitute the ones executed in HPC

clusters. With this modification, we aim to improve the runtime taken by a flood forecasting model to process real-time rainfall data on the cloud. The target weather model we modify in this work is a high-resolution grid-based model known as Hydrology Laboratory Research Distributed Hydrologic model (HL-RDHM), developed by the National Weather Service (NWS) Hydrology Laboratory (HL)[31]. This model is distributed in the sense that it can incorporate both spatially-varied land data as well as high-resolution precipitation information for rainfall-runoff modeling, natural channel routing and flood forecasts. We aim at making this workflow more efficient by incorporating parallelism in it. We test our model’s performance using high-resolution precipitation data collected from the Center for Collaborative Adaptive Sensing of the Atmosphere (CASA) Radar network in the Dallas-Fort Worth metroplex (DFW)[3].

To solve the second problem of improving data quality collected from the radars, we move away from TCP/IP’s push-based model that utilizes a static distribution model in favor of Named Data Networking (NDN) [82]’s pull-based model. Named Data Networking (NDN) is a future Internet architecture instantiation that is content-centric and distributes address-less data based on its name only [83]. In other words, NDN replaces the host-centric approach of the current (TCP/IP-based) Internet with a content-centric approach. Therefore, instead of the radars pushing the files to the merging site for processing as in the current workflow, NDN allows the merging site or data collection facility to directly request necessary files from the radar site in a timely fashion. To enable efficient, NDN-based communication, we develop and utilize a hierarchical naming scheme that explicitly captures radar parameters. Rather than downloading everything that is available from the radars, this naming scheme allows the merging site to specify which exact datasets it requires at a given time. To facilitate this, we develop a signaling mechanism where we piggyback information about new files as they are generated. To the best of our knowledge, this is the first attempt in applying NDN to a real weather sensing infrastructure.

## 1.2 Live video streaming systems

Video streaming continues to be the most popular application in the Internet. For example, currently, 82% of all global traffic is video content [6]. With the advent of applications like Twitch, Facebook Live, live streaming has increased tremendously in popularity occupying upto 17% of the video traffic [75]. Therefore, mechanisms are required that deliver efficient Quality of Experience (QoE) for live video streaming over existing Internet infrastructures. Apart from the traditional TCP/IP, multiple Internet architectures have been proposed for the future Internet [1][2][67]. As mentioned before, Named Data Networking (NDN) [83] is an instantiation of ICN that identifies and serves data by name instead of their location. Recent work [64] has shown that NDN can improve video streaming application performance due to its in-network caching and multi-path forwarding capabilities [64]. Obviously, the in-network caching characteristics of NDN are well-suited for live streaming events like the Superbowl or the FIFA Worldcup final, where the same content is requested by a massive amount of viewers simultaneously (potentially in geographic proximity). Since NDN requires the replacement of the Layer 3 protocol, the immediate widespread adoption of this new approach is difficult. Experiences with the long and painful rollout of IPv6 and some new TCP flavors, which all require changes in the operating systems of virtually all nodes (routers and hosts) in the Internet, have clearly shown how cumbersome this process can be as well.

Motivated by these past experiences, we design, implement, and evaluate an approach that can transparently choose and adapt to multiple Internet architectures (TCP/IP and/or NDN). This requires a portable, flexible and an easy to configure system that can be easily deployed on a variety of hardware platforms and dynamically adapt to application and network demands. To achieve this goal, our approach employs Software-Defined Networking (SDN) and Network Function Virtualization (NFV) [14] for network virtualization. Furthermore, we utilize operating system vir-

tualization (Docker [11], Singularity [32]) to run the video streaming application on a container, bypassing the need for kernel-support in NDN clients. In this dissertation, we present and evaluate our hybrid architecture in comparison to existing traditional IP architecture with respect to live video streaming experience. With detailed analysis of QoE on our setup, we show the benefits of utilizing NDN in this architecture in terms of increased average bitrate and bandwidth utilization. We also identify the shortcomings of using NDN with existing adaptive bitrate streaming algorithms and suggest solutions to overcome them such that live video-streaming clients using NDN always achieve better QoE per client than existing IP.

### 1.3 Dissertation Outline

In **Chapter 2**, we present background concepts crucial to understand the system optimization for each model use-case. Followed by this, we present past related work that forms the basis of design and implementation choices for our optimized systems resulting in its improved performance.

In **Chapter 3**, we present our thesis work on optimizing a flood-forecasting hydrologic model in terms of its processing runtime. We first describe the design and implementation methodology with which we modify an existing hydrologic model, HL-RDHM. A significant increase in resolution results in significant increase in processing time, preventing the computation of the model in real-time. Thus, our focus is on minimizing the time taken by the model to process the rainfall data. The baseline system processes the rainfall over the entire area of interest as a whole. We modify the current workflow of the model by dividing the area of interest based on their geographical characteristics and then parallelizing the workflow between these divided areas. Followed by the methodology details, we present the results of our approach where we receive upto 34% improvement in rainfall processing time. We implement this in a number of ways, explained in further detail in Section 3.1. Finally, we con-

clude this chapter highlighting benefits of our optimized weather-forecasting model and its future scope.

In **Chapter 4**, we present our thesis work of optimizing weather sensing with a more efficient data retrieval mechanism than what currently exists. We develop a name-based protocol which allows us to retrieve weather data deterministically over networks with different connectivity. We integrate NDN-based naming and data retrieval strategy, and demonstrate that NDN’s pull based model enables more accurate weather predictions compared to the current TCP/IP’s push based model. By naming files to represent the actual rotation of the radars (which varies in a heterogeneous set of radars), we allow the consumer to request the exact files it needs for weather prediction, rather than periodically downloading an arbitrary number of files. If necessary, our protocol allows us to explicitly notify the consumer when a new file is available through piggybacking. This chapter introduces our design, explains the namespace and the design rationale behind it, as well as the piggybacking mechanism. We further discuss the experimental setup and demonstrate the improvements that NDN brings to this use case in the results section of this chapter, followed by future directions concluding the work.

In **Chapter 5**, we present our thesis work on optimizing the quality of experience in a cloud-based live video-streaming architecture. First, this chapter proposes a hybrid setup that is flexible to the network as well as the underlying physical layer such that it simultaneously supports multiple Internet architectures for better QoE in an adaptive bitrate streaming (ABR) application. Next, after describing the experimental setup of our new proposed architecture, we present our detailed QoE analysis in the results section of this chapter where we justify using NDN along with our hybrid setup to improve live video streaming experience with minimal end-user involvement. Finally, we conclude the chapter with an overall outlook to our approach and future work.

In **Chapter 6**, we present our dissertation work on identifying the drawbacks of adaptive bitrate live video streaming in a cache-based NDN network and propose solutions for the same. First, we describe the phenomenon called "oscillation effects" which occurs as a result of adaptive video streaming with NDN, and its effect on QoE. Followed by this, we propose a new dynamic algorithm that not only reduces the negative impact of "oscillation effects" on QoE, but can also adapt to changing network conditions. Next, we describe the evaluation setup and design choices followed by evaluating adaptive streaming with our proposed oscillation reduction algorithms compared to no oscillation reduction. We also factor in parameters from our algorithm that have most impact on improving overall QoE. We conclude the chapter with a relative discussion, a summary of our observations and future scope.

Finally In **Chapter 7**, we conclude with a summary of our dissertation as well as the future scope of our work.

## CHAPTER 2

### BACKGROUND AND RELATED WORK

This chapter discusses background and related work with respect to optimization of a flood-forecasting weather model, weather sensing and live-video streaming systems on the cloud. In this direction, we first introduce the cloud testbed and weather radar network used to setup our experiments. Followed by this, we provide brief background on a flood-forecasting model, current weather sensing workflow and video-streaming techniques. Finally, we describe system technologies like Named Data Networking and related work that motivated us to incorporate it in our systems for improvement.

#### **2.1 CloudLab**

Cloudlab is a testbed [15] which offers bare metal servers with substantial compute, storage and networking resources. This testbed provides flexibility to configure individual nodes exactly to our needs. CloudLab supports reproducibility, allowing us to share the complete setup and make the execution and outcome of our evaluation repeatable by other researchers. Finally, Cloudlab resources are much more readily available to users as compared to HPC clusters. We implement and run our weather forecasting system and live video-streaming system on Cloudlab testbed.

For our data retrieval improvement work with radars, we deployed our emulation setup on Google cloud platform [19].

## 2.2 CASA

The NSF Engineering Research Center for Collaborative Adaptive Sensing of the Atmosphere (CASA) was formed to study the lower atmosphere with networks of high resolution Doppler weather radars with the goal to improve severe weather awareness [44].

In our thesis work towards optimizing a hydrologic model, the CASA Radar network in the Dallas-Fort Worth metroplex (DFW)[3] provides meteorological and hydrologic products in support of severe weather and flash flood monitoring and prediction. Our study area is the DFW region in north Texas which is the fourth largest metropolis in the United States by population. For our thesis work to improve data retrieval from heterogeneous radars, we used the network of seven X-band radars in the DFW Metroplex [4] operated by CASA.

The volumetric data produced by these continuously operating remote sensors must be distributed to processing servers quickly and efficiently, such that analysis can occur in near real time for the sake of warning the public of fast developing threats such as tornadoes and high winds. The networked radar concept requires that asynchronous raw data from multiple sources is blended together to create value-added meteorological products. At any given time the characteristics of the ongoing weather regime determine the necessity and priority of certain products. For example, a hail detection algorithm takes on high importance only when strong thunderstorms are ongoing, whereas forecasting algorithms may be of more importance well in advance of such severe weather events and perhaps somewhat less so once the event has started. These radars have mechanically steered antennas and are tasked to perform surveillance scans (antenna rotates a full  $360^\circ$ ) at different elevation angles. Since the radars are not identical, their form of producing atmospheric data is slightly different, and data generation is not synchronized across all radars (e.g., the execution of a surveil-

lance scan does not take the same amount of time for all radars). In our dissertation, we present how data retrieval from these heterogeneous radars can be improved.

### 2.3 Flood-forecasting hydrologic Model

Flooding, in particular, poses one of the most significant natural hazards in urban areas and these occurrences of heavy-to-extreme precipitation are increasing in many parts of the world [5]. Hydrologic distributed model help predict flood forecasts by processing historical rainfall information along with other atmospheric and geographic knowledge (soil moisture, water-level observations, catchment areas, etc.) These flood-forecast hydrologic models usually require HPC clusters for processing high-resolution rainfall data. However, in this thesis, we try to show that with the some modifications to the hydrologic model including parallelization, cloud resources can be a possible alternative to HPC clusters. The challenges of real-time stream-flow monitoring and predicting is discussed in [61]. This work requires cutting-edge satellite data and multiple hydrologic models (calibrated as per the location) to generate probabilistic forecasts with a lead time of 9 days. Moreover, the importance of cloud computing in increasing the effectiveness of weather model predictions is acknowledged in [36][49]. For instance, R.Liu et al. [36] discuss how a web service on the cloud enables hydronomic simulations for multiple weather models. [49] also uses cloud clusters for predicting flood forecasts via uncertainty sampling. Apart from cloud computing, the potential of a parallel programming environment in running sequential or parallel weather models is stressed upon in [39]. The work of Mair et al.[39] presents how model simulation runs can be sped up by applying parallelism. Another work[74] utilizes a MPI parallel framework to analyze and optimize HL-RDHM's sensitivities by reducing the parameters needed for the model's calibration. However, our work makes the operational HL-RDHM model faster by modifying the workflow itself without requiring any computational expertise by any operational per-

sonnel. This motivated us to adopt a parallel programming based design on a cloud system for optimizing the performance of the flood forecast model.

We test our approach on NWS’s hydrologic model called HL-RDHM (Hydrology Laboratory Research Distributed Hydrologic Model) which is used for real-time high-resolution flash flood forecast warning systems. The HL-RDHM model uses a gridded structure for providing efficient remote sensing-based products and atmospheric model outputs. Traditionally, HL-RDHM applications use Hydrologic Rainfall Analysis Project (HRAP) grid cells of size 4 km by 4 km. Although the HL-RDHM model structure is built to support any resolution ranging from 4km (lowest) to 250m (finest) grid size, our focus is on the finest grained resolution, 1/16 HRAP rainfall data (250 m by 250 m grid size) over the Dallas-Fort Worth metroplex to test real-time rainfall processing. HL-RDHM monitors streamflow for flood forecasts by calculating hydrologic parameters such as discharge, runoff from incoming rainfall data and with our approach we improve upon HL-RDHM’s rainfall processing runtime.

## **2.4 Weather sensing**

In this section, we explain current dataflow of CASA radars and their existing challenges.

### **2.4.1 Current Dataflow Paradigm**

The underlying data transport mechanisms used in the CASA network are based on the traditional TCP/IP protocol stack. For example, UCAR’s Local Data Manager (LDM) system [79] is used for event based distribution and analysis of radar data. The event based distribution relies on a sender-driven pub/sub system, whereby data requests from downstream clients are registered with regex-like pattern matching schemes based on expected file naming conventions. Data filenames generally include product or radar name, valid time, and the file format abbreviation as suffix. Client

side data requests include the IP address or DNS name of the upstream server associated with each product pattern and these are contained in a configuration file. The upstream data server must include a corresponding configuration entry allowing the downstream client IP address/DNS name to request such data. Any changes to the configuration files on either client or server side require a program restart to take effect, during which all data ceases to flow for several seconds. Thereafter, any data arriving to or input from the server side matching the client's requested pattern is forwarded to the client. Data flows and the applications using the data are disjointed in this respect. This sender-driven approach is not well suited for mostly data-driven algorithms since a prior knowledge of active algorithms and their data needs is required, and modifications to the data retrieval service is disruptive to the overall system and potentially other users. The paradigm of preconfigured data retrieval on a per machine basis is not well suited for the virtualized, cloud-based, highly adaptive compute resources that are used in the CASA system today [37]. An approach that eliminates the need for a priory knowledge of the applications compute resource and data requirements will benefit current and future CASA data processing workflows.

#### **2.4.2 Asynchronous Data**

The operation of the radars in the CASA network is not synchronized in any respect, however the data they generate still needs to be mosaicked at a central processing location, and there is still a notion of an ideal set of data files that should be ingested for each mosaic. In general, a mosaic is a full volumetric set of data from each radar, as closely linked in time as possible. Radars produce volumes made up of multiple data files at intervals from 50 to 70 seconds. It is less than ideal to not include some of the data files making up a volume in the mosaicking process, but also less than ideal to include data files representing the same portion of the volume at multiple times, which results in a smearing effect. Time based windowing, whereby

an algorithm waits for a specified period of time for data from all radars to arrive, then executes based on the input it received quite often leads to one or the other of these results. Moreover a result we always strive to avoid is to not include any data from a radar in a mosaic, starving regions of the grid of all data and possibly causing users to make ill informed decisions. Therefore the time windows must be kept open longer than the ideal interval, so as to guarantee in all normal network traffic situations that at least one file from each radar shall arrive and be included in the mosaic. A better solution would be to request all data files making up a volume from each radar in an explicit fashion. Current time-based naming schemes can make this problematic to request, therefore in this dissertation, we propose an alternative scheme indicative of the periodic volumetric collections of data files from each radar further discussed in Section 4.2.3.

## 2.5 Adaptive Bitrate Streaming and DASH

Before we discuss improvement in live video streaming, let's discuss existing live video streaming methods and standards. Virtually all commercial streaming services that offer either VoD, live streaming, or both, use adaptive bitrate streaming (ABR). With ABR, the streaming rate, and thus the resulting video quality, can be adapted to the available bandwidth between the video server and the client. One of the most popular realizations of ABR streaming is the MPEG's Dynamic Streaming over HTTP (DASH) standard [70]. Its popularity can mostly be attributed to the facts that *i*) DASH-format videos can be streamed from any kind of HTTP server; *ii*) adaptation logic resides in the client, which makes DASH highly scalable; and *iii*) it is an open standard. In addition to DASH, there exist proprietary ABR implementations like Microsoft's SmoothStreaming [46], Apple's HTTP Live Streaming (HLS) [26], and Adobe's HDS [27] exist. The work presented in this article is based on the open DASH standard.

All ABR approaches mentioned above rely on the same principle, which divides a video into segments (usually between 1 - 10 seconds in duration). Each segment is then encoded in different quality versions, which require different bandwidths to be streamed in real-time from the server to the client. For each video, a media presentation description (MPD) file is created that contains meta-information about the segments and the different qualities they are encoded in. A streaming session starts with the client retrieving the MPD file. In the case of live streaming, a dynamic version of this MPD is required such that the client receives a new version of the MPD file once additional segments of the live video become available at the source. The playback bitrate of the next segment to be retrieved is calculated using the information given by the MPD file, the segment download rate determined at the client, as well as the actual fill of the video player buffer. As opposed to preprocessing the entire video beforehand in video-on-demand services, only small chunks of the newly arriving feed can be segmented and encoded in different qualities in the case of live streaming.

In this thesis, we leverage the interplay of ABR and Named Data Networking to improve live video-streaming experience and discuss existing drawbacks and suggested ways to overcome them.

## 2.6 NDN

Named Data Networking (NDN) [82] has gained traction in recent years due to its hop-by-hop connectivity based on content instead of finite address-based communication amongst hosts. This new approach of using uniquely named content as the core Internet principle makes communication independent of location, application, and means of transportation. Even though it represents an entirely new architecture, NDN preserves the Internet's hourglass architecture, making it compatible to run over and along with existing IP. NDN is particularly beneficial while considering large-scale data dissemination instead of IP that faces challenges when it comes to

fixed point-to-point data delivery. Moreover, NDN’s automatic in-network caching imposes a huge advantage over IP when retrieving a large amount of data or fast retransmissions in case of network failures.

NDN uses a pull based model rather than IP’s push based model. This gives the control of content retrieval to the consumer rather than the producer. Consequently, consumers can decide when and how to retrieve content from the producer. NDN communication has two components: *Interest* and *Data* packets. Interest packets are requests for content sent by a consumer to fetch a data packet generated by the producer in return. All the communication is secure as all the data packets are signed. Each NDN node has a stateful forwarding plane that consists of a Content Store (CS), a Pending Interest Table (PIT), and a Forwarding Strategy that includes the Forwarding Information Base (FIB). If the *Data* for an *Interest* packet has been served before, it will be stored in the CS and directly served from the cache. Otherwise, the Interest would be forwarded to the next suitable forwarder. Additionally, in-network caches lead to better scalability, robustness, and faster failure recovery. This motivated us to utilize NDN in modifying the data retrieval process from weather radars and in live video-streaming systems, briefly discussed in the next section 2.6.1 and more extensively in chapters 4 and 5.

### 2.6.1 Related Work

**Weather Sensing:** Distributed science use cases naturally fit into NDN’s name based paradigm. NDN only encourages hierarchical, human-readable, and semantically meaningful names. A previous work [53] shows that using a name-based system to publish, discover, and retrieve these distributed large datasets can reduce data management complexity [16] and speed up content delivery. Additionally, it was found that these communities either utilize hierarchical and semantically meaningful names or these names are easily translatable into NDN names [69]. To the best of

our knowledge, this is the first time NDN is being incorporated to improve a real weather sensing infrastructure. Hence, in this thesis work, we develop a name-based protocol that allows us to retrieve weather data deterministically over networks with different connectivity.

**Live video-streaming:** Several papers [47, 34, 62, 63] motivate the idea behind using the NDN protocol for live video streaming. Mohammed et al. [47] and Li et al. [34] realize NDN’s superiority as compared to IP in live-streaming over wireless networks using WiFi direct and WiFi’s built-in broadcast mechanism, respectively. Samain et al. [64] and Rainer et al. [58] make similar comparison of the dynamic adaptive streaming performance on IP over NDN as shown in [73]. While Samain et al. [64] experiments with different modes of NDN under wireless loss detection recovery and load balancing state, our work aims to provide the best user viewing experience leveraging native NDN. Although on a different hybrid setup that runs both the protocol stacks simultaneously, we also compare the performance metrics of IP and NDN with respect to video streaming. Our main motivation for this thesis work is to use this comparison to build an architecture directed towards a novel hybrid video-streaming experience.

## CHAPTER 3

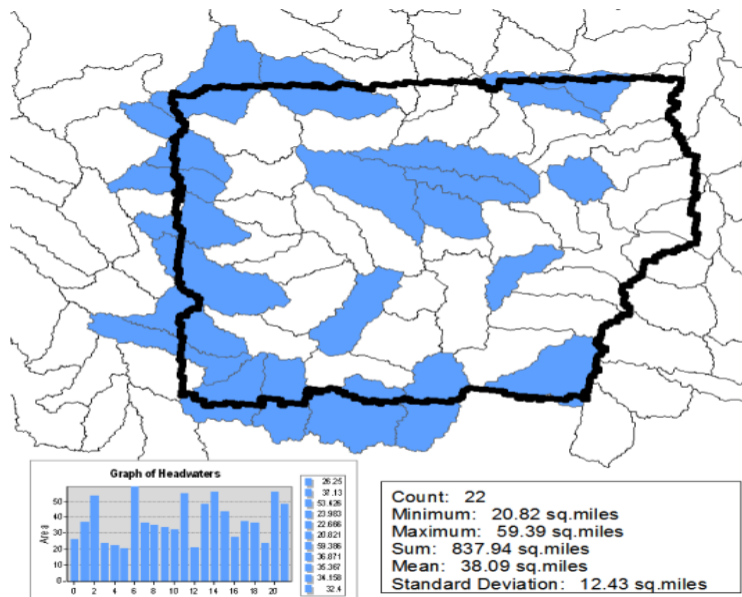
### OPTIMIZING FLOOD FORECASTING ON THE CLOUD

Flooding poses serious hazards to the urban areas. Although hydrologic models use high-performance computing for flood-forecasting, dependence on such a capability poses a possible resource bottleneck for real-time forecasting if unavailable or unaffordable. Hence, in this thesis work we move to more readily available cloud virtual machines to run hydrologic models. We then proceed to improve the model's processing runtime using parallelization approaches implemented by single-host python multithreading. Currently, the HL-RDHM runs in real time on an Intel Xeon CPU E5620 @2.40 GHz 4 CPU-core Linux computer over the DFW domain at 500 m<sup>-1</sup> min resolution, but in this work, we assess the reduction in runtime attainable from different parallel computing approaches toward real-time operation at a higher 250 m<sup>-1</sup> min resolution. This work has also been published in the Journal of Hydroinformatics [23] and can be referred for more details in the domain of hydrology.

#### **3.1 Hydrology characteristics in the DFW region**

Before we describe our parallelization approach on the model, we first give an overview the DFW area. The traditional setup runs the HL-RDHM model on the entire DFW region (bordered by a thick black line) as a whole. This model takes rainfall data as input and processes this information to output discharge and runoff timeseries which is used to calculate the possibility of a flood event. The DFW region as shown in Figure 3.1 has two types of basin distribution: headwater basins (HWBs) and downstream basins (DSBs). Headwater basins (area shaded in blue) have a single

point of outlet and the streamflow is in the direction of headwater to downstream (area shaded in white) basins. For our approach, we divide a model run into the



**Figure 3.1.** Headwater and Downstream basins (in blue and white respectively) within the black border that represents Dallas-Fort Worth metroplex

headwater basin (HWB) and downstream basin (DSB) processes instead of running the model over the entire domain sequentially, as is currently the case. Our hypothesis is that the processing time of the HL-RDHM model on the DFW area can be reduced if instead of processing the rainfall data over the whole region, the computation can be divided into the processing of rainfall data for HWBs and DSBs separately. This separation process can be backed up by hydrologic fundamentals. Being at a higher elevation, the headwater basins receive all of the rainfall first. The rainfall then passes to the downstream basins through the single point outlet at headwater basins. Thus, our idea was to test if separating rainfall processes at these two types of basins would reduce the overall computation time. The blue colored areas are HWBs that process the rainfall data first resulting in the time series output (discharge or surface-runoff), that is then fed to the DSBs represented by white areas within the thick

black line. Thus, streamflow processing over HWBs and DSBs can be separated and parallelized. All of these processes are carried out either along a range of timestep or a single timestep. Here, timestep refers to a one-minute time interval and a range of timesteps is consecutive series of one or more of such one-minute intervals. For instance, 2:00-2:01 pm has a single unit timestep whereas 2:00-2:05 pm has 3 unit timesteps (2:00-2:01 pm, 2:02-2:03 pm and 2:04-2:05 pm). Timestep of rainfall data as an input to HL-RDHM is further explained in detail in Section 3.3.

### 3.2 Methodology

Since the HL-RDHM model is currently setup to process entire DFW region, we had to modify its workflow such that this model can process headwater and downstream basins separately. This workflow is described in Algorithm 1 where processing the entire area of interest is replaced by processing the headwater basins first, followed by collecting its output and single-point outlet information to process the rest of the downstream basins.

---

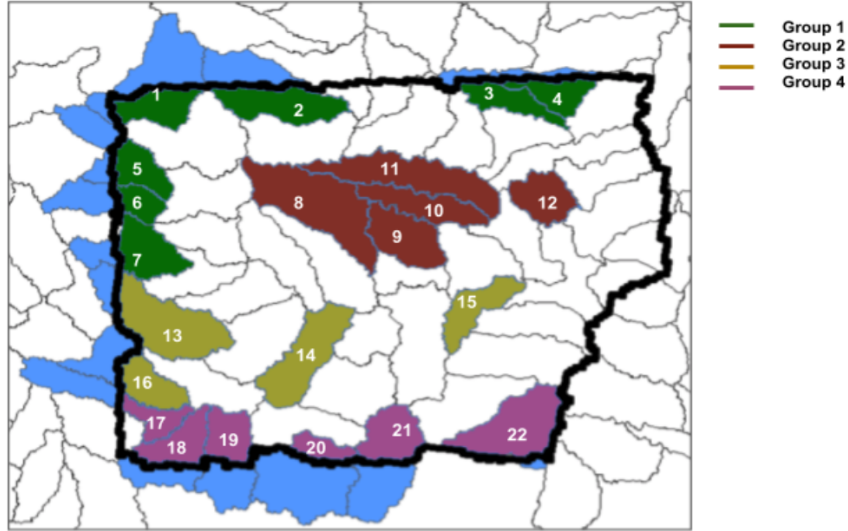
**Algorithm 1** Segregating whole area into headwater and downstream basins

---

- 1: Get the co-ordinates for the headwater basins from the whole area of interest.
  - 2:  $HW \leftarrow$  Set of headwater basins
  - 3: Get discharge and runoff timeseries for these area at the outlet points of these headwater basins by running the HL-RDHM model on HW.
  - 4:  $DS \leftarrow$  Single unit of all Downstream basins
  - 5: To feed the runoff timeseries into the DS as a whole, we get the HRAP co-ordinates of the headwater outlets and modify the input card for Downstream basin such that it receives the output generated by HW.
- 

Now that we have described how the headwater and downstream basins be separately processed by the model, we further parallelize the workflow in two ways: parallel processing of HWBs and interleaved processing of HWBs and DSBs. Further, we vary the degree of parallelization in headwater basins for both these approaches. Firstly, we implement full-parallelization where each headwater basin is processed by a single thread and secondly, all the headwater basins are divided into 4 groups and

each group is processed by a single thread. Our division process is shown in some detail in Fig. 3.2 where each group is denoted by a different color. It should be noted that each group has comparable cumulative geographical area.



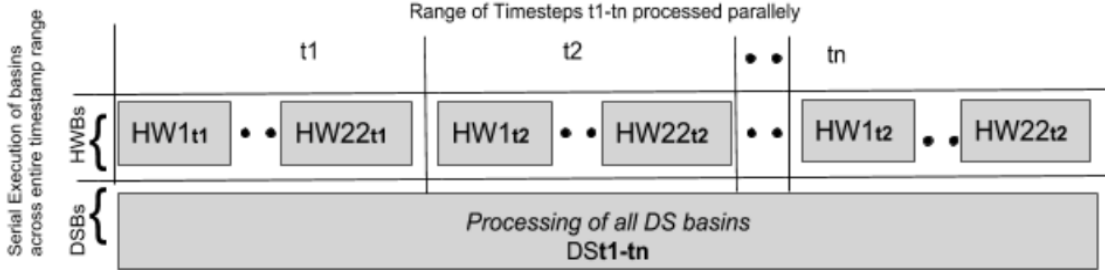
**Figure 3.2.** Division of Headwater basins of our DFW area for 4-fold parallelization. Each group has comparable geographical areas and is processed separately and parallelly.

The process of parallel and interleaved processing of the basins with each case implementing both full and group-based parallelization on headwater basins is further detailed in Algorithm 2,3,4 and 5). The definition and implementation for parallel processing of HWBs and interleaved processing of HWBs and DSBs is described in the next sections 3.2.1 and 3.2.2.

### 3.2.1 Parallel Processing of headwater basins

Although the HWB and DSB are segregated in this model, the distributed computing component is only effective in HWBs in this model. In a given range of timesteps, a single thread runs the rainfall processing for a single timestep, where in each timestep, a single thread processes a single or a group of headwater basins. In other words, the parallelization extended to the HWBs in this approach is nested, as

they are processed parallelly per timestep where each basin (or a group of basins) is again being processed parallelly. This is followed by a serial single thread execution of all the downstream basins. Input for the DSBs are based on the output from HWBs for the entire time range. This approach (which is implemented in and as model B in



**Figure 3.3.** Parallel processing of HWBs for a given range of timesteps ( $t_1 - t_n$ )

section 3.3 and 3.4) is shown in Fig. 3.3. It should be noted that, in this approach, HWBs and DSBs are processed sequentially. The algorithm for this model, with group-based or full parallelization of headwater basins, is described in Algorithm 2 and 3, respectively. Algorithm 2 separates the HWBs into a fixed number of groups

---

**Algorithm 2** Parallel Processing of headwater basins (group-based parallelization)

---

- 1:  $HW \leftarrow$  Set of headwater basins
  - 2:  $DS \leftarrow$  Single unit of all Downstream basins
  - 3:  $ts \leftarrow$  timestamp
  - 4:  $G \leftarrow$  groups of headwater basins
  - 5: Divide set of all HW into  $G$  groups where each subgroup has comparable geographical area as in Fig. 3.2.
  - 6: **for**  $i \in \{1, \dots, ts\}$  **do**
  - 7:   **for**  $g \in \{1, \dots, G\}$  **do**
  - 8:     Run each group of headwater  $HW_g^i \in HW$ ,
  - 9:     parallelly on  $G$  threads for given timestamp  $i$
  - 10:   **end for**
  - 11: **end for**
  - 12: Run  $DS$  for entire timestamp  $ts$
- 

(in our case 4) and each of the headwater subgroups is processed by a single thread. It should be noted that in this case, we divided the DFW headwater basins into 4

---

**Algorithm 3** Parallel Processing of headwater basins (full-parallelization)
 

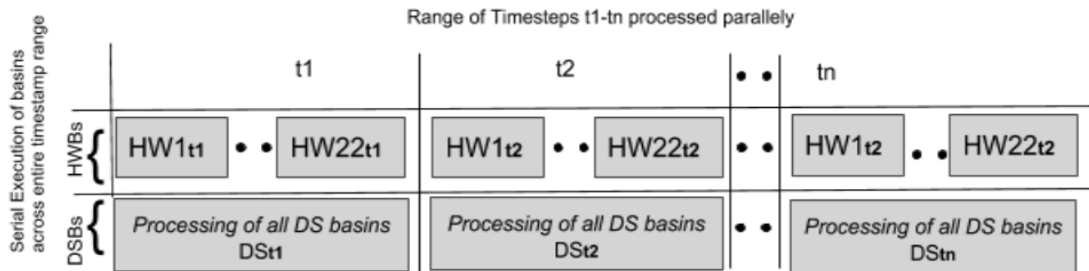
---

- 1:  $HW \leftarrow$  Set of headwater basins
  - 2:  $DS \leftarrow$  Single unit of all Downstream basins
  - 3:  $ts \leftarrow$  timestamp
  - 4:  $T \leftarrow$  size of HW
  - 5: **for**  $i \in \{1, \dots, ts\}$  **do**
  - 6:   **for**  $t \in \{1, \dots, T\}$  **do**
  - 7:     Run each  $HW_t^i \in HW$ , parallelly on T threads
  - 8:     for given timestamp i
  - 9:   **end for**
  - 10: **end for**
  - 11: Run  $DS$  for entire timestamp  $ts$
- 

groups based on geographical proximity and comparable cumulative area. For any other region, this number could be different. In Algorithm 3, each headwater basin is executed by a single thread.

### 3.2.2 Interleaved processing of headwater and downstream basins

The distributed computing component is effective in both the headwater basins and downstream basins for this model. The headwater basins exhibit nested parallelization similar to that described in section 3.2.1. In a given range of timesteps, HWBs and DSBs for each timestep are processed parallelly and within a single timestep, each headwater basin is again processed parallelly (group-based or full-parallelization). This approach is presented in Fig. 3.4 and defined by algorithms 4 and 5.



**Figure 3.4.** interleaved processing of HWBs and DSBs for timestep range ( $t_1 - t_n$ )

---

**Algorithm 4** interleaved Processing of headwater and downstream basins (some form of-parallelization)

---

```
1:  $HW \leftarrow$  Set of headwater basins
2:  $DS \leftarrow$  Single unit of all Downstream basins
3:  $ts \leftarrow$  timestamp
4:  $G \leftarrow$  groups of headwater basins
5: Divide set of all HW into G groups where each subgroup has comparable geographical area as in Fig.3.2
6: for  $i \in \{1, \dots, ts\}$  do
7:   for  $g \in \{1, \dots, G\}$  do
8:     Run each group of headwater  $HW_g^i \in HW$ , parallelly
9:     on G threads for given timestamp i
10:  end for
11:  Run  $DS^i$  for given timestamp i
12: end for
```

---

We have implemented this interleaved processing approach in models C and D. These models are similar in their algorithm and the implementation logic, but vary in the way shared resources are handled in the multi-threaded environment. The HL-RDHM model uses the same resource for processing these different timestep data parallelly on multiple threads. To deal with this resource contention, we use thread-locking mechanism in model C and a resource copy mechanism in model D. By using a thread-locking mechanism, the HWB parallelization benefits are somewhat diminished resulting in higher processing time. Although resource copy mechanism in model D gets rid of this drawback, it causes higher memory usage due to multiple copies per timestep, but it is negligible for our experimental purposes.

This could also be alternatively realized in a distributed fashion utilizing MapReduce [25] as shown in the figure below (Fig. 3.5). Firstly for a given timestep, RDHM parallelly processes each headwater basin, which in itself could be represented by a MapReduce solution, where each headwater basin processing is parallelized. However, due to the hydrological dependencies between headwater and downstream basins, the processing of downstream basins for a unit timestep is slightly more complex as a MapReduce solution, hence discussed in detail in Fig. 3.5. The Figure shows that

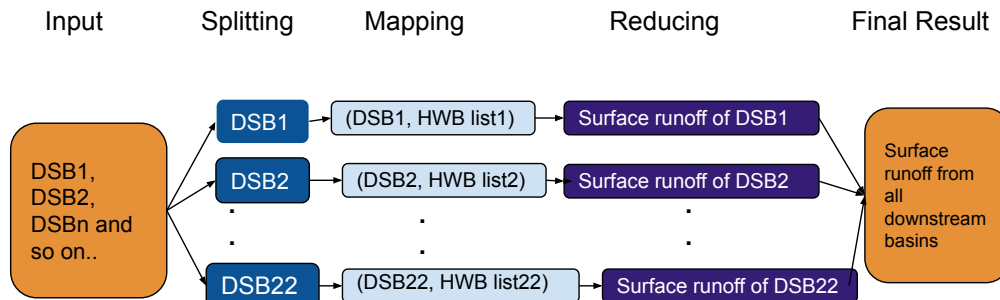
---

**Algorithm 5** interleaved Processing of headwater and downstream basins (full-parallelization)

---

- 1:  $HW \leftarrow$  Set of headwater basins
  - 2:  $DS \leftarrow$  Single unit of all Downstream basins
  - 3:  $ts \leftarrow$  timestamp
  - 4:  $T \leftarrow$  size of HW
  - 5: **for**  $i \in \{1, \dots, ts\}$  **do**
  - 6:   **for**  $t \in \{1, \dots, T\}$  **do**
  - 7:     Run each  $HW_t^i \in HW$ , parallelly on T threads
  - 8:     for given timestamp i
  - 9:   **end for**
  - 10: Run  $DS^i$  for given timestamp i
  - 11: **end for**
- 

for a MapReduce solution to calculate the final RDHM output, first all downstream basins can be split or separated. Next, in the mapping phase, all the downstream basins are mapped to the list of headwater basins, whose output serves as input to the corresponding downstream basin. In other words, the key/value pairs for the map phase are downstream basin and a list of headwater basins its dependent on, respectively. For example, downstream basin DSB1 can be processed only after all the headwater basins in HWB list 1 are processed. Also, the output of HWB list 1 is crucial to processing DSB1 by the RDHM model. Next, in the reduction phase, surface runoff for each downstream basin is computed which is eventually concatenated as the final surface runoff any given unit timestep.



**Figure 3.5.** MapReduce solution for parallelized downstream basin processing within interleaved processing

### 3.3 Experimental Setup

In this section, we describe our experimental setup used to evaluate the approaches introduced in Sections 3.2.1 and 3.2.2 followed by their implementation in models B, C and D. Each of these models denote a modified workflow of the baseline model. We then compare our models to the baseline model with respect to their rainfall processing runtime for a given range of time. Further, we describe the experiments carried out to study the efficiency of our proposed models.

The following data and computing resources were used for our experiments:

- We consider 5-minute timestep rainfall data (generated from CASA QPE radar).
- The resolution of the data is 1/16 HRAP (250m by 250m).
- We use Cloudlab linux x86\_64 VMs with 56 processors (Intel(R) Xeon(R) CPU E5-2683 v3 2.00GHz). (At least 8 or more processors are recommended)

The models being tested and compared have been categorized in Table 3.1.

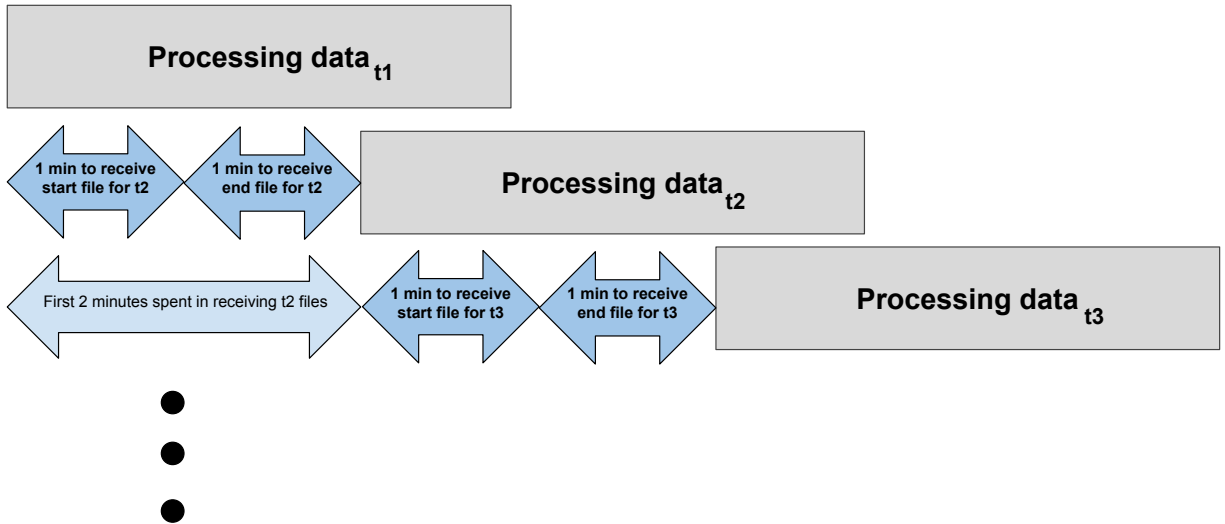
2black!80!yellow!50black!70!yellow!40

Models	Description
Baseline /Model A	This our Baseline model. Here, the HL-RDHM model runs on the Dallas-Fort Worth area as a whole and no division into HWBs and DSBs processes are done. There is no parallelization involved per timestep or in HWB processing.
Model B	This model implements parallel processing of HWBs. This process runs HWBs and DSBs serially over requested timestep. (Section 3.2.1 )
Model C	This model implements interleaved processing of HWBs and DSBs per timestep unit using thread-locking mechanism. (Section 3.2.2)
Model D	This model implements interleaved processing of HWBs and DSBs per timestep unit using resource copy mechanism avoiding thread-locking overhead. (Section 3.2.2)

**Table 3.1.** Model Descriptions

To evaluate the performance of our cloud-based approach, we use Cloudfab resources mentioned above that are readily available for any user. The original HL-RDHM model is available online that serves as the baseline. We modify the existing HL-RDHM model (as described in section 3.1) such that the HWBs and DSBs can be processed separately for models B, C and D. The code for our model B, C and D is available online [7]. To run each of our models on the cloud, all we have to do is transfer our data and model implementation to any such Cloudfab resource. Once the data and models are available on the cloud, we perform a series of four experiments :

- In the first one, we compare the model performances where ingest data is arriving at different intervals. For a 5-min data, we compare the rainfall processing time taken by each model when all data has arrived, or is arriving in real-time in the intervals of 1 or 2 minutes.
- For the second experiment, we study the effect of data ingest speed on rainfall processing times and test what time interval in between the data ingest is most feasible. Before we explain data ingest intervals, let us first introduce how we define timesteps for this experiment. A timestep range is nothing but the start and the end minute for which we want to process the precipitation data. For a 5-minute timestep range, say, 8:00 pm to 8:05 pm, we define a unit timestep as timestep range that is one minute apart, i.e. 8:00 pm to 8:01 pm, 8:02 pm to 8:03 pm and so on. In our case, each minute’s data is contained in a single file. The HL-RDHM model needs a start and end time for processing the rainfall data. Thus, a unit timestep input requires two data files denoting each minute. Coming back to data ingest intervals, we define it as the time period of receiving unit timestep data. For example, for a data ingest interval of 2 minutes, a unit timestep data arrives every 2 minutes (i.e 2 mins to download to two files denoting the start and end minute). The same is explained in Fig. 3.6. Here,  $t_i$  denotes processing data for unit  $i^{th}$  timestep.

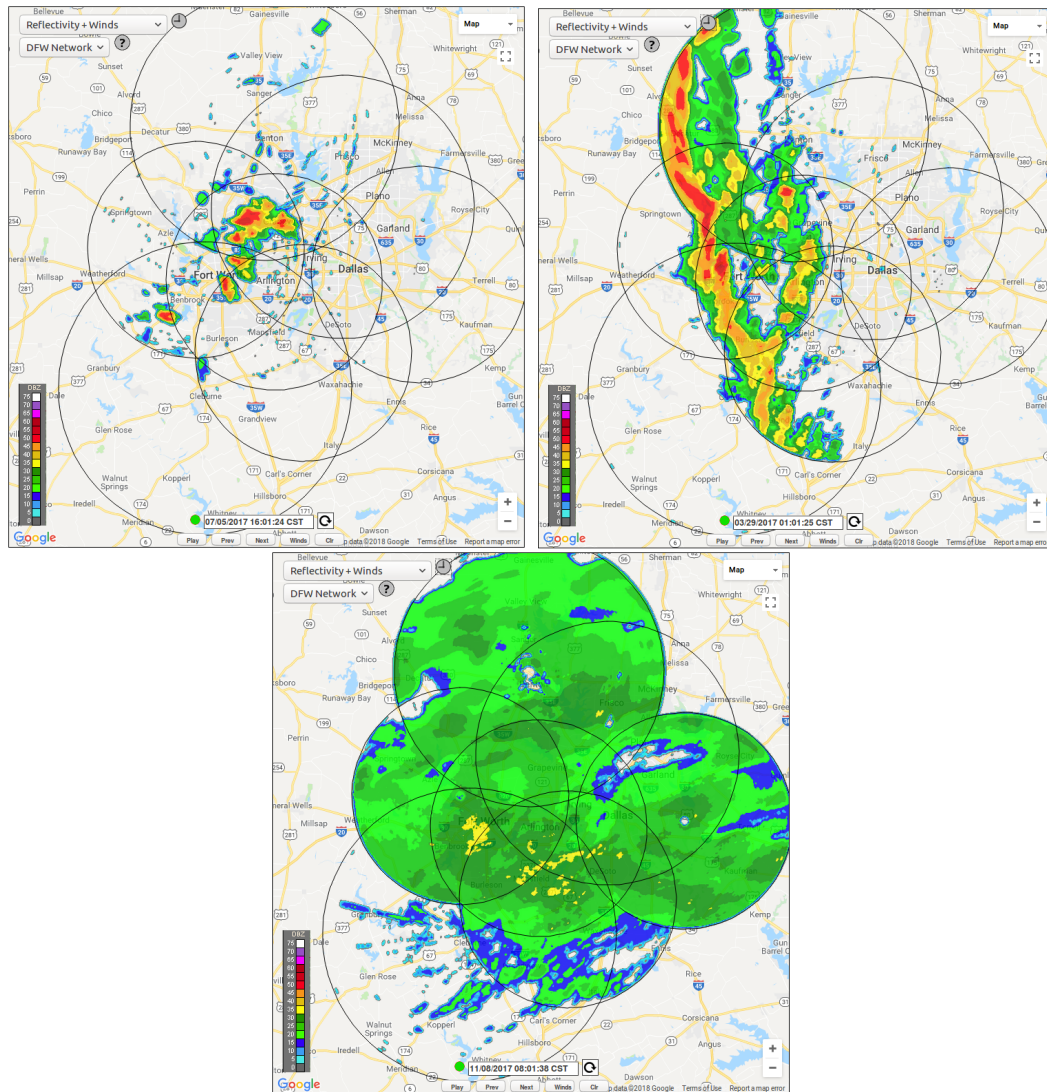


**Figure 3.6.** Timeline of receiving and processing unit timestep data ( $t_i$ ,  $t_{i+1}$  and so on) , with a data ingest interval of 2 minutes.

- For our third experiment, we compare the effect of group-based and full parallelization of HWBs on rainfall processing. We have previously discussed these two degrees of parallelization when HWBs are being processed for a single timestep in Section 3.1. We study the benefits of full-parallelization of the HWBs over a lower degree of parallelization (group-based).
- For our final experiment, we test the performance of our models under varying rainfall conditions. The rainfall cases often cannot be classified quite so easily as they evolve rapidly. Yet, to test our proposed models for robustness, we handpicked few CASA radar data-cases (see Fig. 3.7) that exhibit the following rainfall characteristics:

1. *Non-tornadic flash flood*: On top Left, Between 2200-2300 UTC on July 5, 2017, heavy rainfall in the center of the DFW network occurred.

2. *Clearly defined squall line:* On 29 March 2017, we see a classic squall line with small tornadoes where heavier rain between 0700-0800 UTC throughout the larger part of the network is observed as in the top right image.
3. *Widespread lighter stratiform rainfall:* The bottom image shows lighter widespread rainfall over the network as seen around 1400-1500 UTC, 8 Nov 2017.

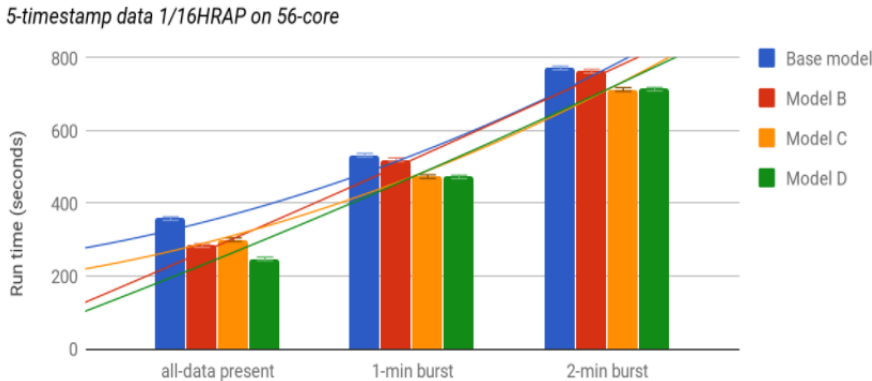


**Figure 3.7.** CASA Radar images: Top Left: non-tornadic heavy rain ; Top Right: squall line forming; Bottom: widespread stratiform rainfall

### 3.4 Results & Discussion

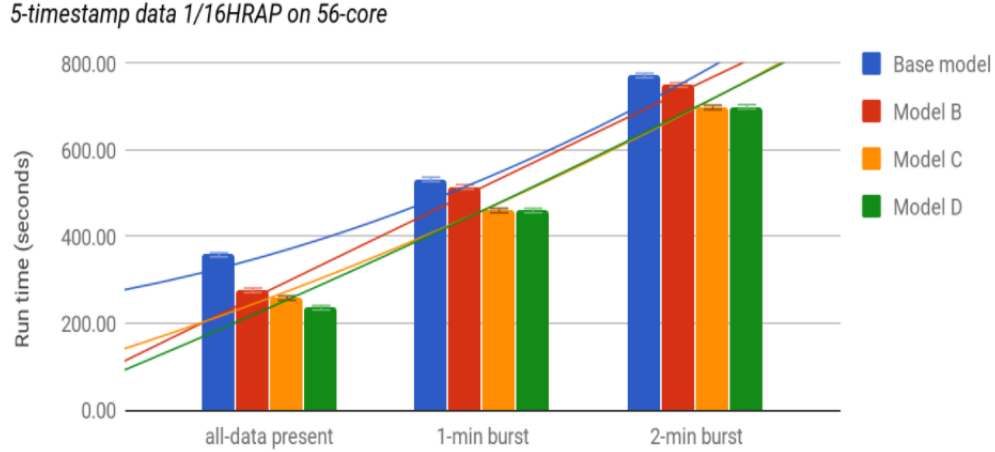
#### 3.4.1 Comparison of models with respect to rainfall processing runtime

We wanted to test if our proposed models take lesser time in processing than the baseline with the rainfall data incoming at different time-intervals. Figures 3.8 and 3.9 compare every model’s performance when 5-minute rainfall data is present, or being received at bursts of every 1 or 2 minutes (simulated as if being received in real-time). While Fig 3.8 shows the results when all models incorporate a fixed degree of group-based parallelization (4-fold where 4 groups of headwater basins are processed parallelly), Fig. 3.9 shows the results with full parallelization implemented on HWBs. Since full-parallelization can be very resource intensive (as number of threads involved increase by the number of headwater basins per timestep), we also wanted to test if introducing a lesser degree of parallelization resulted in similar performance.



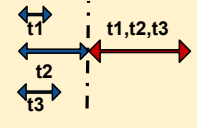
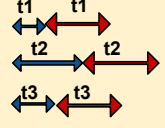

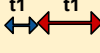
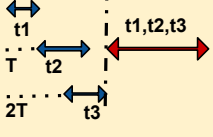
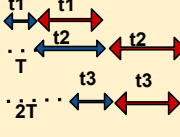
**Figure 3.8.** Comparison of all models based on processing runtime with group-based parallelization on HWBs under varying incoming rainfall data speed.

We test both the scenarios on a 56-core server. As we can observe in Fig. 3.9, the best result in the case of full-parallelization is achieved by model D (see Table 3.1) which outperforms all other models with an improvement of 34% in processing time compared to the baseline (when all of the 5-minute timestep data is present). In case of group-based (4-fold) parallelization with headwater basins, 31% improvement is still observed when model D is compared to the baseline (Fig. 3.8). However, we



**Figure 3.9.** Comparison of all models on processing runtime with full parallelization on HWBs under varying incoming rainfall data speed

observe that with an increasing time interval between the data-ingest, the improvement in the model run-time decreases, irrespective of the model. To understand the relationship between our models, we present the design approach and relationship between model B and D in Fig. 3.10. We had multiple design approaches that we wanted to test, models B and D being the two alternate approaches of how the processing should be distributed amongst HWBs and DSBs. As seen in the first row of Fig. 3.10, the two models B and D behave differently, when all data are present and timestep range is more than one unit. Here, in case of the parallel model (model B), all HWBs for all timesteps are processed before the DSBs are processed. Thus, when all data is present, the unit timestep range that takes longest to process the HWBs, becomes the bottleneck. Whereas in the case of interleaved model (model D), the bottleneck is the joint processing time of HWBs and DSBs for a given timestep. In the case of model D, the processing of DSBs does not have to wait until all HSBs for all timesteps is processed. Hence, the delay in processing time caused due to its bottleneck is lower than the delay caused by the bottleneck of model B and model D always performs better. When a unit timestep range is concerned, both models behave similarly as explained in the second row of Fig. 3.10. When all data is not

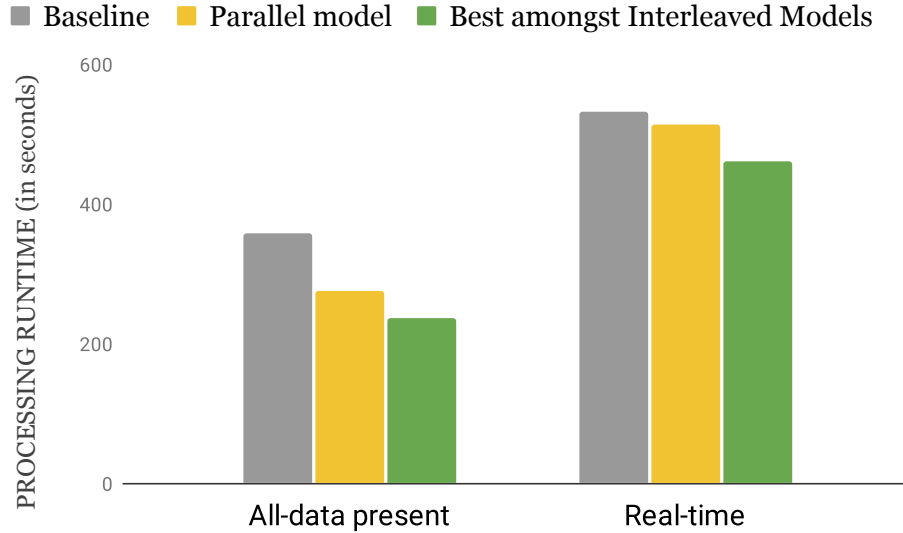
Working of Parallel Model Vs. Interleaved Model			
Blue arrow = time taken for processing the HWBs   red arrow = time taken for processing DSBs   $t_n$ represents n-th timestep being processed			
Data condition	Timestep	Parallel (Model B)	Interleaved (Models C & D)
All data present	> 1 timestep (say,3)		
All data present	= 1 timestep		
Data arriving after every T minutes	> 1 timestep (say,3)		

**Figure 3.10.** Explanation of how Model B varies with Model D when range of timestep varies.

downloaded for a timestep range greater than one unit and arrives at certain intervals, the models again behave differently as shown in the third row of Fig. 3.10. In this case for model B, since the processing of all the HWBs for all timesteps need to complete first, the processing of DSBs takes longer. Again, model D performs much better than model B. Here, the DSB processing for  $t_i$  at  $i^{th}$  timestep starts right after all the HWBs are processed for  $t_i$ , rather than waiting for their processing for  $t_{i+1} - t_n$  to be completed. This is why the performance results of model B and model D differ by a significant margin, as it should conceptually.

As for model C and model D, both use interleaved processing with an similar degree of headwater basin parallelization for each experiment. They only differ in their implementation logic and we observe that model D always performs better when the data is already present, whereas both models C and D have comparable results when the data ingest interval is 1 minute or more. We conclude this section with a summarization of how both our proposed methods improve HL-RDHM's runtime

compared to the baseline, as shown in 3.11. The left hand side of the figure depicts the base case scenario: when all the data is already downloaded or present, our best proposed interleaved model (fully-parallelized model D) takes the least time to process a given rainfall data. The right hand side shows that even when data is arriving in real-time, our proposed models take lesser time to process than the baseline.

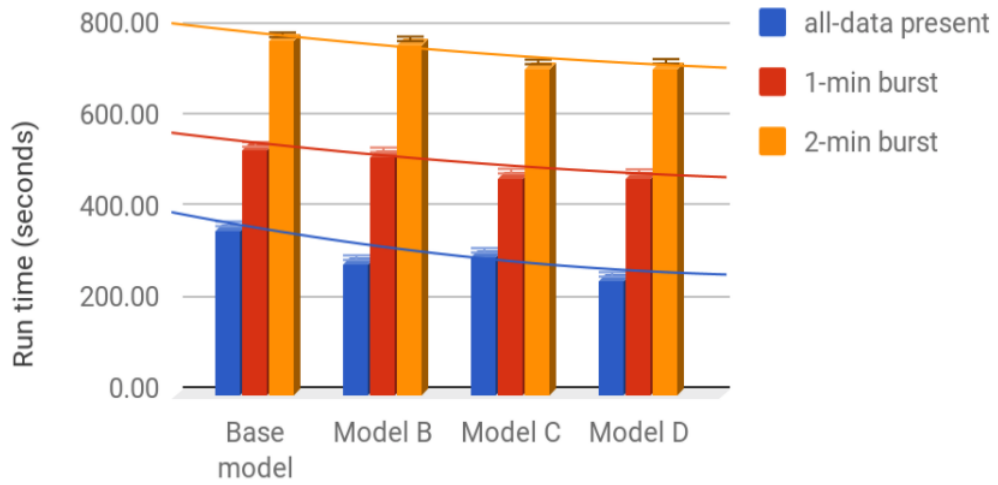


**Figure 3.11.** Summarization of our best case scenario

### 3.4.2 Effect of data-ingest intervals on rainfall processing runtime

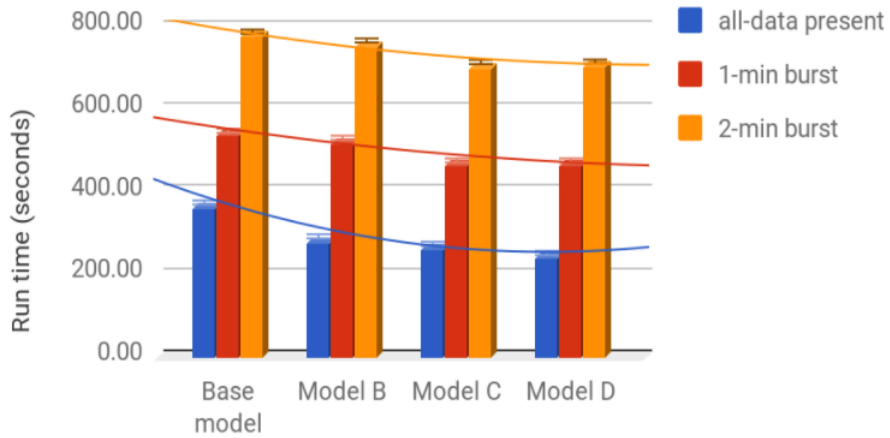
We study the effect of time interval at which the new data for a unit timestep is received every  $T$  time units, on processing rainfall data. As shown in Fig. 3.12 and Fig. 3.13, the best performance is achieved by model D across all cases of delay in data arrival. Ofcourse, we see most improvement in model D when all data has arrived or downloaded but with 1-min and 2-min ingest speed, we wanted to test if our models perform better when rainfall is being processed in real-time. The slope of each line decreasing from blue to yellow in Fig. 3.12 and 3.13 indicate that improvement across the baseline to model D decreases with increase in time-interval between data-ingest. For example, approximately 13% and 17% improvement observed when each

5-timestamp data 1/16HRAP on 56-core

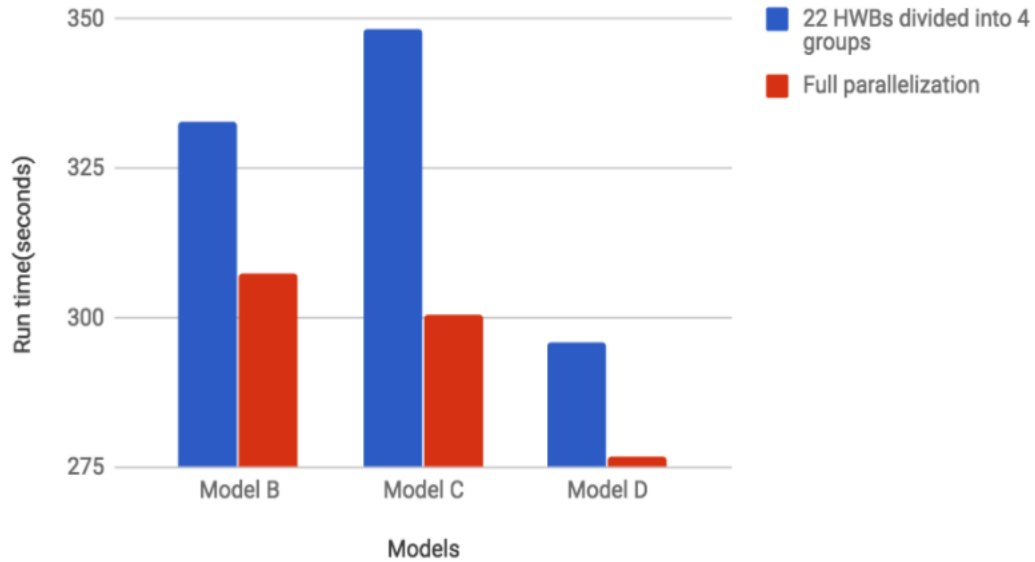


**Figure 3.12.** Effect of different data-ingest intervals on runtime of each model (4-fold group-based parallelization)

5-timestamp data 1/16HRAP on 56-core



**Figure 3.13.** Effect of different data-ingest intervals on runtime (full parallelization) of each model



**Figure 3.14.** Effect of full parallelization on a 2.00 GHz, 56-core cloudlab machine when all-data present

minute’s data comes in every 2 and 1 minutes, respectively. While this still shows improvement over the baseline, it can also support the hypothesis that minimizing the time-interval between rainfall data-ingest will lead to better performance (the idea could be to simultaneously download, store and process the rainfall data than doing all of the processing in real-time).

### 3.4.3 Comparison of different degrees of parallelization in HWB processing with respect to overall rainfall processing runtime

We performed this experiment to compare full-parallelization and group-based parallelization (4-fold) on headwater basins. While full-parallelization gives us best results compared to baseline, lower-degree parallelization can be an alternative when the load on the computing resources crosses a certain threshold. For example: In case of 4-fold, at any time, at most 12 threads were active compared to 66 threads in the full parallelization case. Fig. 3.14 shows model D improves in its computation run time up to approximately 6.5% as an effect of full-parallelization on the HWBs. This

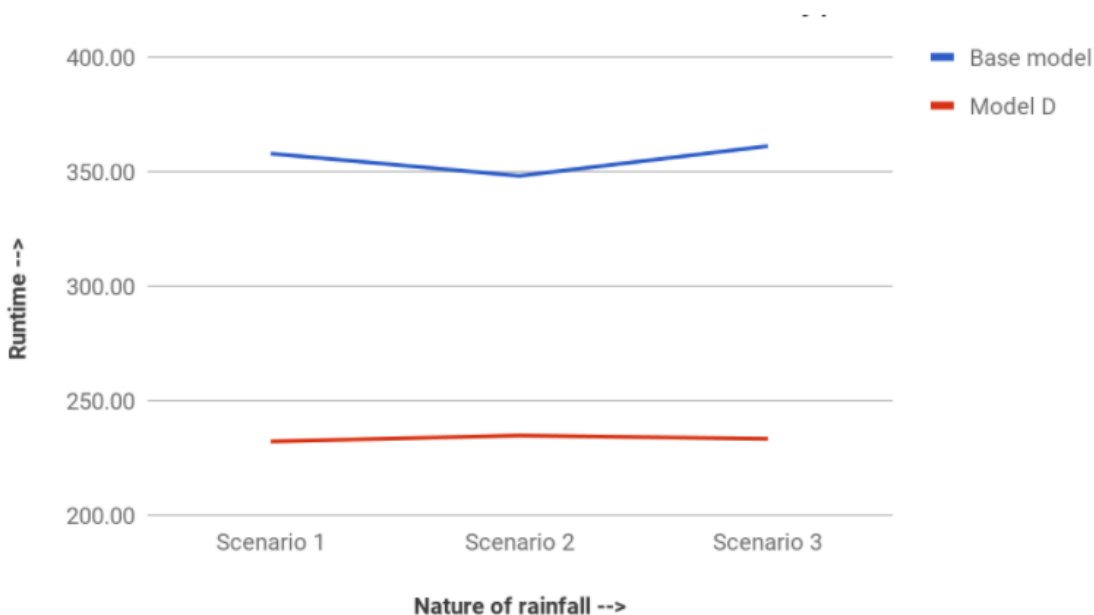
explains model D's 34% improvement in runtime in contrast to 31% when compared to the baseline in figures 3.8 and 3.9). Maximum improvement is seen in case of model C because thread locking overhead (which is absent in other models) is better dealt with full-parallelization. The wait for a group of headwater basin in 4-fold is higher than the wait for a single headwater basin in full-parallelization.

#### 3.4.4 Performance study under different rainfall conditions

So far model D outperformed all other proposed models and the baseline in terms of processing runtime of the hydrologic model. However, we wanted to further test the robustness of our approach and present whether the performance improvement in model D is consistent throughout different precipitation scenarios in DFW. We tested our model in the following scenarios:

- Scenario 1: In the case of a non-tornadic discrete flash flood event in the DFW network, flash floods can be caused by a number of factors but are most often due to extremely heavy rainfall from thunderstorms. Flash flood has begun at 22:00 UTC. Thus for this case, the 5-minute timestep we have considered is 22:20-22:25, Jul 5, 2017.
- Scenario 2: In this case, the weather in DFW has a clearly defined squall line with small tornadoes, heavy wind and rain. A squall line is a line of thunderstorms forming along or on the side of a cooler mass of air replacing the warmer air at ground level. It involves heavy precipitation and lightning. In this scenario, a clear squall line has already formed at 07:00 UTC. Thus for this case, the 5-minute timestep we have considered is 07:20-07:25, March 29, 2017.
- Scenario 3: A typical widespread lighter stratiform rain with low variability is observed in this scenario. Stratiform precipitation occurs when large air masses rise diagonally as larger-scale atmospheric dynamics force them to move over

each other. For this case, we have a uniform widespread rainfall starting at 14:00 UTC, Nov 8, 2017 over the DFW area before the rainfall moves away completely. The 5-minute timestep we have considered here is 14:20-14:25.



**Figure 3.15.** Performance of our model under different rainfall conditions

As Fig. 3.15 shows, throughout all conditions, model D processes rainfall data faster than the baseline HL-RDHM model. The improvement in runtime processing ranges from 33% to 35% approximately, which is consistent with the improvement percentage we have observed until now. Although the base model runtime for all the scenarios is comparable, we see that overall it is slightly higher in scenarios 1 and 3. This is because scenario 1 is a flash flood event with heavy rainfall, whereas 3 has widespread uniform rainfall over the entire network. There is a slight dip in scenario 2 as this case mostly witnesses heavy wind and tornadoes with likely lesser rainfall intensity. We observe comparable runtime in scenarios 3 and 1 because lighter rainfall intensity spread throughout the network and heavy rainfall concentrated at a much smaller part of the network is observed in respective scenarios. Although the runtime varies

with different rainfall conditions, our proposed model D always takes less time (and almost similar runtime no matter the rainfall type) to process the rainfall data than the existing baseline model. We conclude that the improvement in overall runtime processing in our model compared to baseline is robust and consistent with varying rainfall conditions.

### 3.5 Conclusion & Future Work

In this thesis work, we proposed models by modifying the existing workflow of HL-RDHM, a hydrologic model that performs flood forecasts. Our hydrologic model ran on cloud resources and we improved its runtime processing, which in turn improves flood forecasts. The approach of dividing a given region of interest into headwater and downstream basins and parallelizing their workflow made the existing model more efficient. We tested our proposed models on varying conditions of rainfall and can safely conclude that our best model improves rainfall data processing time up to 34% when compared to the baseline system that currently exists. Amongst the two proposed approaches of parallel and interleaved models, the latter outperformed all others, although parallel model still performs better than the baseline. The improvement in performance time decreases with increase in time interval between data ingested. Implementing full parallelization on headwater basins processing gives the best results with minimal overhead, hence all future experiments will have full parallelization on our proposed models. This work of distributing the workflow parallelly can also be extended similarly to other widely used weather models such as WRF-Hydro. In future, the scalability of our approach can also be tested when higher amount of rainfall data is involved. Additionally, we want to test the optimum resolution for which our model performs closest to real-time predictions.

## CHAPTER 4

# IMPROVING WEATHER SENSING WITH NAMED DATA NETWORKING

Weather radars play a significant role in weather forecasting & warning systems. The large amount of data retrieved generated by these radars are requested by a collection facility where it is merged into a mosaicked product before weather applications use it for further processing. For instance, hydrologic models retrieve rainfall data from remote radars for flood forecasts. In our case, we use CASA radars which are currently connected to a data collection facility at National Oceanic and Atmospheric Administration (NOAA) in the DFW area. The remote placement of these radars lead to varying bandwidth capacities: Some radars are connected to high-speed networks with hundreds of Mbps connectivity, some share a 10Mbps connectivity with competing traffic, and some have dedicated, guaranteed 10Mbps connections. This results in limited bandwidth problems in the case of uncoordinated requests. Additionally network congestion, difference in data generation rates of the radars (radars from different vendors have different rotational speeds), different network speeds, lead to data arriving at the collection facility at different times. Data that is not temporally synchronized may reduce the quality of the merged product and affect weather prediction. In this thesis work, we present and evaluate an optimized way of data retrieval process in weather sensing radars using Named Data Networking (NDN) [78]. We have already detailed the current dataflow paradigm and how it is not well suited for large-scale data dissemination in 2.4. Before, we present our approach of improving the data retrieval process, we briefly discuss existing radar networks, the motivation behind using NDN for our end-goal and not TCP/IP.

## 4.1 Weather Sensing with NDN

In contrast to the nationwide NEXRAD radar network operated by the National Weather Service (NWS) which consists of 160 homogeneous radars, the CASA system in DFW consists of a heterogeneous and federated network of radars. While in the NEXRAD system there is only little atmospheric volume that is covered by more than one radar, a significant portion of the coverage area of the CASA system is covered by 2 (or in some cases 3) radars. Thus, merging the individual radar data in a mosaic in a timely manner is important for algorithms that use these merged data for weather product generation. Current radar networks (including NEXRAD) make use of data distribution applications like LDM [79] that are implemented on top of the traditional TCP/IP stack. In federated networks of radars, the radars may communicate with clients through paths consisting of several network hops. These paths may span different service providers, become congested over time, have certain bandwidth constraints, while the overall connectivity between radars and clients may be intermittent. TCP/IP relies on end-to-end connections, without any support from the network infrastructure. HTTP over TCP/IP can provide a pull based mechanism over TCP/IP. However, utilizing HTTP still binds the data with its location (the IP address or domain name). If packets are lost, a real possibility given the remote locations of the radars, the underlying TCP/IP connection must be restarted, throwing away already retrieved data.

Contrary, the work presented in this dissertation presents a new approach based on NDN to make radar data distribution and the execution of weather algorithms more efficient. In NDN, the network can utilize multiple available paths in parallel, adapt to network failures, and facilitate fast retransmissions. At the same time, solutions, such as SSL/TLS, which are widely used in TCP/IP, secure the communication channels, largely depending on the underlying connectivity. On the other hand, NDN makes security a property of the data itself, decoupling it from the underlying connectivity.

Zhang et al. [84] have presented a synchronization protocol that is also designed for the distribution of weather data. While the work presented by Zhang et al. focuses on the distribution of weather products to the end users, our approach focuses on the generation of such products by providing a new approach for transmitting data from the radars to a central compute site where weather product algorithms are running. In addition, our design and proposed naming scheme for data generation and retrieval in rounds is inspired by RoundSync, a protocol for distributed dataset synchronization in NDN [10].

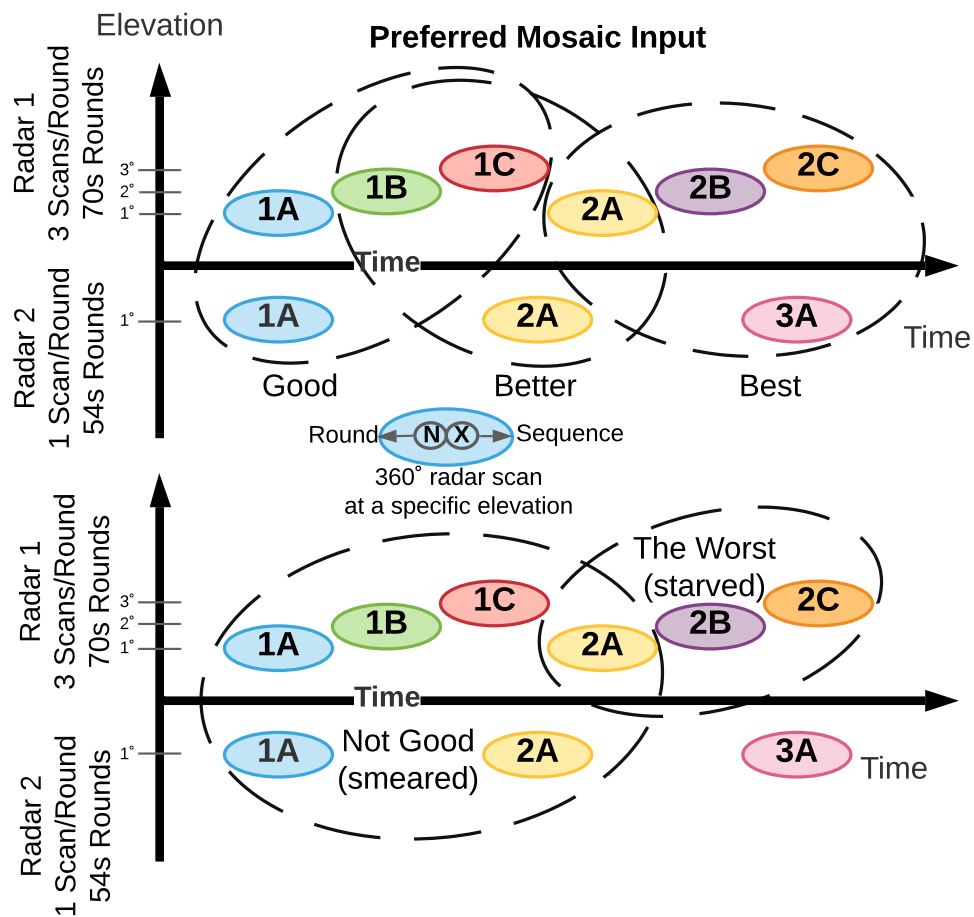
Distributed science use cases naturally fit into NDN’s name based paradigm as NDN only encourages hierarchical, human-readable, and semantically meaningful names. In this work we create a hierarchical naming scheme - /radarName/round/sequence/ (see 4.2.2 for details). The naming scheme captures essential information (radar location, rotational information such as round and sequence number) and allows us to efficiently retrieve data over the network for the workflow. The hierarchical naming scheme also allows for future expansion - we might add timestamp and some location information depending on the workflow. By naming files to represent the actual rotation of the radars, we allow the consumer to request the exact files it needs for weather prediction, rather than periodically downloading an arbitrary number of files. By integrating an NDN-based naming and data retrieval strategy, we demonstrate that NDN’s pull based model enables more accurate weather predictions compared to TCP/IP’s push based model.

## 4.2 Methodology/Design

### 4.2.1 Scenario

The CASA system generates dozens of meteorological products in near real time. Some of these products are generated 24/7/365, others on demand, based on the characteristics of the ongoing weather regime. First order processes include calculating

rainfall rate and accumulations, short term nowcasts (0-30min), hydrometeor classifications (rain/hail/snow), hydrological products (runoff, streamflow), and network wind products. In addition, various post-processing routines operate on the gridded product data, including raster image generation, contouring, format conversions, and end user driven, GIS based data extraction. Timely generation of these products is essential for the warning process and requires significant network and compute resources. For better illustration of the design of our information-centric approach, we



**Figure 4.1.** Illustration of the input for the merging algorithm to create a mosaic from data generated by two heterogeneous radars. Top figure shows the preferred scanrios while the bottom figure shows worst-case scenarios.

focus on the process of mosaicking data from individual radars into a merged grid, as

shown in Fig. 4.1. We have chosen this example since it requires a minimum amount of data from each individual radar to generate an accurate grid. The accuracy of the gridded data impacts the performance of weather algorithms that perform on gridded data. Ideally, data from all radars would arrive simultaneously at the processing node to allow an immediate generation of the merged grid. However, as described before, radars and the access networks connecting them to the Internet are heterogeneous. Additionally, competing background traffic means available bandwidth are different across these radars. This heterogeneity may result in suboptimal merged products - if a less than ideal number of files are included in the merged product, it may result in a worst case scenario (bottom part of Fig. 4.1). When too many files are included in the merged process it may end up “smearing” the merged product, again resulting in a suboptimal product. A suboptimal merged product might impact weather prediction (e.g., decision of a weather forecaster issuing a tornado warning or not) as we will demonstrate in Sect. 4.3.

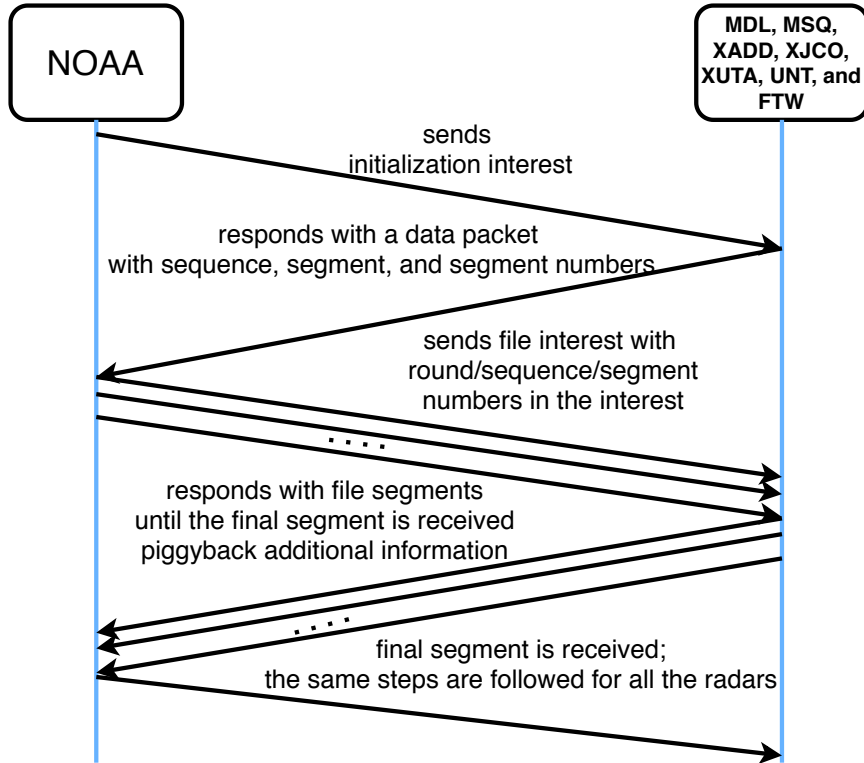
We present a new design for the pull-based retrieval of individual radar data that is based on NDN to allow for a more efficient transport, and better results for downstream meteorological products. We utilize rounds (a complete volume scan by the radar) and sequence numbers (files generated within a round). For example, a particular radar might make a complete 360 degree rotation in a minute. It may also create 3 files for the whole round, they can be sequence 1,2, and 3, each containing approximately 20 seconds worth of data. The use of round and sequence numbers allows the radars to operate independently. For example, each radar may have their own rotational speed (RPM) that does not need to be synchronized. Additionally, this scheme allows the radars to dynamically change the rotation speed or even allow them to go offline without the need for synchronization with the other radars.

### 4.2.2 Data Retrieval in Rounds

Currently, the weather data collected from the radars are named as location-state.YYYYMMDD-HHMMSS.netcdf.gz. One such file on the radar might be named as addison.tx-20200909-000056.netcdf.gz. This naming convention provides sufficient information for subsequent processing - information such as location and time allows the workflow to analyze files from different radar sites, merge all files over a given time window, and run subsequent computations on them. However, the radars have a fixed rotational frequency (e.g., three times every 70 seconds for radar 1 shown in Fig. 4.1), which is not captured by the time-based file names. The generation of files is not connected to a particular round. When using time-based windows, a computational workflow might end up with truncated data (e.g, when a full round of data is not captured in the file) or miss data from a few rounds (e.g., the last rotation was not captured in the netcdf files with the time window).

Moving from implicit naming where the workflows will have to make assumptions or look into the actual data is cumbersome. In this work, we transition from implicit naming (time based) to explicit naming (round based) where each file generated by the radars are named as /data/radar1/\_round=1/\_seq=A (first data file for radar 1 in the example shown in in Fig. 4.1). For workflows that need timestamps, we can simply add the timestamp to the name - /addison/tx/radar1/\_round=15/\_seq=7/YYYYMMDD/HHMMSS.

This naming scheme allows the workflows to look at the names and choose the most relevant data. One example might be when a workflow looks at all rounds collected up to a certain time. The NDN naming and data retrieval makes it much easier to retrieve content based on actual radar rotation, rather than using timestamps. Further, these names allow the clients to predict the exact names of the future data (e.g. /data/radar1/\_round=1/\_seq=C, simplifying the application and workflow logic.



**Figure 4.2.** NDN based Interest/Data Exchange

### 4.2.3 A Naming Scheme for Periodic Radar Data

As introduced in Sect. 2.6, NDN clients indicate a request for data via an Interest packet. In this work, we utilize two types of Interest - initialization Interests and normal Interests. The clients send initialization Interests and query the radars about their current states. On receiving this Interests the radars return their current state of data collection, specifically the current round and the sequence number. The current round can be derived from current time (time since epoch modulo  $n$ ) or based on some other numbering scheme. There are a fixed number of sequences in a round depending on the rotational speed of the radar (3 sequences for radar 1 and 1 sequence for radar 2 in the example shown in Fig. 4.1).

Figure 4.2 shows the actual Interest/Data exchange protocol. The initialization Interests assumes the following format -

/Interest/radar{1..n}/\_current\_round/current\_seq/number\_of\_files. On receiving this Interest, the radar returns the most recent file with the current round number and the current sequence number. The client then may start requesting the files by their actual names, e.g., /data/radar1/\_round=15/\_seq=7. We assume that the radars do not have large storage and only store a small number of files. As a file is downloaded, the radar has the opportunity to piggyback information on the returning data packets, as we discuss in the next section.

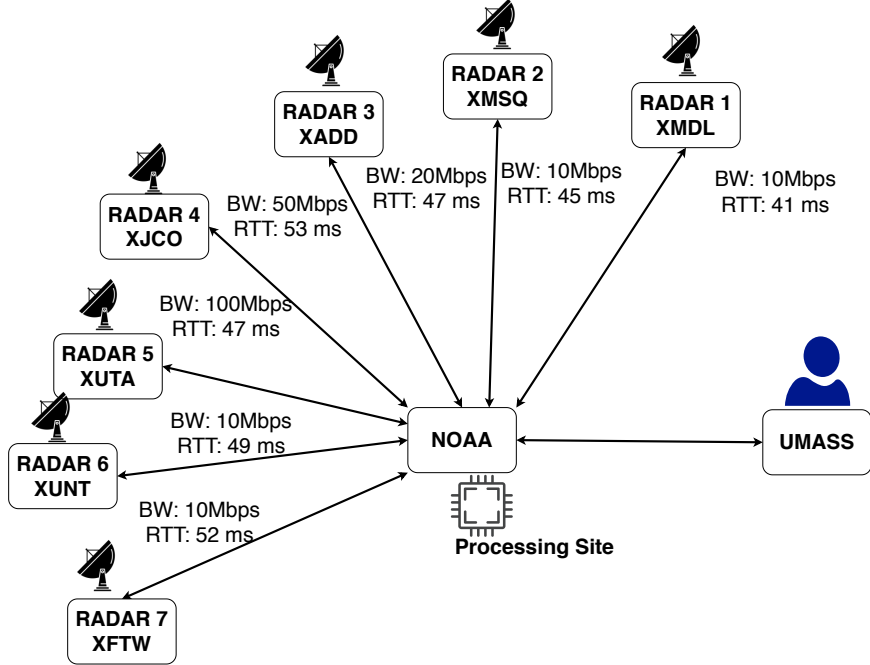
#### 4.2.4 Piggybacking

To inform clients about the name of the next file that will be produced by a radar, we design a piggybacking mechanism, where the file name is piggybacked to clients through the data of the current file. Specifically, this name is added to the metadata field of NDN Data packets. Once the client receives this information, it can launch another thread to start the parallel retrieval of the new file, before the retrieval of the current file has finished. This piggybacking approach can also be used in cases where the frequency of the radars change dynamically (e.g., when weather changes occur), so that clients stay informed about such changes.

This approach allows clients to adapt to changes that may happen on the radar side (e.g., generation of a new file, frequency changes). It comes at the cost of slightly increased sizes of certain Data packets, so that the required information can be encoded and piggybacked from radars to clients.

### 4.3 Evaluation

In this section, we present the evaluation of our information centric framework for atmospheric data. In this work, we have utilized both simulation and a testbed of Virtual Machines (VMs) deployed on Google Cloud. We first describe the evaluation setup followed by a presentation of the results.

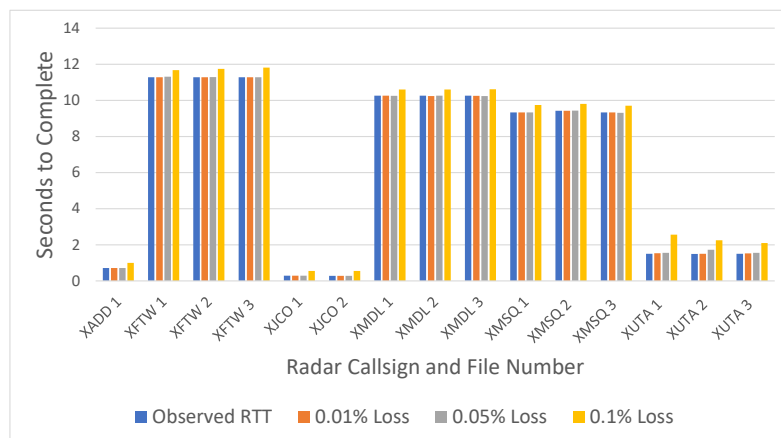
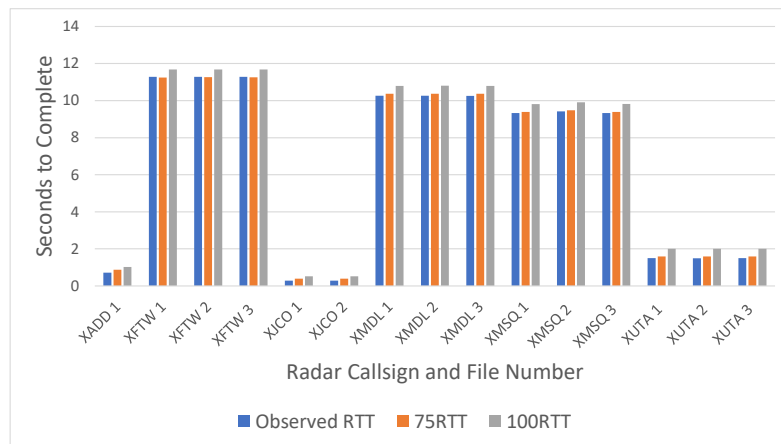
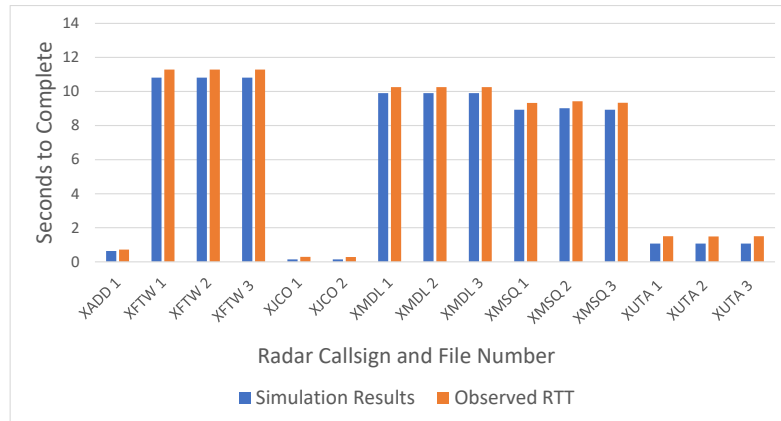


**Figure 4.3.** Experiment topology. The site names, bandwidth and delays are obtained from the actual CASA deployment.

### 4.3.1 Experiment Setup

We used both simulation and emulation to evaluate our protocol. Figure 4.3 shows the topology we used for our experiment. We created this topology from the actual CASA radar topology. We use *ping* between the NOAA and the radar sites to obtain the round trip time (RTT). The bandwidth numbers are obtained from operational experience. Note that some of these links are shared with other traffic (e.g., XMDL) while some links are dedicated (e.g., XUTA). We also looked at the actual request pattern obtained from CASA. We preserved the time between these requests and replayed the requests in real-time indicated by the actual logs. One radar was offline during our experiments so we did not use it for our experiments.

For simulations we utilize ndnSIM [41] and we used Virtual Machines (VMs) deployed on Google Cloud for emulation. One of the VMs was designated as the consumer with the other 6 as radars producing data. We used ndn-tools (ndncatchunks



**Figure 4.4.** TOP: (a) Time required for downloading datasets from radars using NDN. All datasets in this figure are needed for the weather prediction workflow. The observed RTTs from emulation are slightly higher than simulation; MIDDLE:(b) Effect of increased RTT on file download times; BOTTOM:(c) Effect of increased loss on file download times

and `ndnputchunks`) to publish and retrieve data. We used a script to determine when files were originally requested in the CASA log, and downloaded them using `ndncatchunks`. For emulating bandwidth and latency on Google Cloud, we utilized the linux utility “`tc`” to set the delay and loss. During the scenarios with increased latency and loss, we changed the “`tc`” parameters to accurately reflect the network condition. We tested each scenario 20 times for statistical significance.

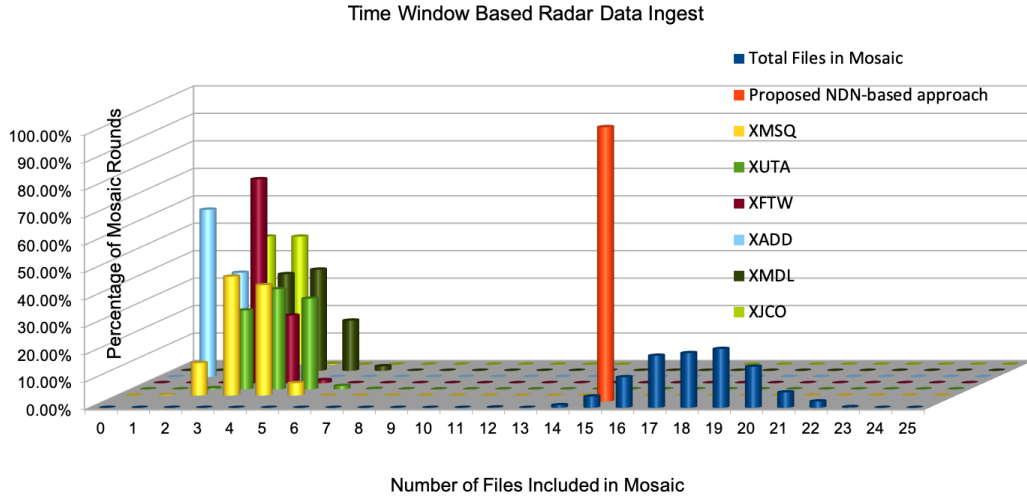
As shown in Fig. 4.3, NOAA is the processing site that communicates with the seven radars. It expresses Interest based on the naming scheme we described in Section 4.2. The producer produced a file of a certain size at a certain time. We obtained the file sizes and the respective generation times from an actual data log at CASA. The consumer requests those file at times specified in the data access log.

### 4.3.2 Results

The baseline timings from each radar can be seen in Fig. 4.4a. This baseline simulation consists of each radar starting the round at simulated second 0 and completing once all of the files from each radar are available on the consumer. The total transfer time begins once the file interest is sent out and finishes after the entire file is available on the consumer. The hosted file sizes are based on the file sizes from each radar collected during a weather event and the link bandwidth and latency match the ones shown in Fig. 4.3. While each link has a relatively low latency, the total transfer time is dependent on the hosted file size and link bandwidth. For example, the Cleburne file transfers take very little time due to the small file sizes (.2 MB) and the link’s high bandwidth (50Mbps). The opposite can be seen with the files generated by the Fort Worth radar, which are the largest of all radars (12MB) and the smallest available bandwidth (10Mbps).

As the delay and loss increase the files take slightly longer to download. The variance in time also goes up, especially with increased loss. However, the increase in

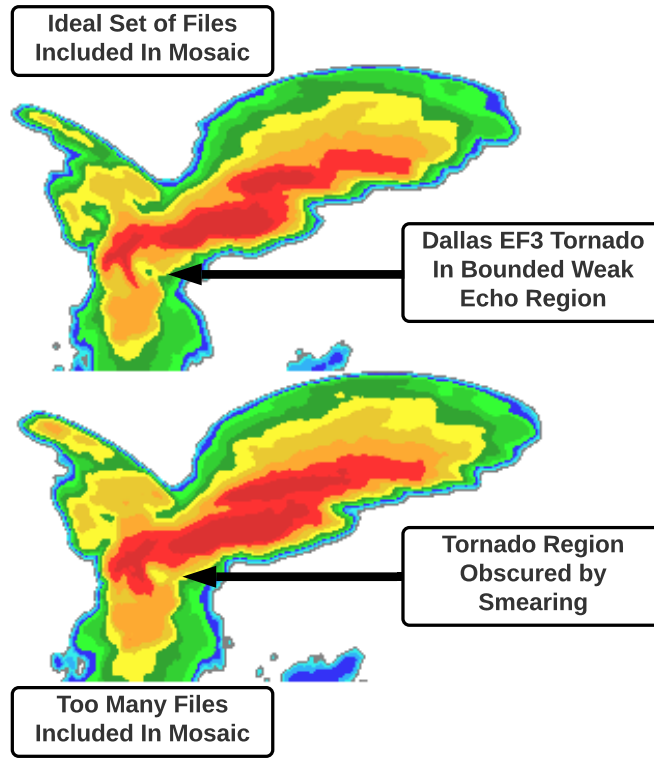
download time is small since NDN enables fast retransmission from cache and does not need to decrease congestion windows as aggressively as TCP/IP. Figures 4.4b and 4.4c show the effect of increased loss and delay on transfer times. We do not compare NDN with TCP/IP with increased loss and delay but refer the reader to previous work [76].



**Figure 4.5.** Current method vs NDN based file retrieval. The left bars show the number of files from each individual radar that are included in mosaics on a percentage basis. The blue bars show the cumulative files used for a mosaic. The red bar shows NDN’s improvement, always the ideal number of files. One radar was offline and was excluded from the simulation.

Figure 4.5 shows the comparison between the current, TCP/IP based workflow and the proposed NDN based workflow. With the current time based window, a different number of files can be included in the mosaic since the client does not always receive all the required files from the radars, an artifact of TCP/IP’s push based model. The left side of the figure shows only 40-50% of the files are included in the mosaic due to the non-determinism of the time window based process. As a result, the total number of files included in a weather prediction workflow varies considerably (12-24) as the blue bars show on the right. The ideal number of files for this operation is 15 (all files from a round across all radars, see Figure 4.4a). On the contrary, NDN’s pull based

model always provides the ideal number of the files (i.e., 15 in this case) for inclusion in the workflow.



**Figure 4.6.** Merged radar data from the same sever weather event, in the case of ideal data ingest (top) and the ingest of too many files (bottom).

Figure 4.6 shows the effect of this change. With the current time based data ingest process, it is difficult to locate the tornado that is obscured by too many files included in the mosaic. However, with the NDN based dataflow providing the ideal number of files, it is easier to see the tornado in the rendering - this could impact the decision of a weather forecaster issuing a tornado warning or not.

#### 4.4 Conclusion and Future Work

In this thesis work, we utilized NDN to improve the data retrieval process in weather sensing. We present a round-based naming scheme for the generated data and implement a piggybacking scheme to inform clients of the next available data as well as

communicate them of any data generation changes from radars. Our evaluation results show that the data-centric communication model of NDN enables the uncovering of hazardous weather events that would be hard to detect with the current CASA dataflow model. In the future, we plan to develop a prototype of this design and deploy it in the real-world CASA system. We also plan to explore the performance of our prototype on lossy links with varying delay.

## CHAPTER 5

### OPTIMIZING VIDEO STREAMING ON THE CLOUD

Video streaming occupies a significant amount of Internet traffic. With increasing online and social media platforms providing live and on-demand services upto 4K resolution, it is imperative that the user experience continues to be optimized. In this regard, we can leverage from multiple available Internet architectures and system technologies like Named Data Networking, software defined networking (SDN) and network function virtualization (NFV) to improve the overall quality of experience. In this thesis work, we present a flexible hybrid setup adaptive to run varied Internet architectures (TCP/IP and NDN) for cloud-based live video-streaming [8, 9]. Our hybrid setup [8, 9] utilizes the benefit of Named Data Networking like in-network caching and multicast mechanism without completely replacing existing IP. We evaluate our hybrid setup used for live video streaming followed by a detailed analysis of QoE [8, 9] to showcase the benefits of using NDN, its challenges when combined with adaptive bitrate streaming and suggest solutions to overcome them. Hence this chapter, focuses on:

- 1. *Evaluating our Hybrid architecture with HEVC dataset over a multi-tier network infrastructure:*** We analyze our hybrid infrastructure in comparison to traditional IP with respect to live video streaming experience. This end-to-end approach supports transparent architecture selection all the way to the application running on end systems.
- 2. *Detailed QoE analysis on live streaming with NDN & IP:*** In live stream-

ing, quality of the video playback is as important as the quality of the video. Hence with the focus to optimize user’s overall QoE, we inspect the following:

- Predict the viewability of the video content streamed at the client using Video Multimethod Assessment Fusion (VMAF) metric. This metric helps us evaluate the performance of our setup and live streaming sessions with respect to a user’s perception.
- Average bitrate & bandwidth utilization by NDN and IP clients.
- The effect of caching on live streaming QoE.
- Existing drawbacks due to the way ABR algorithms work with NDN on user’s QoE.
- Performance evaluation of NDN vs. IP clients across server and client-side bottlenecks.

Before we present the design and evaluation setup of our approach, we present the basis of building our hybrid system utilizing NDN, SDN and NFV.

## 5.1 NDN and Live Streaming

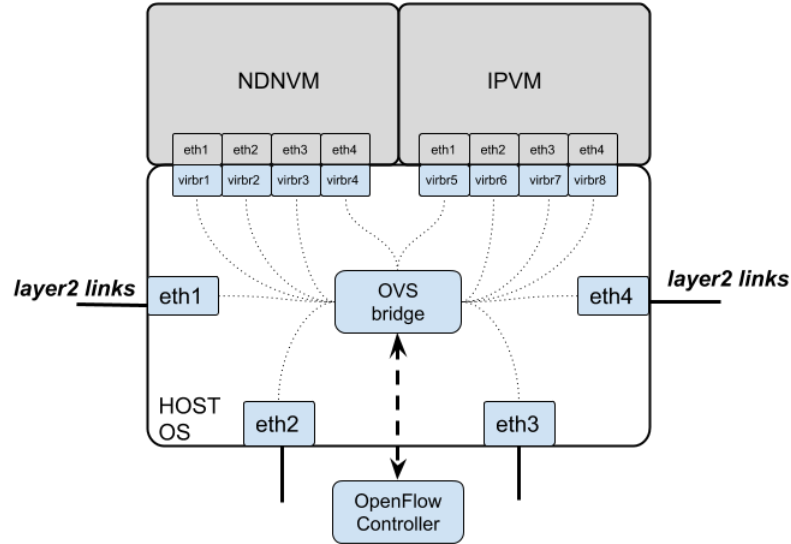
NDN provides a set of features that are well-tailored for live streaming. Especially, the feature to cache content at each router (even if only small amounts can be cached compared to a CDN or web cache) supports live streaming where the same content is requested simultaneously by many viewers that might have a high degree of geographical locality. This creates an efficient multicast mechanism. Thus, many user requests can be served from a router in the network instead of the origin server or a CDN edge server. Opposed to the traditional IP case where content sources are less diversified, in the case of NDN, DASH segments may come from various sources (a data publisher that is closest to the consumer, an in-network cache, and so on),

when multiple sources are available. Additionally, each DASH segment will be chunked into multiple NDN Data packets that are 8800 Bytes in size (by default, the size of the packets is configurable). These Data packets may come from various sources as well. Further research is needed to create congestion control algorithms that can take into account this heterogeneity of Data sources in NDN [60], which is a part of proposed work of this dissertation. The advantages of NDN for efficient delivery of video streams have been investigated in works such as [21] and [65], which make ICN-based protocols a likely candidate for the transport of live video. While the general feasibility of supporting live streaming with NDN has been demonstrated earlier [73], the approach in this thesis focuses on providing transparent solutions that do not require users' involvement to configure the underlying network technology. Additionally, this dissertation provides a much more detailed evaluation of live video streaming based on our hybrid streaming architecture.

## **5.2 SDN and NFV support for Parallel Live Video Streaming**

The inherent caching properties of NDN have proven to support live streaming applications well [73]. However, to benefit from this improvement, significant changes throughout the network have to be performed to support NDN, including installation at end systems. Hence, we propose a hybrid approach where clients can stream videos over both IP and NDN without needing NDN kernel support on hosts or fully replacing the TCP/IP protocol stack with NDN. We achieve this by utilizing virtualization techniques for standalone NDN-based container applications that can run on any client host supporting containerization. Further to enable the parallel support of IP and NDN protocol stacks, we combine SDN and NFV. Our primary goal is to build and evaluate an environment that can utilize the benefits of both these network architectures simultaneously in the domain of video-streaming while optimizing resource use and user experience. In that context, we present a brief

overview on network components that enable flexible hybridity of our architecture. We have chosen an SDN supported NFV approach for the network device design that



**Figure 5.1.** SDN-NFV setup in an intermediate router

handles IP and NDN traffic in an isolated and adaptive fashion. This architecture enables flexible topology setup and efficient resource use. In addition, it can be easily extended to support other network layer protocols (e.g., IPv6 or IPSec). In our particular use case, this architecture is used to dynamically configure customized hybrid routers that simultaneously support IP (v4) and NDN. In this architecture, SDN is used to internally (within the device) direct traffic from the physical network interface to the respective virtual interface of the respective router. This is realized by running openvswitch [56] (OVS) on the Host OS of the node. An SDN controller implements rules for isolating NDN from IP traffic in the OVS-based SDN switch as shown in Figure 5.1. While we use a centralized OpenFlow controller in our prototype, the architecture is designed such that any type of controller architecture (centralized or distributed) can be supported. Furthermore, our architecture reduces resource utilization in the case of live-streaming as we do not need client machines with

NDN-supported kernels, only hosts that can run NDN-based container applications whenever live-streaming is requested.

### 5.3 Evaluation Setup

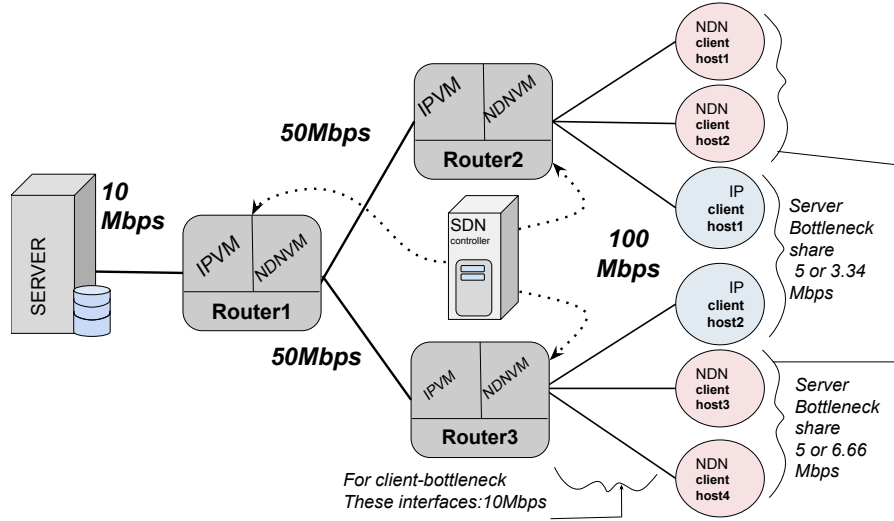


Figure 5.2. Topology for our Evaluation setup

#### 5.3.1 Experimental Setup

In this section, we describe the components of the architecture we employ for the evaluation of our approach.

**Topology:** Figure 5.2 shows the topology used for the evaluation experiments, which consists of a server, three intermediate routers (OpenFlow enabled), six Ubuntu clients, and a centralized SDN controller. The server node contains the videos being served for live streaming. The clients can be classified as an NDN client or IP client depending on which Internet architecture is being used by the client node for streaming. We chose this topology as we wanted to evaluate tiered cache effects amongst NDN clients in comparison with traditional IP clients.

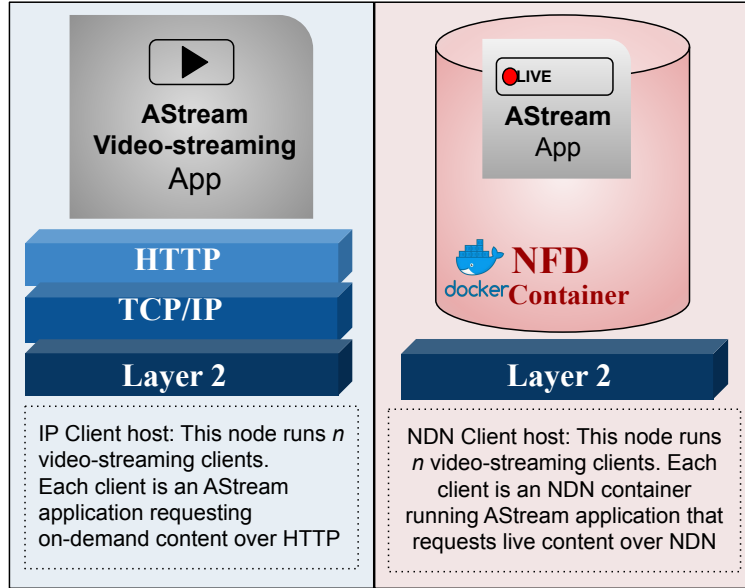
We implemented this topology on the Cloudlab testbed [15] The decision to use a testbed for our evaluation instead of using a simulation or emulation environment

like ndnSIM [42] or Mininet [24], respectively, came from the consideration that the latter would abstract several important factors that have an impact on the overall performance of our approach.

**Network Setup:** Figure 5.2 shows the available bandwidth at each link interface. We employ traffic control (tc) [17] to enforce the maximum bandwidth limits and the share of bandwidth that is available for NDN and IP traffic. Preliminary experiments revealed that TCP saturates the available bandwidth quickly due to highly optimized congestion control, slowing down NDN transfers. On the contrary, NDN is implemented in the application layer and currently lacks sophisticated congestion control mechanisms. If not mentioned otherwise, the link between Server and Router1 is shared equally between IP and NDN traffic. For the virtualized NDN routers (running on the physical router nodes 1-3 in Figure 5.2) the cache size was varied between 0MB, 250MB, and 500MB.

As we will see in the case of server-side and client-side bottleneck experiments in Sects. 5.4.2.1 and 5.4.2.5, the chosen bottlenecks were 5Mbps to serve upto 20 clients and 10Mbps to serve 5 clients, respectively. In the server-side scenario, this was done to maintain the proportionality of a limited bandwidth to multiple customers. In the case of client-side bottleneck, the highest quality segment in our dataset is 3.8Mbps. To serve 5 clients at highest quality, a bandwidth of 20 Mbps would be required, hence the bottleneck is set to 10 Mbps. In other words, these bottleneck bandwidth numbers were chosen keeping proportionality with real world in mind and not the absolute values themselves.

**Virtualization:** As introduced in Sect. 5.2, network virtualization is implemented using a combined NFV-SDN setup, whereas containers are used for application-level virtualization in the case of NDN live streaming clients. The Kernel-based Virtual Machine (KVM) hypervisor is used at the routers to handle IP and NDN traffic separately and in an isolated fashion. Figure 5.1 shows the virtualized network con-

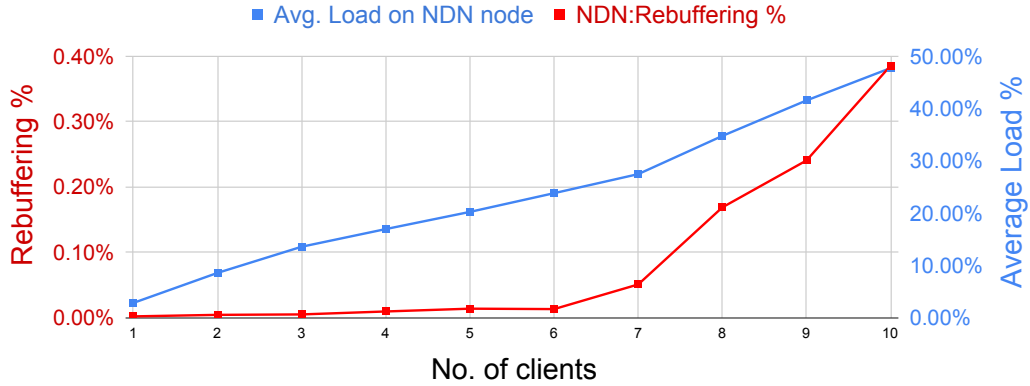


**Figure 5.3.** Execution environment for IP (left) and NDN (right) ABR streaming clients.

figuration of Routers 1-3. The physical layer 2 links of the host are mapped to the virtual interfaces of the internal VMs via OpenVswitch [56], which is managed by the SDN OpenFlow controller. Internally in the router, the traffic is routed from the server to client nodes via these virtual VM interfaces at the intermediate routers.

Using this approach, the controller uses the EtherType field (0x8624 for NDN, 0x0800 for IPv4) of an incoming frame to determine which virtual interface it has to be forwarded to. As shown in Figure 5.1, an incoming IP packet on interface `eth1` would be forwarded to `virbr5`, while an outgoing NDN packet from `virbr4` would be forwarded to `eth4`. More details on this configuration can be found in [73]. At the client side, Docker containers [45] are used for the NDN live streaming application. For the networking aspect of our containers, we used Docker macvlan networks [13] that connect each container’s virtual interface to a host’s physical interface. This also lets us monitor and regulate the traffic at Layer 2, which is needed for the NDN clients (see Figure 5.3). The choice for Macvlan networking bypasses the necessity to make the host entirely visible (as in host networking [12]) or the need to assign individual IP

addresses (as in `ipvlan` networking [13]). User-defined or default bridge networking on the docker containers was not an option because communication between the containers running on the host was not required.



**Figure 5.4.** Effect of increasing clients on CPU Load and Rebuffering percentage.

**Deciding the number of NDN containers per host:** For the large scale experiment scenarios we describe in Sect.5.4, using one physical node per client would have required a total of 65 physical nodes. Due to resource limitations of the CloudLab testbed, using such a large number of physical nodes is not feasible. Therefore, we had to resort to run several NDN containers on the same physical nodes. We ran experiments to decide how many clients we could safely run on each physical node. We ran a series of experiments where we constantly increase the number of containers (NDN) running on a physical node. We start with a single client streaming over NDN and monitor the CPU load during the process as well as the resulting Rebuffering percentage (see Sect. 5.3.3 for the definition of the metric). This experiment is then scaled up to ten clients and we observe an increase in the Rebuffering percentage along with CPU load, as shown in Fig. 5.4. While both metrics increase proportionally to the number of clients, the Rebuffering percentage for the case of five NDN container-based clients per physical node is only 0.01%. The other QoE metrics (e.g., Spectrum and QS) show the same behavior though we do not include them for brevity. Since

five clients do not significantly affect the QoE, we decided to run five NDN containers per physical host for most of our large-scale experiments.

**Video:** For our evaluation, we made use of the Big Buck Bunny video [57], which is encoded in the DASH/HEVC format and supports up to 10 different quality bitrates. (0.15 Mbps, 0.28 Mbps, 0.45Mbps, 0.61 Mbps, 0.92 Mbps, 1.54 Mbps, 2.26 Mbps, 2.89 Mbps, 3.4 Mbps, 3.8 Mbps). Each quality representation is a 10-minute long video divided into 2-second segments. We use this well-known encoding format for our dataset since it represents the popular, state-of-the-art HEVC coding standard. It should also be noted that the same dataset is used for IP and NDN evaluations, hence the data itself or its encoding type, is of less importance with the respect to the comparative performance evaluation.

**Video Server:** Since DASH is employed as the streaming technique in this work, we use a regular web server (vanilla Apache2) for IP-based streaming. For NDN, the live-streaming content needs to be named at the granularity of a DASH segment’s quality, which can be achieved with minimal effort. For a given video “*v*” whose DASH segment “*n*” is of quality “*q*”, the segment is named as “`<ndn-prefix>/<v>_<q>_<n>.m4s`.” This content is served using *ndn-python-repo* [30] which creates a repository at the server containing the DASH video segments in their respective quality levels. Combined with *ndncatchunks* [55] on the client-side, it delivers data based on content names. *ndncatchunks* implements a TCP CUBIC-like [22] congestion control algorithm that can adjust the data transfer rate based on the observed network conditions (e.g., congestion, packet loss).

**Video Client:** For the streaming client, we use a python-based video streamer, *AStream*[28] that supports multiple DASH adaptation algorithms. Here, *AStream* uses HTTP libraries and *ndn-python-repo* combined with *ndncatchunks* to download video segments over IP and NDN, respectively. We selected BOLA [71] as the bitrate adaptation algorithm. We choose BOLA because it is a near-optimal state-of-the-art

adaptive bitrate streaming algorithm for our player at the client and is also a part of the DASH reference player [72]. As long as the same ABR algorithm is used by NDN and IP for comparative analysis of the QoE, we can always use alternate ABR algorithms with this setup in the future.

### 5.3.2 Live Streaming

Since our focus is only on the streaming part, we ignore the production process of live content and only focus on the content delivery. For the evaluation of our approach, we use the Big Buck Bunny video as described in Sect. 5.3.1, where only the very first client starts requesting segments from the very beginning of the video. We specify that this first request occurs at  $t_0$ . We log this time at the video server and once a request from a new client arrives (e.g. at  $t = t_1$ ), we determine the starting segment for that client as  $(t_1 - t_0)/segmentlength$ . For example, if the first client starts requesting  $t_0 = 0$  seconds and the next client request is received at  $t_1 = 10$  seconds and we assume a video segment length of 2 seconds, then the second client is served the video from segment 5 onwards. To allow the client to determine the correct starting segment, this information is transmitted from the server to the client in the dynamic MPD file.

### 5.3.3 Metrics

Since one of our goals is to optimize QoE, we use the following metrics that are widely accepted as good representations for viewers' perceived quality.

- Average Quality Bitrate (*AQB*): One of the objectives of quality adaptation algorithms is to maximize the average quality bitrate of the streamed video. For a comprehensive QoE representation, we need to combine this metric with the *Number of Quality Switches*.

- Number of Quality Switches ( $\#QS$ ): This metric is used together with  $AQB$  to draw quantitative conclusions about the perceived quality (QoE). For example, for two streaming sessions having the same  $AQB$ , the session with the lower  $\#QS$  will be perceived better by the viewer.
- Spectrum ( $H$ ) [85]: This metric combines the variation of the video quality bitrate around the average bitrate. A lower  $H$  indicates better QoE.
- Rebuffering percentage ( $RB$ ): The average rebuffering percentage is given by:

$$RB = \mathbb{E} \left[ \frac{t_a - t_e}{t_e} \right] \%, \quad (5.1)$$

where  $t_a$  is the actual playback time and  $t_e$  is the entire video length in seconds.

- VMAF score ( $VMAF$ ): Video Multimethod Assessment Fusion [35] is a metric designed by Netflix to reflect human perception of video streaming quality. We adopted this metric to gain additional assessment of the perceived video quality at the clients. VMAF a score between two videos, original and distorted, indicates how viewable a distorted video is in reference to the original. VMAF uses Support Vector Machine regression trained on a Netflix video data set and existing image quality metrics to provide a score in the range of 0-100, where 100 means distorted video has a quality identical to the original video. For example, if the client streams the video at 3.8 Mbps throughout the streaming session of 10 minutes, the VMAF score would ideally be 100. Hence, VMAF is also highly dependent on the quality of reference.

For each of our streaming sessions, we compare the VMAF scores of the video streamed at the client (where each segment could be requested at a different quality) to the reference video. For all of our evaluations the reference video was set to the highest bitrate and resolution available in the dataset (3.8 Mbps, 1920x1080 pixels). It should be noted that to compute VMAF between videos of

different resolution, the lower resolution was upsampled in resolution to match our reference video.

## 5.4 Results

The aim of this thesis work is to present the feasibility and benefits of our hybrid network model over a traditional IP-based one with respect to live video streaming. Additionally, we motivate why NDN should be favored over IP in live streaming scenarios. In this light, the results show that our hybrid model achieves higher bitrate as NDN achieves overall higher bandwidth utilization than traditional TCP/IP. In this thesis work, we also compare the perceived video quality of NDN and IP clients by computing VMAF metrics and found that NDN always outperforms IP across different scenarios. We observe that improvement in certain QoE metrics for NDN live streaming clients increases with increasing cache size but with diminishing improvements. Finally, we identify drawbacks of implementing adaptive streaming with NDN that lead to oscillation effects. Based on our findings we suggest changes in the streaming approach to further improve the QoE even with a client-side bottleneck. To evaluate our approach, we carried out small as well as large-scale experiments on the hybrid model that also tests the scalability and robustness of our approach.

It should be noted that for each experiment, all clients start streaming at the same time and the reported results were accumulated from an average of 10 streaming sessions.

**Experiment I: Small-Scale** This scenario serves as a baseline in observing the impact of increasing clients (running on a single physical node) on the overall QoE. For this initial small-scale evaluation of our approach, we run one NDN streaming client application in a docker container on each of the four physical client nodes (one container per node as shown in Fig. 5.2), while two IP-based streaming applications run on each of the two physical client nodes (two per node).

**Experiment II: Large-Scale** For this set of large-scale experiments, we evaluate the following setups:

*a) 20 NDN & 20 IP:* Here, we increase the number of clients to 20. Since we cannot increase the number of physical nodes in the topology, we have to run 20 IP clients on two physical nodes (10 on each) and 5 NDN containers on four physical nodes, each (see Fig.5.2). Apart from the QoE analysis, we chose this setting to also study **i)** the effect of varying cache sizes (0MB, 250MB, 500MB) on the performance of NDN clients and **ii)** compare the performance of NDN and IP clients when the bottleneck is on the client-side instead of server-side.

*b) 40 NDN & 20 IP:* Further testing the scalability, the number of NDN clients is increased to 40. This increase is motivated by the assumption that popular live streaming events will be watched by many viewers (almost in a flash crowd style) putting additional stress on the system. To adjust for the imbalance between the number of NDN and IP clients, we adjust the bandwidth allocation on the 10Mbps link between the server and Router 1. As explained in 5.3.1, this number was chosen to study the effect of limited server-side bandwidth on midgress traffic. For this experiment, we allocate *2/3rd* of the bandwidth to NDN sessions and *1/3rd* to IP sessions (proportional to the number of clients of each type).

The average QoE metrics for all experiments are shown in Table 5.2. In this table, we present the Rebuffering percentage, #QS, average bitrate, spectrum, and VMAF reported across 10 live streaming sessions for experiments I, IIa & IIb. In optimizing the viewer’s QoE, a higher avg. bitrate, VMAF and lower Rebuffering percentage, #QS, spectrum is preferred. A more detailed analysis on these results is presented in the following sections. The CDFs for QoE metrics for Experiments I, IIa & IIb are presented in Fig. 5.5, 5.6, 5.7 respectively.

With respect to the end-to-end video-streaming architecture, this work focuses on the subset of this architecture from post video-processing to the distribution to the

viewers. Henceforth, we do not regard the latency from video capture (camera or camera set at a live event) to making segments available on the video server in our evaluation. To estimate latency differences between IP and NDN clients between the central server and end-client, we compare the download time differences for identical content with Experiment IIa 's 500MB cache setup. Our results show that IP clients take an average of 0.5 seconds more time than NDN clients to download the same content. In-network caching in case of the NDN clients lead to lower download times, hence, reduced average latency.

#### 5.4.1 Hybrid Streaming Architecture Analysis

Our hybrid model has the flexibility to stream videos over multiple network architectures without affecting each other adversely. On top of providing this benefit, the overall impact on QoE also needs to be investigated. This evaluation allows the comparison between an IP-only scenario and a mixed NDN/IP scenario (as presented in experiments I & II). For the IP-only case, we run 40 IP live streaming clients. Similar to NDN clients in the hybrid setup, half of the 40 IP live streaming clients also run in a container. Table 5.1 shows the average QoE metrics for the IP-only case and the NDN/IP case. For a fair comparison with cache-less IP, the cache size at the NDN routers was set to 0MB in this experiment. Table 5.1 shows that NDN live streaming performs better with respect to average bitrate, even if no caching is performed. In NDN, requests for the same content that are close in time are aggregated in the Pending Interest Table (PIT). When corresponding data arrive at an NDN router it will be multicast to more than one client if more than one outgoing interface is registered in the PIT, thus, creating an inherent multicast mechanism. We attribute the improved performance to this inherent multicast characteristics that complement live streaming scenarios well. In terms of perceptual quality of the video, it is seen that our hybrid setup gives a better VMAF score than its IP-only counterpart. In

other words, when compared to the best quality video available in our dataset, clients in our hybrid setup streamed a better quality video as opposed to when clients only use IP. However, this bitrate and VMAF enhancement is accompanied with increased #QS, Rebuffering Ratio and Spectrum. We identify the cause of this increase and present a solution to this drawback in the next section.

**Table 5.1.** Average QoE for IP-only and NDN/IP case.

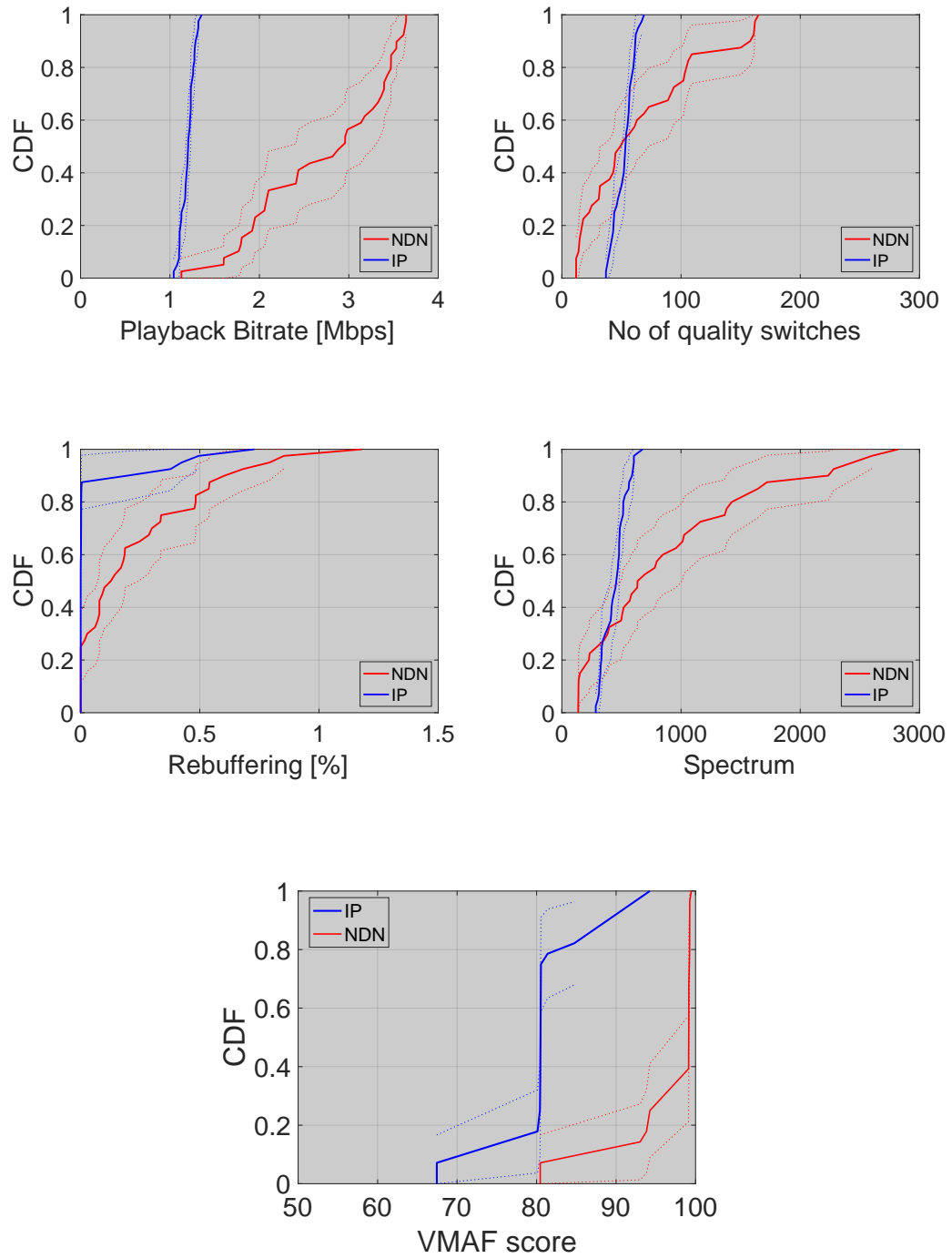
Scenario	Rebuf %	#QS	Avg. bitrate	Spectrum	VMAF
20 IP, 20 IP, 50/50BW split					
IP	0.72	33.43	0.25	23.69	42.34
20 NDN, 20 IP, 50/50 BW split, 0MB Cache					
NDN-IP Hybrid (0-cache)	3.08	68.46	0.32	95.01	55.81

#### 5.4.2 Live Streaming with NDN vs. IP

This section first analyzes the benefits of using NDN over IP for live video streaming due to its in-network caching capabilities. We further study the effect of increasing the cache size of NDN routers with respect to QoE. After discussing the benefits, we analyze the drawbacks of using NDN in relation to live video streaming and suggest possible solutions. Finally, we show that with our suggested improvements, NDN outperforms IP across all QoE metrics with server-side as well as client-side bottlenecks.

##### 5.4.2.1 Higher bitrate & bandwidth utilization by NDN

NDN clients report higher average playback bitrate across all experiments as can be observed from the corresponding column in Table 5.2 and their respective CDFs (Figures 5.5, 5.6, 5.7). While throughput for NDN traffic is limited to 5Mbps (50% of the 10Mbps link between server and Router 1) in experiments I & IIa, the combined average bitrate is almost double ( $4 \times 2.74 = 11\text{Mbps}$ ) or higher ( $20 \times 0.75 = 15\text{Mbps}$ ).

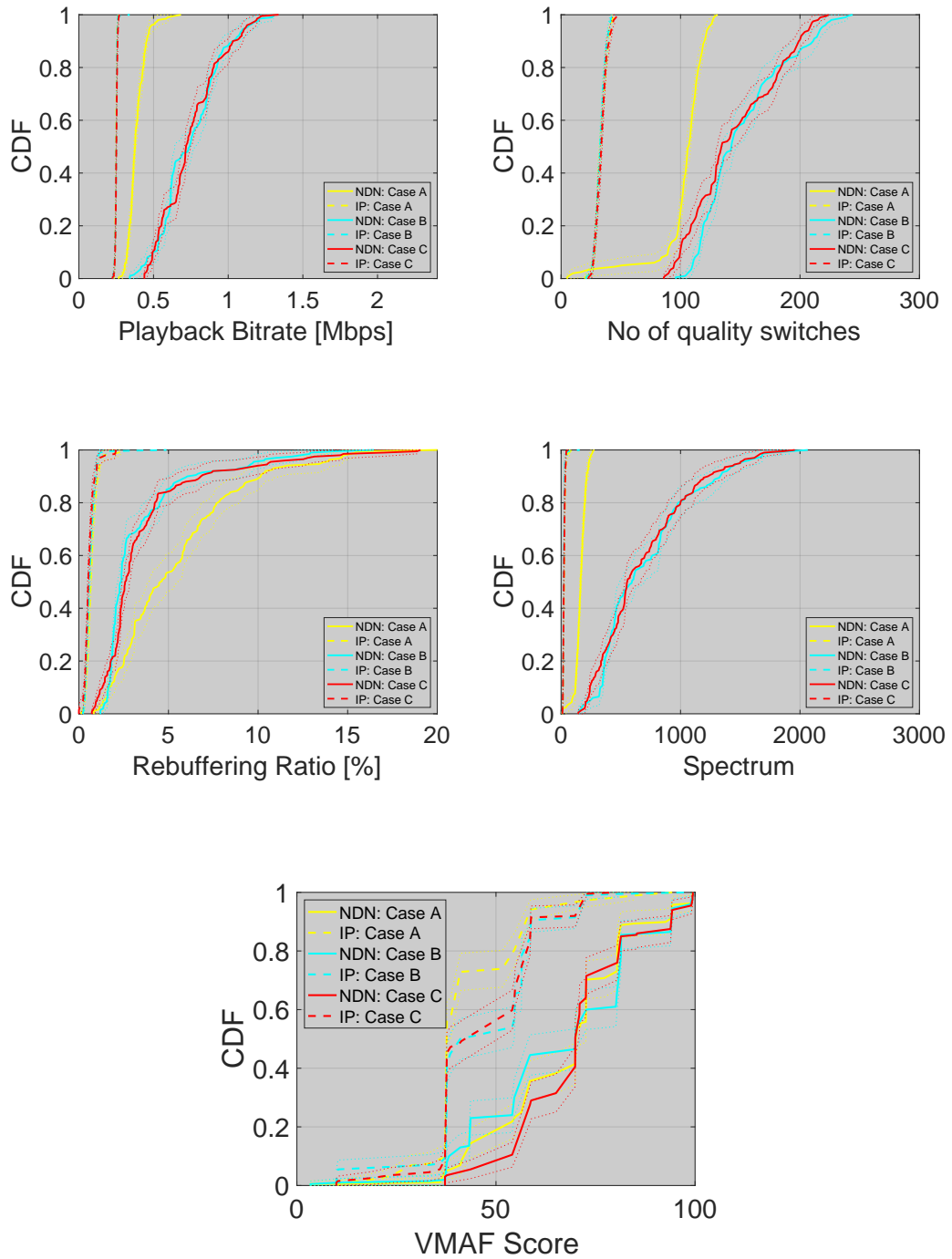


**Figure 5.5.** Cumulative Distribution Functions for QoE metrics for the case of 4 NDN and 4 IP clients.

Similar conclusions can be drawn from experiment IIb results, as cumulative avg. bitrate increases with increasing clients. This confirms our hypothesis that NDN benefits live streaming scenarios due to its inherent in-network caching and improved handling of potential NDN segment retransmissions. In comparison, the cumulative average bandwidth for IP clients for both experiments stays slightly less or equal to ( $4 \times 1.2 = 4.8\text{Mbps}$ ,  $20 \times 0.25 = 5\text{Mbps}$ ) the allotted  $5\text{Mbps}$  on the link between server and Router 1. Motivated by NDN consistently reporting higher avg. bitrate, we also report bandwidth utilization by NDN versus IP streaming clients as presented in Tab. 5.3. If we define bandwidth utilization as the percentage of average playback bitrate per client for bottleneck bandwidth allotted per client, then NDN utilizes up to 450% of the available bandwidth ( $40 \times 0.78 = 31.2\text{Mbps}$  out of  $6.6\text{Mbps}$ ) whereas IP can only utilize up to 100% ( $20 \times 0.17 = 3.4\text{Mbps}$  out of  $3.4\text{Mbps}$ ) of it in the best case scenario (computed from experiment IIb results). These results indicate NDN’s superior bandwidth utilization over IP across all scales. This clearly demonstrates the scalability and the benefits of using NDN for live streaming applications.

#### 5.4.2.2 Comparison of Perceived video quality

Traditional metrics might not always correlate with a user’s perception of a good video quality. It is imperative to test the quality of the video being streamed as well as its playback quality. The VMAF metric (as previously described in 5.3) predicts a viewer’s perception of a video streaming quality in the form of a score between 0 to 100. Since, VMAF is reference-based, a score of 100 in our evaluation denotes video streamed at best available quality in the dataset. As we compare the quality of the video streamed at NDN and IP clients for each experiment in Tab. 5.2, it can be observed that NDN consistently reports higher VMAF than IP with same share of available bandwidth per client. The individual scores decrease with increase in number of clients (65 for 40 NDN clients in experiment IIb vs. 96.9 for 4 NDN clients



**Figure 5.6.** Cumulative Distribution Functions for QoE metrics of 20 NDN client cases with varying cache sizes. Case A: NDN cache size 0MB; Case B: NDN cache size 250MB; Case C: NDN cache size 500MB

**Table 5.2.** Average QoE metric results from streaming sessions across different experiment setup

Scenario	Rebuf %	#QS	Avg. bitrate	Spectrum	VMAF
<b>Exp I: 4 NDN, 4 IP, 50/50 BW split, 500MB Cache</b>					
NDN	0.2	66.9	2.74	925.1	96.9
IP	0.1	52.9	1.20	446.9	82.2
<b>Exp IIa: 20 NDN, 20 IP, 50/50 BW split, 0MB Cache</b>					
NDN	5.44	103.33	0.39	164.06	67.87
IP	0.71	33.6	0.25	25.96	43.74
<b>20 NDN, 20 IP, 50/50 BW split, 250MB Cache</b>					
NDN	3.39	152.06	0.75	710.90	66.97
IP	0.64	33.31	0.25	22.99	47.32
<b>20 NDN, 20 IP, 50/50 BW split, 500MB Cache</b>					
NDN	3.36	145.58	0.75	685.35	70.9
IP	0.58	34.18	0.25	24.56	47.82
<b>Exp IIb: 40 NDN, 20 IP, 66/33 BW split, 500MB Cache</b>					
NDN	6.24	147.03	0.78	749.28	65.03
IP	7.14	22.49	0.17	17.39	13.75

in experiment I). The correlation of available bandwidth to VMAF is unclear, as seen with 0-cache and 250 MB cache results in IIa. Here, increased cache and ABR do not necessarily lead to increased VMAF. However it is clear that NDN’s caching benefits lead to a better user viewing experience compared to IP.

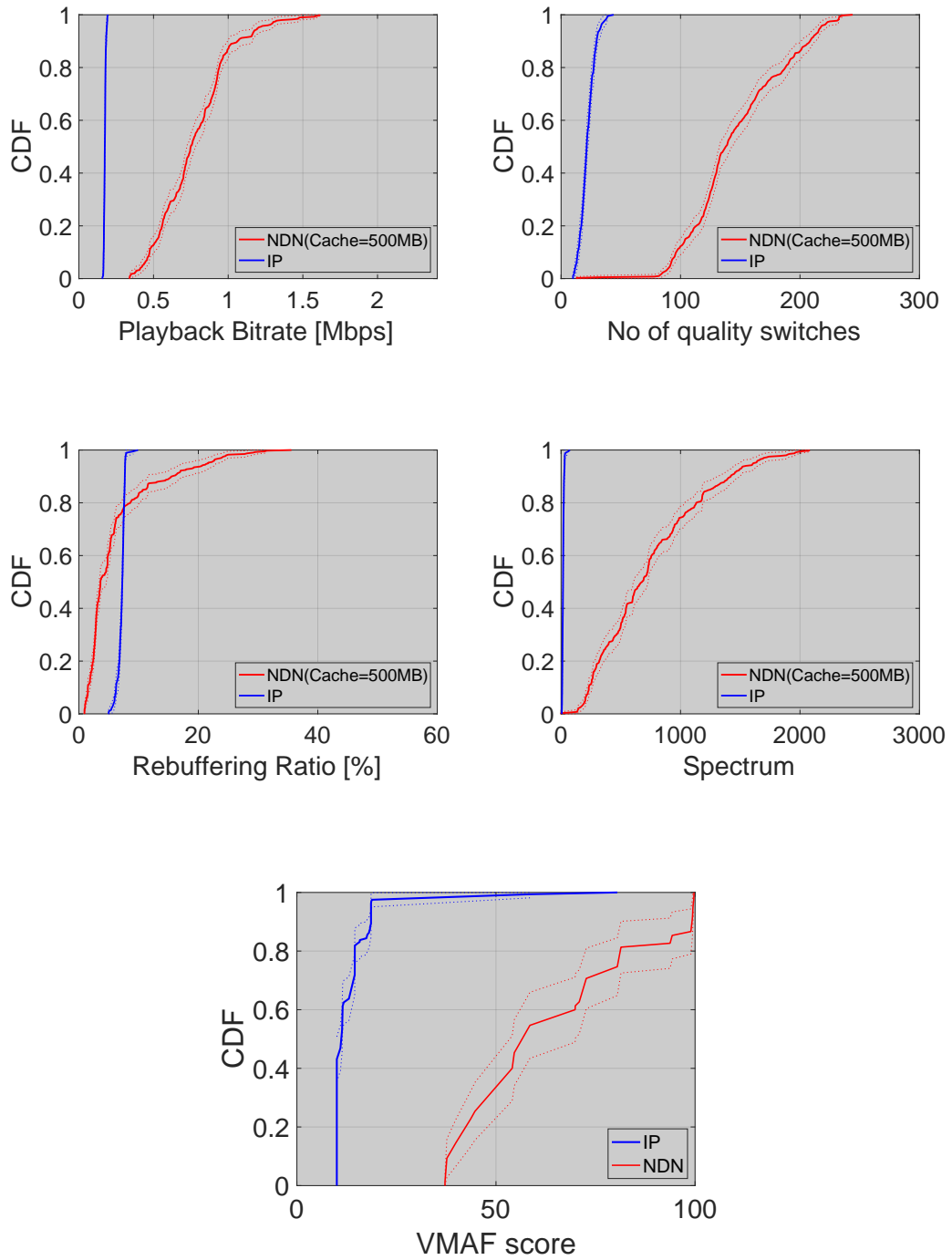
Figures 5.5, 5.6, and 5.7 also report higher average VMAF in the case of NDN clients as compared to IP clients for experiment I, IIa & IIb respectively. In other words, NDN clients consistently streamed better perceivable quality video than IP clients.

**Table 5.3.** Comparison of the theoretical available average bandwidth for each client with the average playback bitrate observed by each client.

Layer 3	Total # of clients	Bottleneck BW split	Theoretical average BW	Avg. playback bitrate	BW utilization
NDN	4NDN-4IP ( $\Sigma$ 8)	50%/50%	1.25	2.74	219%
IP	4NDN-4IP ( $\Sigma$ 8)	50%/50%	1.25	1.2	96%
NDN	20NDN-20IP ( $\Sigma$ 40)	50%/50%	0.25	0.75	300%
IP	20NDN-20IP ( $\Sigma$ 40)	50%/50%	0.25	0.25	100%
NDN	40NDN-20IP ( $\Sigma$ 60)	66%/33%	0.17	0.78	458%
IP	40NDN-20IP ( $\Sigma$ 60)	66%/33%	0.17	0.17	100%

#### 5.4.2.3 Effect of caching

In this section, we present the effect of varying cache sizes for NDN clients in a large-scale scenario (experiment IIa). Comparing the results for the three different cache-sizes, we make three major observations. First, increasing the cache size beyond 500MB does not lead to a further increase in QoE. Even the increase from 250MB to 500MB results in marginal improvement of the QoE metrics. Third, the rebuffering percentage is almost identical for all three cases (see Figure 5.6), but slightly increasing #QS indicating interdependence between these QoE metrics. Fourth, as shown in Figure 5.6, the #QS metric is lowest for the 0MB cache and increases for the 250MB and 500MB cache cases. Compared to IP, the #QS metric is higher for all three caching scenarios and the difference to NDN is larger than in the small-scale scenario. From the large-scale scenarios in Table 5.2, we observe that #QS increases for NDN and decreases for IP. In the next section, we explain the reason behind this with a hypothesis as well as proposed solutions.



**Figure 5.7.** Cumulative Distribution Functions for QoE metrics for the case of 40 NDN clients and 20 IP clients.

#### 5.4.2.4 Effect of Oscillation effect on Rebuffering, #QS & Spectrum: Causes & Solution

Examining the additional QoE parameters shows that the increased playback bitrate and higher VMAF in the NDN case comes with the trade-off of increases in quality switches and spectrum. While the rebuffering percentage stays comparable to the IP scenario in small-scale results (experiment I), it increases in comparison with IP for large-scale cases (experiment IIa & b). This increase might have a negative impact on the viewer’s QoE that can outweigh the positive impact of an increased playback bitrate. Hence, in this section we first identify the underlying cause and then suggest a solution to this problem.

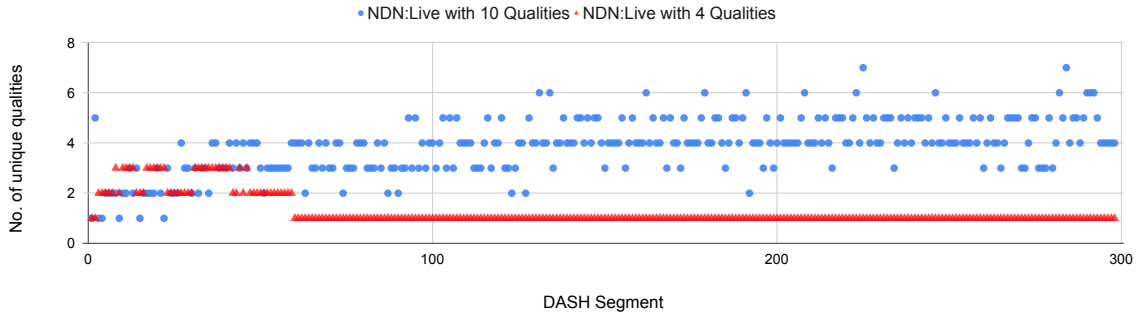
*Cause/Hypothesis:* First, higher #QS in NDN as opposed to IP can in part be explained by examining the average bandwidth available to each IP client on the bottleneck link. In the case of 20 IP clients the average bandwidth is 0.25Mbps per client. This bandwidth is only sufficient to stream the lowest out of 10 playback bitrates (0.15 Mbps). For NDN live streaming with 250MB cache size, the average bitrate is 0.75Mbps, which is sufficient to stream the four lowest playback bitrates. This clearly demonstrates higher opportunities for bitrate quality changes in NDN as opposed to IP. Second, we conjecture that some of the decreases in QoE are caused by the interplay of ABR streaming and NDN. As presented by Grandl et al. [20], the potentially random placement of video segment on either the server or the caches can lead to so-called “oscillation effects”. For example, if a client receives a video segment in low quality from a cache, the measured download rate might be high. Based on this observation, the ABR algorithm at the client (e.g., BOLA [71]) might decide to request the next segment in a higher quality. If this segment is currently not stored at the cache, the client has to retrieve the segment from the server and most likely experiences a lower download rate. This results in the client requesting the next segment at a (much) lower quality. This alternate retrieval of segments from server or

**Table 5.4.** Avg. QoE and hit-miss ratio at respective caches for Livestreaming on NDN with 10 qualities vs. 4 qualities

Layer 3	Rebuff%	#QS	ABR	Spectrum	VMAF	Hit-Rate at Router1	Hit-Rate at Router2	Hit-Rate at Router3
NDN (10-qualities)	3.36	145.58	0.75	685.35	70.9	0.09	0.17	0.21
IP	0.58	34.18	0.25	24.56	47.8	NA	NA	NA
NDN (4-qualities)	0.1	10.5	1.47	14.3	78.6	0.12	0.63	0.72
IP	0.6	34.2	0.25	25.0	48.3	NA	NA	NA

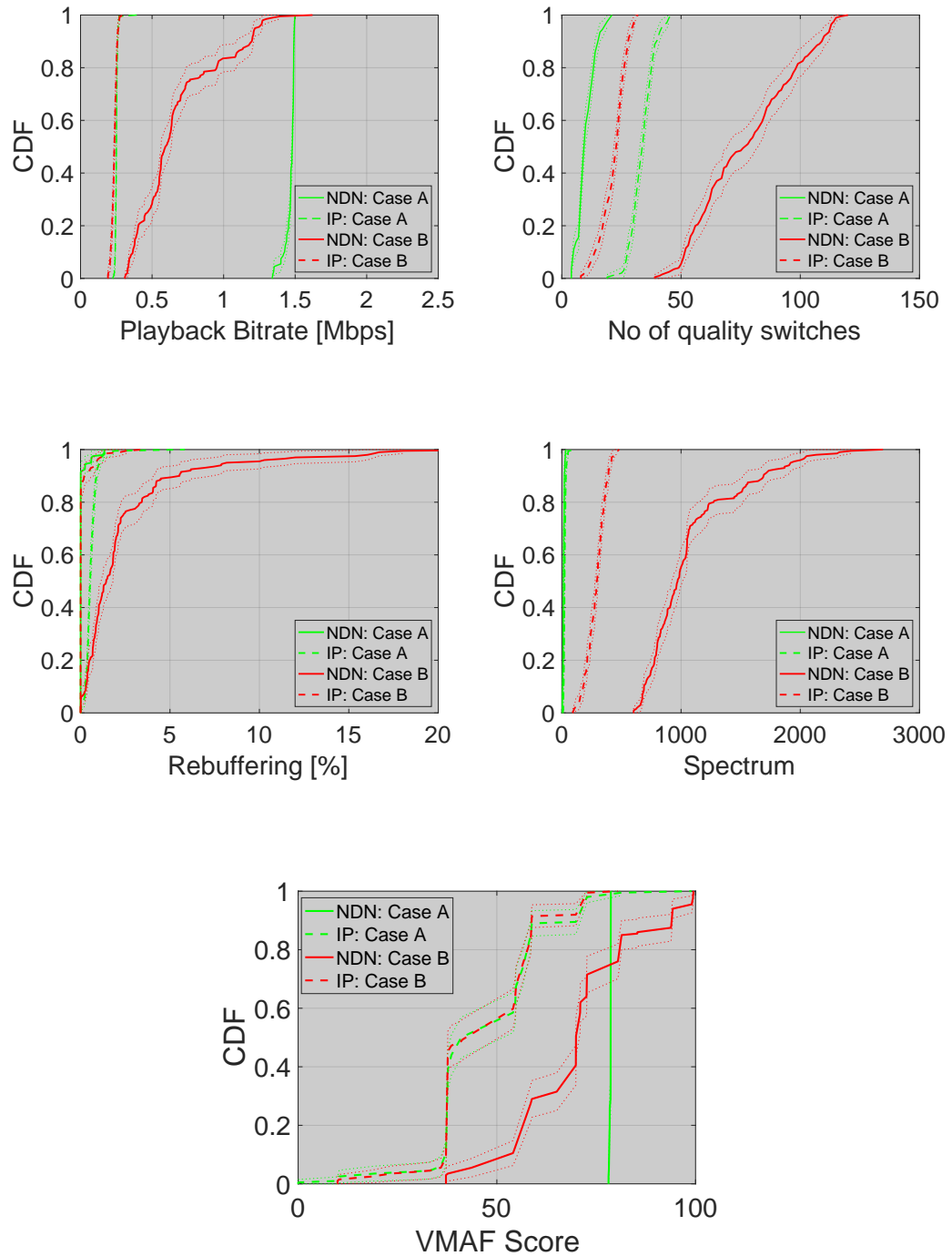
cache can happen several times during a streaming session leading to increased #QS and spectrum in the case of NDN. Furthermore, these suspected oscillation effects caused by potential low hit rates on the caches lead to lower download rates and hence higher rebuffering percentages. This led us to further investigate the effect of available bitrates on cache hit-ratios causing potential “oscillation effects” impacting QoE. *Confirmation:* To gain more insight in the correlation between a higher number of available bitrates (in the case of NDN), the oscillation effect caused by low cache hit-rates, and the resulting QoE (#QS, spectrum & Rebuffering percentage), we first analyze the hit rate (per NDN segment) on the three routers (1-3) used in the topology for these experiments. In an additional experiment for the 20NDN/20IP client case, we set the cache sizes on all three routers to 2GB (large enough to cache all DASH segment in all playback bitrates of the video which totals to 1.4GB). We observed that the hit rates, surprisingly, do not increase with an increase in cache size and are consistently low. Reported hit-rates were 0.09, 0.22, and 0.25 for 250MB caches, 0.09, 0.17, and 0.22 for 500MB caches and 0.08, 0.23, and 0.19 for 2GB caches in Router1, Router2, and Router3, respectively. The most probable reason behind low

hit rates would be that all 20 NDN clients request a very disjunct set of qualities for individual DASH video segments (every segment is available in ten different bitrates (see Sect. 5.3.1)). Fig. 5.8 further proves this hypothesis by showing the number of unique qualities per segment that are requested by the clients in an entire streaming session. With a maximum of ten qualities available, NDN clients requested up to



**Figure 5.8.** Distinct qualities requested per DASH video segment by an NDN client when #qualities available at the server are 10 vs. 4. Total DASH segments in our streaming video are 298. Maximum possible unique qualities requested in both cases would be 10 and 4 respectively and minimum would be 1.

seven different qualities for a given DASH segment and an average of five different qualities for all DASH segments. This seemingly high variation in requested qualities results in low hit rates. With ten clients connected to Routers 2 and 3 each, almost every client requested a different quality per DASH segment. *Solution:* Based on these observations, we reduced the available playback bitrates from ten to four (0.15 Mbps, 0.45Mbps, 0.92 Mbps and 1.54 Mbps) for NDN live streaming and conducted an experiment with 20 NDN and 20 IP clients (similar to Experiment IIa in Sect. 5.4) and cache sizes set to 500MB at the routers. The IP clients are still able to select from all ten playback bitrates. We observe that the number of distinct qualities were much lower in the case of NDN. Previously with ten bitrates, *mode* for the distinct qualities requested per DASH segment was four and a *maximum* of seven distinct qualities. But with four available bitrates, the *mode* reduces to one with a *maximum* of three

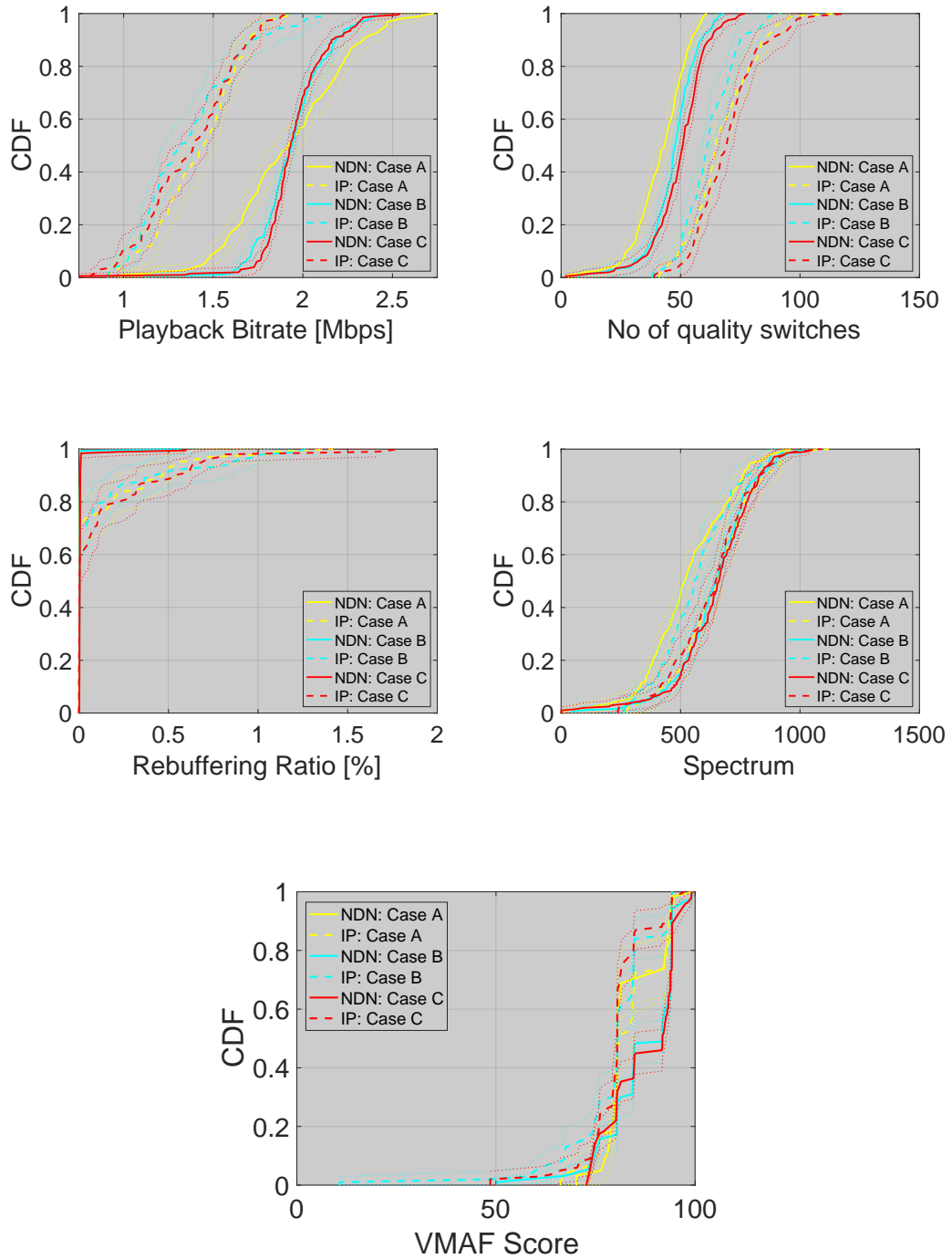


**Figure 5.9.** Cumulative Distribution Functions for QoE metrics for the case of limited playback bitrates. Case A: NDN chooses from 4 qualities & IP from 10. Case B: Both NDN & IP choose from 10 qualities

distinct qualities requested by all NDN clients. More importantly, this resulted in higher hit rates as reported in Table 5.4. Figures 5.9 and Table 5.4 further confirm our hypothesis that the reduction of available qualities has a significant impact on QoE. From the CDF graphs in Figure 5.9, we observe that when NDN chooses from 4 qualities, it results in higher playback bitrate, lower quality switches, lower rebuffering percentage, spectrum as compared to when NDN chooses from more available qualities (in our case, 10). The VMAF score also improves with our solution indicating better visual quality of the streamed video. Furthermore, when NDN clients choose from fewer available bitrates, they outperform IP clients in terms of all QoE metrics with reduced #QS, rebuffering ratio, spectrum and increased ABR & VMAF. Additionally, we ran similar experiments for I & Iib (4NDN and 4IP, 40NDN and 20IP clients respectively) with reduced available playback bitrates around their expected average. For each case, reducing available bitrates improves all QoE metrics of NDN clients as compared to IP as well as when compared to NDN clients streaming with full set of qualities. For Iib, VMAF score improved from 65 to 77, higher ABR of 1.4Mbps and lower #QS, Rebuffering % and Spectrum. These results conclusively show that with the suggested modifications, NDN outperforms IP across all QoE metrics and across different scales of experiments. Clearly, the selection of the playback bitrate for NDN live streaming was informed by the results presented in Sect. 5.4.2.1 and Table 5.2. With an average bitrate of 0.75Mbps in the case of 500MB cache size, selecting playback bitrates was straight-forward. To make such an approach feasible for an actual system, an approach could be implemented that collects the average client bandwidth and adapts the playback bitrates that are advertised in the MPD file once a sufficient amount of data has been collected.

#### 5.4.2.5 NDN vs. IP with client-side bottleneck

Our motivation to enforce a server-side bottleneck so far was to observe how the architecture responds to reduced midgress traffic. Obviously, the last hop to the client can also be a bottleneck (especially in mobile scenarios). Hence, a comparative QoE analysis of NDN and IP clients streaming over a network with client-side bottleneck is also required to provide a complete picture of NDN's superior performance in the case of live-streaming. To further study such a scenario, we increased the bandwidth of the server side link (see Figure 5.2) to 1Gbps and added a client-side bottleneck of 10Mbps. For equal distribution of resources and a fair comparison, 5 clients were run on each node (both IP and NDN) and all nodes were connected to routers 2 & 3 with 10 Mbps links. As Figure 5.10 shows, NDN outperforms IP across all QoE metrics with 4 qualities. Even though the caching is limited with this client-side bottleneck, NDN still benefits from it as well as from its inherent nature of multicast and retransmissions to the nearest cache. It is also interesting to note that contrary to the results shown in Figure 5.9, QoE for 10 qualities is more comparable to the one with 4 qualities (0.15 Mbps, 0.92 Mbps, 2.89 Mbps, 3.8 Mbps) around an ideal average of 2Mbps. In this case, the bottleneck at the last hop dampens the oscillation effect. The bitrate is slightly increased in the case of 10 qualities because of more available qualities which in turn slightly worsens #QS, spectrum, and rebuffering percentage. In the event of loss, it is intuitive that the QoE will be lower than in the lossless cases. However even without reduced bitrates, Figure 5.10 shows that NDN still outperforms IP with respect to all QoE metrics besides the spectrum due to increased magnitude of variability with 10 qualities. NDN clients also consistently report higher VMAF across all cases than IP.



**Figure 5.10.** Cumulative Distribution Functions for QoE metrics with client-side bottleneck. Case A: NDN chooses from 4 qualities & IP from 10. Case B: Both NDN & IP choose from 10 qualities. Case C: Case B with 1% packet loss

## 5.5 Discussion

### 5.5.1 CDNs and NDN

To accomplish large-scale live streaming, CDN providers need to *work around* several limitations of the TCP/IP architecture such as lack of native IP multicast support, lack of support for caching at the network layer, and more. First, they need to create a data delivery infrastructure that can scale to the enormous demand. Second, TCP/IP does not support easy and efficient multicast. Finally, TCP/IP does not support caching of content at the network layer. Caching at the application layer (e.g., HTTP caches) also introduces several issues. For example, clients must find and connect to the appropriate caches, the CDNs must strategically place content in the caches, and continuously monitor for failure or service degradation. To further emphasize this point, we conducted a simple experiment where we provide IP clients with application-level caches using nginx [50]. The topology used is same as shown in 5.3 where each of the IP VMs have the nginx reverse proxy caches running on them. For a fair comparison, we let 20 NDN and 20 IP clients live stream the same video with full dataset of qualities available at the server and both IP and NDN clients are connected to 500MB intermediate caches. As can be observed from figure 5.11 and Table 5.5, IP clients benefit from the caches in terms of average bitrate only (refer Table 5.2) but performs very poorly in terms of other QoE metrics. It should be noted that both IP and NDN clients are provided with default caching strategy and even though IP clients report slightly higher average bitrate than NDN clients, the rebuffering %, #QS and spectrum are considerably worse. NDN, on the other hand, provides several desirable properties for live streaming. First, NDN provides native multicast at the network layer through Interest aggregation and caching, caches Data packets at the network layer, and does not require complex infrastructure setup beforehand. In the case of cache or link failure, NDN is able to mitigate the failure by forwarding traffic around it (if there are multiple paths) without application or

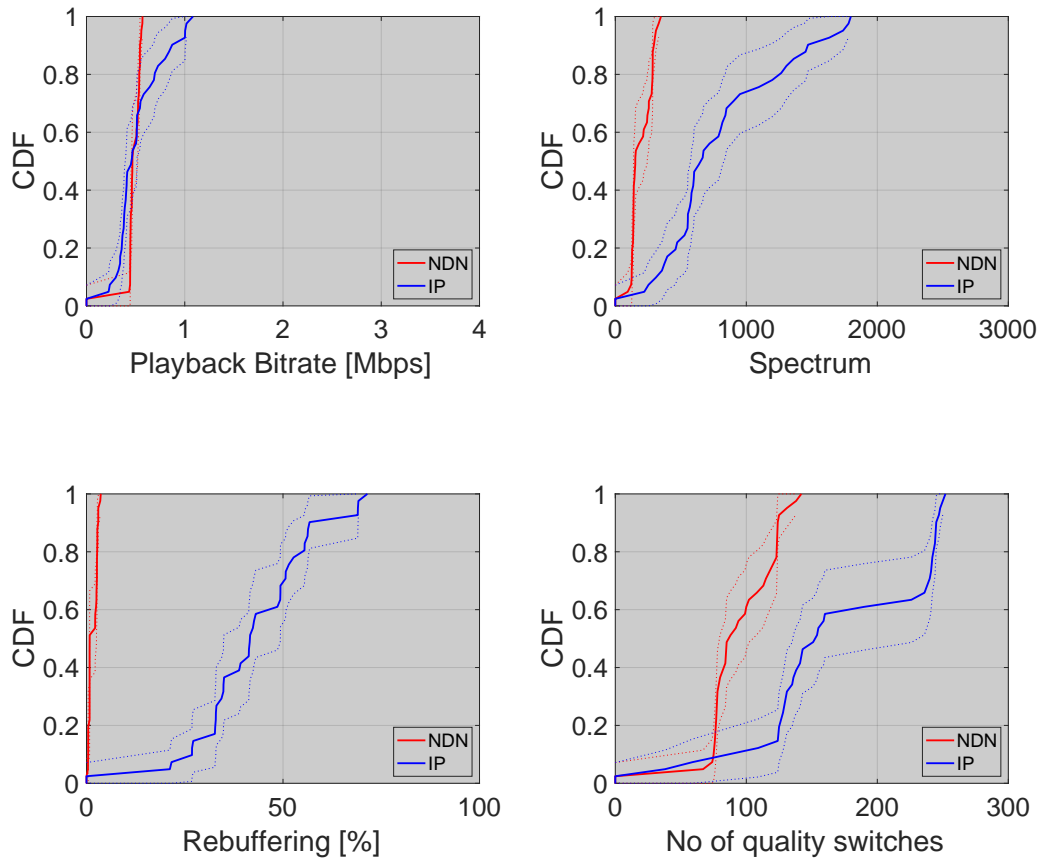
operator intervention. These properties can be utilized for live-streaming - both inside a CDN infrastructure and user-based live streaming. A CDN can certainly deploy several NDN based live-streaming servers inside its infrastructure that will work in parallel with existing IP-based streaming mechanisms. Previous work by Ghasemi et al. [18] compared an NDN based video streaming solution deployed on the NDN Testbed with two well-known CDNs, Akamai and Fastly. While this work did not attempt a hybrid solution, it showed NDN can reduce origin workload compared to traditional CDNs. Another work by Thelagathoti et al. [77] demonstrated that NDN deployment inside a CDN infrastructure will help with efficient data retrieval and improve QoE.

**Table 5.5.** QoE for NDN and IP clients streaming with full available dataset with intermediate caches present for both IP and NDN

Client type	Rebuf %	#QS	Avg. bitrate	Spectrum
NDN	1.65	95.78	0.48	195.97
IP	42.93	171.3	0.52	812.4

### 5.5.2 SDN, NFV, and NDN

Several works have highlighted the benefits of an architecture that integrates SDN with NFV [14, 43]. For instance, the agility this integration provides to infrastructure and network service design can be very desirable for any dynamic and scalable architectural framework [54]. NFV and SDN combined have revolutionized network architectures that are able to cope with the continuous growth in data-traffic [51]. They provide the ability to virtualize any network infrastructure based on its requirements. Hence, the decision to use our SDN-NFV setup. There has been work on SDN-NFV infrastructures that handle heterogeneous network technologies (e.g., [40, 80, 81]) but none of them compare multiple network stacks and their performance. In [38], Mai et al. have implemented NDN technologies with SDN-NFV support but the novelty



**Figure 5.11.** Cumulative Distribution Functions for QoE metrics of 20 NDN and 20 IP client cases with 500MB cache setup for both.

of our work lies in the heterogeneity of the network protocol stacks as implemented in [73]. Performance analysis over IP versus non-IP protocols [29] or specifically IP versus NDN protocols [66, 64] have been executed before but not with the design flexibility that comes with the benefits of SDN programmability [73]. Kanada et al. [29] use virtual link tunnels to encapsulate IP and non-IP packets whereas Satria et al. [66] evaluate them separately and not in the context of video streaming in a non-virtualized setup.

Several works have proposed translation between TCP/IP and NDN so that they can coexist. Moiseenko et al. proposed TCP over ICN where TCP traffic is converted into ICN traffic [48]. Shannigrahi et al. proposed IPoC [68] where TCP/IP traffic is encapsulated into NDN packets for transport. Refaei et al. proposed an IP-ICN gateway that allows IP client-server communication to operate seamlessly through an NDN cloud [59]. Nour et al. [52] propose an approach that uses NFV for ICN/IP hybrid routers that require predefining a set of regions. On the contrary, we utilize layer 2 header information to indicate different types of traffic that allows us to differentiate between IP and NDN traffic in real-time and decide whether to forward it to IP or NDN data sourceS.

Several papers [47, 34, 62, 63] motivate the idea behind using the NDN protocol for live video streaming. Mohammed et al. [47] and Li et al. [34] realize NDN’s superiority as compared to IP in live-streaming over wireless networks using WiFi direct and WiFi’s built-in broadcast mechanism, respectively. Samain et al. [64] and Rainer et al. [58] make similar comparison of the dynamic adaptive streaming performance on IP over NDN as shown in [73]. While Samain et al. [64] experiments with different modes of NDN under wireless loss detection recovery and load balancing state, our work aims to provide the best user viewing experience leveraging native NDN. Although on a different hybrid setup that runs both the protocol stacks simultaneously, we also compare the performance metrics of IP and NDN with respect to video streaming.

Our main motivation is to use this comparison to build an architecture directed towards a novel hybrid video-streaming experience.

## 5.6 Conclusion & Future Work

In this thesis work, we propose a hybrid architecture that supports live video-streaming parallelly over NDN and TCP/IP to improve overall quality of experience. We implement this setup in the CloudLab testbed and perform an extensive performance evaluation of our hybrid streaming approach where we not only measure the playback quality but also the video quality itself. The evaluation results demonstrate the profit in terms of average bitrate, bandwidth utilization and better video quality (VMAF) from our approach and reveal that live streaming can be performed efficiently, is scalable, and provides good QoE with the help of NDN. Using containers for the NDN streaming clients provides a method that can activate such clients without end user involvement.

Counter-intuitive to experience gained in the case of ABR streaming over TCP/IP, we show that a reduced set of available playback bitrates leads to better performance in the case of NDN-based live streaming and outperforms IP-based live streaming under both server and client-bottleneck conditions and across different scale of clients. We also show that NDN-based live streaming behaves fairly to IP-based session and does not negatively impact the QoE of these sessions. In future work we plan to develop new ABR algorithms that are cognizant that NDN provides in-network caching. We will also study if the approach of caching at the level of NDN Segments is appropriate in the case of ABR streaming.

## CHAPTER 6

### HANDLING OSCILLATION EFFECT DUE TO CACHING TOWARDS IMPROVED QOE

Dynamic adaptive streaming over HTTP (DASH) is a popular standard as an adaptive bitrate streaming solution. This technique not only enables high quality streaming of all types of media content but also adapts dynamically to the changing network conditions. The end-goal of such a streaming technique is to provide high quality content to the end-user while reducing unwanted phenomenon such as too many quality changes, playback stalling due to re-buffering, etc. To manage everyday increasing media content over the web, caching solutions are adopted to reduce the amount of traffic being served by the media server. However, the role of caching in combination with adaptive bitrate streaming needs to be investigated further. As discussed in the previous chapter 5, NDN's in-network caching properties help improve average playback bitrate and perceived video quality in live video streaming, but they can also adversely affect other QoE factors such as rebuffering, number of quality switches (#QS) and spectrum [8, 9]. This negative impact on the QoE can be contributed to a phenomenon called *Oscillation effect*. As introduced in Grandl et al. [20] and discussed in chapter 5, oscillation effect is the phenomenon of frequent alteration between server and cache as the source for requesting video segments. But before we further investigate into how oscillation effect can be identified and mitigated, we need to understand the factors that cause an oscillation effect and how they adversely affect an end-user's QoE.

## 6.1 Oscillation effect and its disadvantages

In adaptive bitrate streaming (ABR), each video segment is available in multiple qualities at the server. The video streaming client requests each segment at a quality determined by the client's ABR streaming algorithm based on factors like download rate of the previous segments, buffer occupancy, etc. During a streaming session, the segment requested by the client maybe served directly by the server unless its cached at one of the intermediate network caches. In a scenario where the requested segment is served by the cache, the download time is potentially faster than it would be if it were fetched from the server. This fast segment download time leads to the client requesting the next segment at a higher quality. If the next segment requested at higher quality happens to be unavailable at the cache, the request gets served directly from the server or a cache higher up in the hierarchy, leading to slower download times which effectively leads to the client requesting the next segment at a lower quality. These oscillating requests of alternate segment qualities being served from the server and cache leads to overall high quality changes and spectrum. Additionally, high quality segments being fetched all the way from the server also increases rebuffering time stalling overall playback time at the client. We investigated the cause behind this in detail (as discussed in 5.4.2.4) and found that the frequent oscillations led to higher number of unique qualities being requested per segment. This led to poor cache hit-rates, thus negating the benefits of a cache [8, 9]. Hence proving that the interplay of ABR streaming algorithms with network caches can lead to Oscillation effect which adversely affects end-user's QoE.

Therefore, to optimize the end-user's experience in a live video streaming environment that depends on both network caches and adaptive bitrate streaming, we need to identify the occurrences of oscillation effect as well as provide solutions to mitigate them. The solution should not only minimize the adverse affect on end-user's QoE

(rebuffering, #QS, spectrum) but also be dynamically adaptive to changing network conditions.

## 6.2 Methodology

Grandl et al.'s work [20] first pointed out that the involvement of caching in rate adaptation algorithms could lead to significant degradation in cache hit rates and misjudgements by the rate adaptation algorithm causing possible high throughput. This work further proposes a framework suitable for adaptive streaming mechanism for in-network caching where download rates are calculated separately depending on the location from where the content is served (i.e., cache or server). Furthermore, in this framework, the cache modifies its content by transcoding existing content to the quality requested by the client. Such a cache-dependent solution to the oscillation problem assumes cache's transcoding capabilities and comes at the cost of additional processing time to serve requested content per segment of the video. Another work by Lee et. al [33] proposes a cache-based approach that uses traffic shaping to control requested bitrates to reduce bitrate oscillation and sudden rate changes, tested on a single client and video setup. In this work, the cache makes path bandwidth measurements such that client requests don't exceed it. Since cache is where the decision is made, in instances of single point of failure, all clients connected to this cache might still face oscillations. Moreover, the common disadvantage to cache-based decision-making here might be that the client and cache both are involved in bitrate estimation which might lead to stalling and could potentially become a scalability issue. This motivated us to propose a client-based approach that can work in combination with any adaptive bitrate algorithm that the client player chooses for streaming. We propose an algorithm to minimize the oscillation effect that is not only robust to fluctuating network conditions but is also scalable and can be implemented independent of the caching strategy and the live video streaming application. Also,

each client is individually responsible for checking and managing oscillations without interfering with other clients.

### 6.2.1 Proposed dynamic ABR algorithm to reduce oscillation effect

To reduce the effect of oscillations, we follow a two-step approach where 1) we determine the occurrence of the event and 2) take action to minimize its effect. This two-step approach is realized with an algorithm (referred to as  $OR_d$  from this point forward), which is dynamically adaptable to the network conditions and can work as an add-on to any adaptive bitrate algorithm.

#### 6.2.1.1 Identifying oscillation

Oscillation effects are characterized by alternating qualities for consecutive video segments requested by the client. Henceforth to identify the occurrence of oscillations, we define the event of oscillation if either of the following criteria is met:

- Increase in requested segment quality followed by a decrease in requested segment quality
- Decrease in requested segment quality followed by an increase in requested segment quality

For example, during a live video streaming session, if a client requests three consecutive segments in qualities  $q_{i-1}$ ,  $q_i$  and  $q_{i+1}$ , then:

$$\forall i \in [10, n], \textit{Oscillated} = \text{true, iff } q_{i-1} > q_i > q_{i+1} \text{ or } q_{i-1} < q_i < q_{i+1}$$

Where,

$n$  total segment count in the video

$q_i$  = quality of segment number  $i$

$\textit{Oscillated}$  = boolean value denoting the occurrence of oscillation when true.

Furthermore, we experimented with different settings for oscillation detection where we vary the following:

1. *threshold for oscillation count*: We set a threshold for number of oscillations that must occur before we take an action to reduce it.
2. *detection window size*: This is range of segments to detect oscillations within. Since in our case, each video segment is two seconds long, we can also denote detection window size in seconds.

All these decisions are incorporated into an algorithmic form in 6.2.2

#### **6.2.1.2 Reducing oscillations**

Once the live streaming client is aware that oscillation has happened, it takes appropriate steps to reduce its effect. One idea to reduce the oscillation effect has been discussed previously in 5.4.2.4 where oscillations cause too many distinct qualities per segment to be requested leading to low cache hit rates. The naïve client-based solution was to simply reduce the scope of distinct qualities to be requested, which directly increased the cache hit rate and improved the overall QoE. In other words, if the original MPD offered ten qualities per video segment and oscillations caused the clients to request seven distinct qualities out of those ten qualities, then the cache hit rate suffered and benefits from the cache were impeded. Hence, modifying the MPD to offer a smaller amount of qualities (4 qualities instead of 10, chosen through observations made from experiments) led to fewer variation in qualities requested by the clients resulting in fewer cache misses. Consequently, with higher cache hits, we saw an improvement in all QoE factors (AQB, #QS, H, RB, VMAF) reducing the oscillation effect and its negative impact. However, this naïve solution (referred to as  $OR_s$  from this point forward) is a one-time static change made through observations gathered from experimental data and is not adaptive to fluctuating network conditions. Therefore, using this idea of reducing the qualities to be requested on client's

end, we propose a more dynamic version to this solution that adjusts according to the state of the network and also does not require any modifications to the original content offered at the server. We implement this as follows:

- Once an oscillation event has been identified during the live streaming session, the client reports its current average segment quality (or average bitrate) requested so far.
- Based on the reported current average quality, the client proposes a subset of all the qualities originally offered by the server.
- If more oscillations occur during the streaming session, the current mean is recalculated and new quality proposals are made by the client for the segments remaining to be requested or downloaded.

These two components to identify and reduce oscillation effect are implemented in an algorithm called  $OR_d$  as described in the next section.

### 6.2.2 $OR_s$ and $OR_d$ : Static and Dynamic Oscillation effect Reduction

In this section we present the two algorithms  $OR_s$  and  $OR_d$  as algorithms 6 and 7 respectively. While  $OR_s$  was already presented and evaluated in the previous chapter 5, its also compared here with its dynamic counterpart  $OR_d$ . For the sake of brevity, all the possible boundary conditions for  $B'$  are not depicted in algorithm 7.

---

**Algorithm 6  $OR_s$ :** *Static Oscillation effect Reduction* algorithm

---

- 1:  $B \leftarrow$  Set of original bitrates offered at the server via original MPD
  - 2: From experimental data, determine most requested bitrates from set  $B$
  - 3:  $B' \leftarrow$  Reduced Set of bitrates offered at the server via modified MPD
  - 4: Client streams from amongst  $B'$  qualities instead of  $B$
-

---

**Algorithm 7 OR<sub>d</sub>:** Dynamic *Oscillation effect* Reduction algorithm

---

```
1:  $B \leftarrow$  Set of original bitrates offered at the server via original MPD
2:  $n \leftarrow$  Total number of Segments in the video to be livestreamed
3:  $q \leftarrow$  Bitrate quality of a given segment
4:  $osc\_detected \leftarrow$  TRUE if oscillation is detected, else FALSE. Default value = FALSE
5: Ignore the first 10 seconds during startup phase of the streaming session
6: for  $i \in \{15, \dots, n\}$  do
7:   for  $j \in \{i - 4, \dots, i\}$  do
8:     if  $(q_{j-1} > q_j \text{ and } q_j < q_{j+1})$  OR  $(q_{j-1} < q_j \text{ and } q_j > q_{j+1})$  then
9:        $osc\_detected \leftarrow$  TRUE
10:      break
11:     else
12:        $osc\_detected \leftarrow$  FALSE
13:     end if
14:   end for
15:   if  $osc\_detected ==$  TRUE: then
16:     // here, threshold = 1 (see 6.2.1.1)
17:      $m \leftarrow$  current mean average bitrate
18:      $idx \leftarrow$  index of bitrate in  $B$  closest to  $m$ 
19:     Client requests next segments from qualities in  $B'$  instead of  $B$ 
20:     where  $B' = (B[idx-3], B[idx-3], B[idx-1], B[idx])$ 
21:   else
22:     continue
23:   end if
24:    $i \leftarrow i+5$  //when detection window size = 5 segments or 10 seconds (see 6.2.1.1)
25: end for
```

---

It should be noted that in case of OR<sub>s</sub>, if multiple clients are live streaming the same video, they all will start and end streaming from the same reduced set of bitrates  $B'$ , since they share the same MPD available at the server. However, in case of OR<sub>d</sub>, each client continuously detects oscillation per detection window and proposes to stream from a reduced set of bitrates  $B'$  depending on its observed average bitrate. Therefore, if multiple clients live video stream with OR<sub>d</sub>, they might share the same reduced set of bitrates  $B'$  only if they report the same observed average bitrate, and not otherwise.

## 6.3 Evaluation setup

This section describes the experimental setup to test the efficiency of our proposed algorithm as well as explain design choices made in alignment with it. We evaluate our newly proposed dynamic Oscillation effect Reduction algorithms  $OR_d$  with previously proposed static Oscillation effect Reduction Algorithm  $OR_s$  in a live streaming scenario. We also analyze the benefits of adaptive video streaming with our proposed oscillation reduction algorithms ( $OR_s, OR_d$ ) compared to a scenario when no oscillations are detected or reduced.

### 6.3.1 Experimental Setup

We use the experimental setup described in 5.3 to carry out our evaluation experiments. The topology, network setup, video to be live-streamed, video server and client are all described in 5.3.1. The evaluation is done in terms of playback QoE metrics (AQB, #QS, RB, H), also defined in 5.3.3.

### 6.3.2 Design Choice

To fairly compare the live video streaming QoE with and without oscillation reduction and analyze the benefits of one over another, it was imperative that all experiments in question share the same experimental setup. The only design modification we had to make was to the streaming client application that incorporates  $OR_d$ . This was a code design modification made to the bitrate adaptation algorithm (BOLA-O [71], as described in 5.3.1) that had no impact on its function. One behaviour of the BOLA bitrate adaptation algorithm is that it does not allow the next estimated quality to drop below *last seen segment quality*. This sometimes leads the next estimated quality to be set as the *last seen segment quality*. When  $OR_d$  is implemented with BOLA, this estimated *last seen segment quality* might not be in the list of proposed bitrates to be streamed from by  $OR_d$  when oscillation is detected. In such a scenario, we simply include *last seen segment quality* in the list of bitrates

proposed by  $OR_d$  to bypass this contradiction. This code modification retains the original behaviour of BOLA while adapting to the addition of  $OR_d$  to BOLA. This modification is not necessary when no oscillation reduction is done or with  $OR_s$  because in neither cases will there be an estimated quality that will not be present in the list of bitrates to be streamed from. This is because in both these cases, the list of bitrates always remains static and no modification to this list is ever made during the course of live video streaming.

## 6.4 Evaluation Results

The objective of this work is to propose a solution to reduce oscillations for ABR streaming when network caches are involved in the content distribution. We propose an oscillation detection and reduction algorithm ( $OR_d$ ) which dynamically checks for oscillations and makes effort towards reducing them. In this section, we investigate whether live video streaming clients that use ABR algorithms deliver better playback QoE with oscillation detection and reduction or not.  $OR_d$  can not only be used as an add-on to any ABR algorithm but it is also a more practical solution to reducing bitrate oscillations than  $OR_s$ . This is because in case of  $OR_s$ , the oscillation reduction process is static and happens just once, hence cannot adapt to changing network conditions. On the other hand, in case of  $OR_d$ , oscillation detection and reduction happens continuously throughout the streaming allowing it to be more responsive to changes in the network. Additionally, we also study other streaming configuration factors and their effect on oscillation reduction and overall QoE. In this context, we evaluate the following aspects about  $OR_d$ , our dynamic solution to oscillation effects:

- QoE comparison of live streaming with dynamic and static oscillation reduction ( $OR_d$ ,  $OR_s$ ) and live streaming with no oscillation reduction
- Performance of  $OR_d$ ,  $OR_s$  for live video streaming clients with variable bandwidth connectivity.

- Performance of  $OR_d$ ,  $OR_s$  for live video streaming with changing network conditions
- Effect of number of oscillations and detection window size for detecting oscillations in  $OR_d$  on overall QoE
- Effect of the bitrates proposed by  $OR_d$  on overall QoE

We test the proposed algorithms with ABR streaming using an ICN instantiation (NDN) that uses in-network caching capabilities. The caching capabilities of NDN clients offers high average bitrate and visually perceived better quality, however suffers from low cache hits due to oscillations that leads to poor QoE (as detailed in the previous chapter 5.4.2.4). The previous chapter also provided the motivation for this chapter where we observed that reducing offered bitrate qualities directly ( $OR_s$ ) improves the cache hit rates and henceforth improves all the QoE factors (#QS, rebuffering and spectrum). In this section, we propose and analyze the dynamic version of  $OR_s$ , i.e.  $OR_d$ , and its effect on overall live-streaming QoE. Our proposed algorithms  $OR_d$  and  $OR_s$  can be used with any other networking configuration that utilizes in-network caches with ABR streaming.

#### 6.4.1 QoE Analysis of clients live video streaming with and without oscillation reduction

In this section, we aim to evaluate the QoE of live video streaming clients with and without oscillation detection and reduction. Next, we compare the two types of oscillation reduction proposed,  $OR_s$  and  $OR_d$ , namely static and dynamic oscillation reduction where the prior has been studied and evaluated in 5.4.2.4. In this context,  $OR_o$  in table 6.1 denotes no oscillation detection and reduction happening during the streaming other than that introduced in the ABR algorithm BOLA-O [71] used by our streaming client AStream[28]. BOLA-O has inbuilt logic to prevent bitrate

changes frequently in order to avoid oscillations. However, as per our experimental observations, this is not adequate in reducing the effect of oscillations during streaming and our clients streaming with BOLA-O still face significant degradation in QoE due to the same. For clarity,  $OR_s$  and  $OR_d$  clients stream using AStream application which incorporates  $OR_s$  and  $OR_d$  on top of BOLA-O. Hence proving that our suggested static and dynamic oscillations algorithm can work on top of existing ABR algorithm.  $OR_s$  is static oscillation reduction, where oscillation reduction happens only once in the beginning when final bitrates offered at the server are a reduced form of original available bitrates. Since  $OR_s$  is static, the set of reduced bitrates to choose from doesn't change for  $OR_s$  clients during streaming. In such a case, QoE can drastically differ depending on the reduced bitrates. This is demonstrated by the performance of  $OR_s$  and  $OR_s(\text{worst})$  clients indicating the best (ideal) and worst case scenario of streaming with static oscillation reduction.  $OR_d$  indicates dynamic oscillation reduction where clients detect and reduce oscillations continuously during streaming. We also report a Quasi-dynamic oscillation reduction  $OR_{qd}$ , which in other words acts like  $OR_d$  (best) but the bitrate reduction takes place once and is static. As explained in 6.2.2,  $OR_d$  might lead to different clients proposing different set of reduced bitrates, however the best case scenario could be imagined when all the clients would propose the same set causing lesser cache conflicts. To simulate this case, we fix the reduced bitrates set in  $OR_d$  to represent  $OR_d$  (best) scenario. For the remainder of this section, we dive deeper into analyzing the performance of clients streaming with all these above-mentioned algorithms and present the insights we draw from this analysis.

#### 6.4.1.1 QoE With and without Oscillation detection and reduction:

As observed from Tab. 6.1 and Figure 6.1, both  $OR_s$  and  $OR_d$  outperform  $OR_o$  across all QoE metrics. Thus, proving that end-users livestreaming on a network that

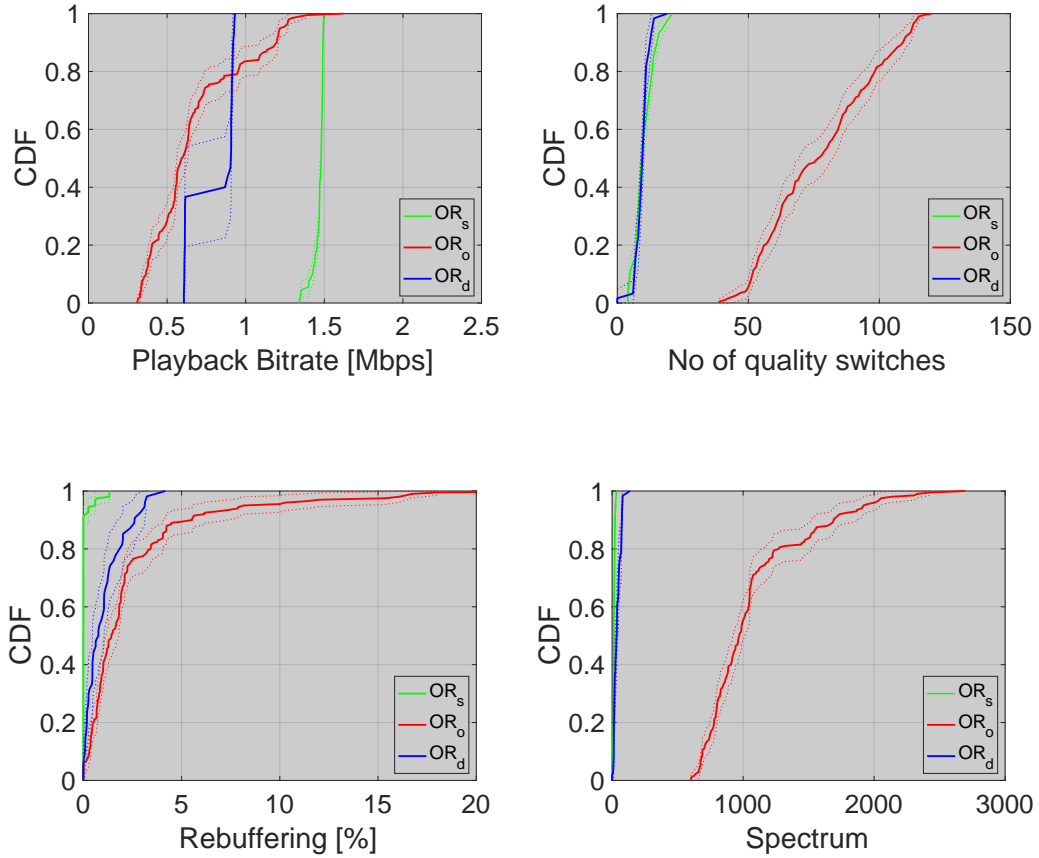
**Table 6.1.** Average QoE metric results from streaming sessions across different experiment setup. 20 NDN clients live streaming with 500MB in-network caches

Rebuf %	#QS	Avg. bitrate	Spectrum
<b>OR<sub>o</sub></b> : No oscillation detection and reduction			
3.6	145.6	0.75	685.3
<b>OR<sub>s</sub></b> : Static oscillation effect reduction			
0.1	10.5	1.47	14.3
<b>OR<sub>s</sub>(worst)</b> : Static oscillation effect reduction			
512.11	0.2	3.39	0.0
<b>OR<sub>qd</sub></b> : Quasi-Dynamic oscillation effect reduction			
0.5	11.3	1.43	44.40
<b>OR<sub>d</sub></b> : Dynamic oscillation effect reduction			
1.05	9.95	0.80	40.29

uses in-network caches benefit if oscillations are detected and reduced during the streaming. This not only leads to fewer #QS and lower spectrum, but fewer Rebuf% and better average bitrate too. The QoE improvement seen in the case of OR<sub>s</sub> and OR<sub>d</sub> are a direct result of increased cache hits caused by oscillation reduction. Table 5.4 already showed how OR<sub>s</sub> (0.12,0.63,0.72) saw much higher cache hits than OR<sub>o</sub> (0.09,0.17,0.21). In comparison, OR<sub>d</sub> clients also reported high cache hit rates of 0.21, 0.47 and 0.68 for the same set of routers in our network. This clearly indicates that end-users will benefit in terms of improved QoE with our static and dynamic oscillation reduction algorithms. It should be noted that OR<sub>s</sub>(worst) simply depicts the QoE performance of OR<sub>s</sub>, if ideal settings were not known beforehand, hence not used in this comparison. We discuss OR<sub>s</sub>(worst) in detail in the next section.

#### 6.4.1.2 Static vs. Dynamic Oscillation detection and reduction:

When we compare the QoE of clients with OR<sub>s</sub> to OR<sub>d</sub>, it is evident that OR<sub>d</sub> clients incur slightly worse QoE performance. Figure 6.1 indicates that OR<sub>d</sub> narrows



**Figure 6.1.** Cumulative Distribution Functions for QoE metrics for the case of oscillation reductions.  $OR_o$ : No oscillation reduction,  $OR_s$ : Static oscillation reduction,  $OR_d$ : Dynamic oscillation reduction

down its playback bitrate requests to the mean and its neighbouring bitrates allowing a smoother playback than  $OR_o$  but also slightly lower average bitrate than  $OR_s$ . This is expected because in case of  $OR_s$ , the ideal bitrates were chosen ahead of time via observations made through experiments on our network setup (see chapter 5 for more detail). Moreover, all the clients streaming from a fixed set of reduced bitrates at the server further minimizes cache conflicts leading to optimum QoE with low #QS, rebuffering, spectrum and high avg. bitrate. Since this represents an ideal scenario for clients using  $OR_s$ , we also refer to this as the best case for  $OR_s$ . However, in practical live video streaming scenarios, it would not be possible to know the ideal bitrates that would lead to optimum QoE experience. For instance, if instead of

four ideal bitrates chosen from the full original set of ten as in the case of  $OR_s$  (see 5.4.2.4 for more detail), the two highest bitrates ( $OR_s(\text{worst})$ ) in the original set (3.33 Mbps, 3.89 Mbps) were offered, then the QoE experience would be much different. The  $OR_s$  (worst case) clients suffer over 500% rebuffering (Table 6.1) which is detrimental to end-user’s playback quality. Since its not possible to predict ideal bitrates beforehand,  $OR_s$  lacks practical application in a network with fluctuating bandwidth capacity. This is where  $OR_d$  comes into play.

As discussed in section 6.2.2, one crucial difference between  $OR_s$  and  $OR_d$  is that all clients live streaming with  $OR_s$  choose from a single set of reduced bitrates that doesn’t change during the streaming session (static) whereas each  $OR_d$  clients continuously proposes their own set of reduced bitrates throughout streaming. This variation in reduced bitrate proposals by all live streaming  $OR_d$  clients is inversely proportional to their cache hit-rates. Thus, highest cache hits in case of  $OR_d$  clients could be achieved if there was no variance in their bitrate proposals, which, practically might or might not happen. To simulate the best case scenario behavior for  $OR_d$ , we proposed  $OR_{qd}$ , a quasi-dynamic oscillation reduction on clients where in the oscillation reduction phase, a fixed subset of bitrates are proposed for any client that faces oscillation. Even though  $OR_{qd}$  sounds dynamic, it really is static to changing bandwidth requirements, because, like  $OR_s$ , all clients choose from the same fixed bitrate set and this set doesn’t change throughout streaming. This is why QoE for  $OR_{qd}$  clients are closer to  $OR_s$  than that of  $OR_d$ . Finally,  $OR_d$  was proposed which allows the client to check for oscillation and then proposes a set of bitrates for rest of the segments to stream from based off of average quality seen so far. Even with the variance in bitrate proposals,  $OR_d$  performs at par with the theoretically ideal scenarios ( $OR_s, OR_{qd}$ ) in terms of #QS and spectrum. This also comes at a slight cost of reduced avg. bitrate and increased Rebuf% because  $OR_d$  can still propose higher bitrate qualities to be fetched than that proposed in the fixed set in  $OR_s$  increasing

the rebuffering time in case of  $OR_d$ . However,  $OR_d$  still remains the only practical solution to solving oscillation effects in a dynamic network environment.

While  $OR_s$  does not detect oscillations, it does effectively reduce them whereas  $OR_d$  both detects oscillations and dynamically adapts to the network by monitoring average downloaded quality and proposing a subset of bitrates based on it. Finally,  $OR_d$  also has negligible but slightly higher rebuffering percentage than  $OR_s$  (as shown in Figure 6.1) owing to the additional processing overhead of oscillation detection and reduction.

#### 6.4.2 Oscillation reduction in live video streaming clients with variable bandwidth connectivity

In this section, we varied the bandwidth connectivity of the clients with their immediate routers to study its effect on the oscillation reduction performance and resultant QoE. Originally all clients were connected to their immediate routers with a 100 Mbps bandwidth connectivity, whereas in this experiment half of them have reduced 10 Mbps connectivity. In table 6.2, we present the QoE of these clients with static and dynamic oscillation reduction.

**Table 6.2.** Oscillation reduction for clients with variable bandwidth traffic.

Algorithm	Rebuf %	#QS	Avg. bitrate	Spectrum
All clients share same bandwidth connectivity(100Mbps)				
$OR_s$	0.1	10.5	1.47	14.3
$OR_d$	1.05	9.95	0.80	40.29
Clients with variable bandwidth connectivity				
$OR_s$	0.1	9.2	1.41	17.97
$OR_d$	0.5	11.4	0.675	54.6

On comparing the metrics with that presented in table 6.1, we observe that the performance of  $OR_s$  and  $OR_d$  does not drastically differ and hence can adapt to varying

bandwidth connectivity scenarios. The slight reduction in average bitrate is obvious as half of the clients now stream at a reduced bandwidth capacity of 10Mbps instead of 100Mbps. Request of lower quality bitrates due to lower bandwidth capacity also reduces the Rebuf% in  $OR_d$  clients. The reduction in last hop bandwidth capacity from 100 Mbps to 10 Mbps does not significantly affect other QoE performance as the network bottleneck is also set to 10Mbps.

### 6.4.3 Comparison of QoE with changing network traffic

In this section, we tested the effect of cross traffic while live streaming clients incorporate  $OR_d$  or  $OR_s$  for reducing oscillations. To test the effect of cross traffic, we introduced static TCP and UDP traffic as well as dynamic TCP cross traffic between the routers (R1-R2 and R1-R3 links shown in 5.3.1) during the course of streaming. In case of static cross traffic, the traffic is presented throughout the entire course of streaming whereas in case of dynamic cross traffic, the cross-traffic is removed halfway through the streaming session. As can be observed from table 6.3, both TCP and UDP cross traffic does not have much adverse impact on the QoE for  $OR_d$  clients because the algorithm can adjust to the changing network conditions. If there is new incoming traffic on the network and bandwidth availability for  $OR_d$  clients drops, their observed average bitrate would drop too and on facing oscillation,  $OR_d$  clients would simply propose reduced bitrates around their new lowered average bitrate. This could lead to a slight drop in average bitrate but it does not affect rest of the QoE metrics. The unaffected quality switches in table 6.3 shows that  $OR_d$  clients can maintain reduction of oscillations even with cross traffic and this shows the adaptability of our proposed algorithm.

On the other hand, QoE of  $OR_s$  clients slightly worsens with both static and dyanmic TCP traffic. Additional traffic can affect the bandwidth and the bitrate qualities at which a client can smoothly stream live video. Since  $OR_s$  clients can only

stream from a fixed set of pre-defined bitrates, if the chosen bitrates are not ideal for the network conditions, it can lead to increased quality switches and spectrum.

With a slight difference in QoE from  $OR_s$ ,  $OR_d$  is capable of consistently delivering this QoE performance with or without competing traffic, unlike  $OR_s$ .

**Table 6.3.** Oscillation reduction with cross traffic.

Algorithm	Rebuf %	#QS	Avg. bitrate	Spectrum
<b>No Cross traffic</b>				
$OR_s$	0.1	10.5	1.47	14.3
$OR_d$	1.05	9.95	0.80	40.29
<b>Static Cross traffic with UDP at 60 mbps</b>				
$OR_s$	0.01	4.12	1.47	4.28
$OR_d$	0.46	9.7	0.70	42.15
<b>Static Cross traffic with TCP at 60 mbps</b>				
$OR_s$	0.04	34.71	0.71	33.45
$OR_d$	0.9	8.8	0.63	32.19
<b>Dynamic Cross traffic with TCP at 60 mbps</b>				
$OR_s$	0.1	26.1	0.992	55.7
$OR_d$	0.6	8.5	0.533	17.9

#### 6.4.4 Effect of oscillation count and detection window size on overall QoE

**Effect of oscillation count:** During a live streaming session with  $OR_d$ , the client keeps a track of number of oscillations detected. The way oscillation detection in  $OR_d$  works, we can keep a count of the number of oscillations detected, and then trigger the oscillation reduction action depending on the count, also referred to here as the "threshold". For instance, we could trigger a reduction action at every instance of detected oscillation, where threshold = 1 or we could trigger a reduction action only after a certain number of oscillations have been detected. In other words, for threshold = 5, we do not take any action for oscillation reduction until 5 oscillations

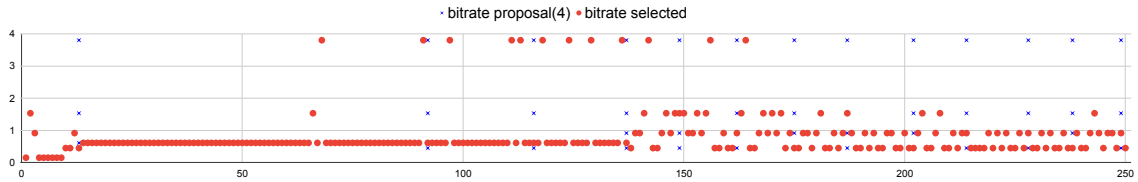
have been detected, and so on. Naturally, we wanted to test the effect of this threshold number (oscillation count) on overall QoE for the clients to understand whether in real life scenarios, oscillations should be reduced at every possible instance or whether some form of threshold should be set for ideal QoE. Interestingly, it was found that the threshold number in itself did not have a huge impact on QoE for NDN clients streaming in our given network. As can be seen from table 6.4, changing the threshold does not necessarily lead to drastic changes in QoE.

**Effect of detection window:** During the process of oscillation detection in  $OR_d$  (as described in Algorithm 7), we can set a window of downloaded segments, for which any instance of oscillation is to be detected. This is done to avoid redundant calculations and adding processing time per segment. For our original experiment, we check for oscillations every 5 segments (10 seconds in time), however, it was necessary to find out if this detection window has any significant impact on overall QoE. Therefore, we experimented with two different detection window settings, i.e. 5 segments and 10 segments. The ideal detection window should avoid missing detection of oscillations such as in large windows as well as avoid redundant calculations such as in small windows. Moreover, this series of experiments was also to find out whether detection window size has any impact on end-user's QoE. As noted from table 6.4, there is an insignificant difference in QoE caused by changing detection window size in clients running  $OR_d$ .

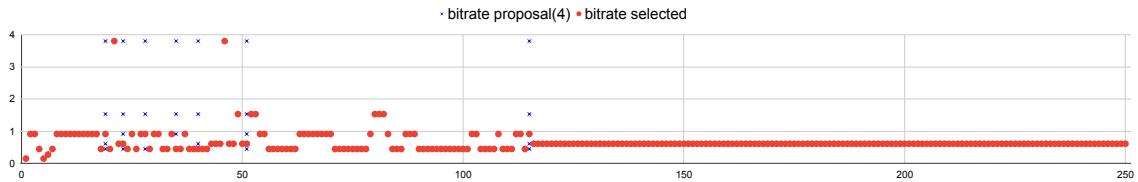
This motivated us to further study underlying factors that might have more significant impact on a client's QoE than oscillation count or detection window. Although oscillation detection is an important step before oscillation reduction, the methodology used for detection had minimal impact towards improving QoE as compared to the reduction process chosen. This is discussed in more detail in the next section.

**Table 6.4.** QoE for clients streaming with OR-d with varying oscillation count as threshold and detection window

Case	Rebuf %	#QS	Avg. bitrate	Spectrum
<b>Effect of varying oscillation count as threshold for oscillation detection</b>				
Threshold = 5 oscillations	0.8	10.33	0.68	47.30
Threshold = 10 oscillations	1.357	11	0.78	52.96
<b>Effect of varying detection window for oscillation detection</b>				
window = 5 segments	1.05	9.95	0.80	40.29
window = 10 segments	1.13	11.12	0.65	71.75



**Figure 6.2.** Client A that reported 117 quality switches and 27% Rebuf %.



**Figure 6.3.** Client B reported 46 quality switches and 6.25% Rebuf %

### 6.4.5 Effect of proposed bitrate qualities on overall QoE

Insights from 6.4.4 led us to investigate other factors that are more significant when it comes to affecting QoE. To do this, we backtracked and looked at the streaming session of live streaming clients with very different QoE. We compared client A that reported high Rebuf% (Fig. 6.2), high #QS and client B that reported significantly lower Rebuf% and lower #QS (Fig 6.3). For both graphs denoting a streaming session, we report video segment number on the X-axis and bitrates proposed per segment as well as the bitrate at which it was downloaded on the Y-axis. One clear distinction

between client A and client B is that in the case of client A, more bitrate proposals per segment included the highest available bitrate. This directly caused more segments to be requested and downloaded from the server at the highest bitrate possible leading to very high Rebuf% for client A. Additionally, we noticed that certain bitrate proposals caused more #QS (as seen in segment number 137 - 250 in Fig. 6.2 for client A). To be precise, if the bitrate proposals didn't include the average bitrate or bitrates closest to the average bitrate, it led to more #QS. All of these observations emphasize that the bitrates proposed by the dynamic oscillation reduction algorithm plays a significant role in terms of client's playback performance (QoE).

**Table 6.5.** QoE for clients streaming with no oscillation reduction  $OR_o$  & dynamic oscillation reduction with varying bitrate proposals for oscillation reduction in the form of  $OR_d$ ,  $OR'_d$  and  $OR''_d$

Bitrate proposal	Rebuf %	#QS	Avg. bitrate	Spectrum
$OR_o$	3.6	145.6	0.75	685.3
$OR_d$	1.05	9.95	0.80	40.29
$OR'_d$	5.95	135.47	0.59	519.96
$OR''_d$	15.7	139.7	0.62	1294.0

This can also be tested with a set of experiments where only thing we change is the way reduced bitrate proposals are calculated on detecting oscillations. In this context, we propose  $OR_d$ ,  $OR'_d$  and  $OR''_d$ , with their reduced bitrate proposal defined as follows:

- $OR_d$  : This previously discussed algorithm proposes reduced bitrates around the observed mean bitrate during streaming
- $OR'_d$  : This algorithm proposes any four bitrates whose average is closest to the observed mean bitrate during streaming, and finally
- $OR''_d$  : This algorithm proposes four bitrate (L,H,X,Y) where L=lowest bitrate, H=highest bitrate, X=closest to mean, Y= closest to average(X,H)

We share their QoE performances in table 6.5 and draw the following insights:

1. Algorithm that proposes highest quality bitrates suffer from very high rebuffering, #QS, spectrum. ( $OR''_d$ )
2. Algorithms that change bitrate proposals too often and randomly across all possible bitrates offered by the server also do not exhibit significant QoE improvement compared to  $OR_o$
3. The mean bitrate observed so far during the course of streaming is a piece of important information that can be leveraged by the client to deal with the oscillation effect. This observation motivated our final dynamic algorithm  $OR_d$  that proposes bitrates closer to the current mean bitrate.

This validates our hypothesis that the choice of bitrates are crucial when oscillations are detected and a subset of original set has to be proposed to reduce oscillations. Furthermore,  $OR_d$  also reports best possible QoE amongst all other cases shown in 6.5 proving that current mean bitrate and proposals around it are essential towards streaming with optimum QoE.

## 6.5 Discussion

### 6.5.1 Deciding on number of bitrates in Bitrate proposals by $OR_d$

The streaming client that streams live video with our proposed dynamic algorithm  $OR_d$  requires proposing a subset of the original bitrates offered by the server to reduce oscillations. It was concluded in 6.4.5 that the choice of bitrates proposed also plays an important role towards the QoE of the clients. Similarly, the size of this subset or numbers of bitrates proposed from the original set also affects the QoE as shown in table 6.6. We conducted an experiment where we used Algorithm 7 but only varied the size of the bitrate proposals from 1-9 since the original dataset available at the server offered upto ten different qualities of bitrates. The results show that bitrate proposals

containing four bitrates yielded optimum QoE. For bitrate proposals of size lower than four, #QS, spectrum, Rebuf% were lower but so was the avg. bitrate. For bitrate proposals of size higher than four, #QS, spectrum and Rebuf% deteriorated. Thus, for our experiments, the bitrate proposals generated by  $OR_d$  selected four bitrate qualities. The results also indicate that a bitrate set reduction to half or close to half **Table 6.6.** QoE for clients streaming with  $OR_d$  with varying bitrate proposal size)

#bitrates in Proposals	Rebuf%	#QS	Avg. bitrate	Spectrum
1	0.73	7.47	0.51	34.26
2	0.57	8.28	0.54	30.57
3	0.46	7	0.6	31.3
4	1.05	9.95	0.80	40.29
5	1.73	46.38	0.74	146.53
6	3.53	100.10	0.77	325.29
7	5.19	142.81	0.60	471.23
8	14.97	99.57	0.72	471.45
9	8.12	160.80	0.59	592.09

of the original size (ten) yields the best results. We could project this information for any live video streaming experiments with a different number of available qualities available at the server, however, further experiments need to be conducted in this context to verify this hypothesis.

### 6.5.2 Alternative to reduced bitrates for oscillation reduction

As detailed in 6.2.1.2, one of the ways oscillations can be reduced is by reducing the choice of bitrates to be streamed from. This choice can be based upon live streaming statistics, for instance, the average quality bitrate downloaded so far. However, there could exist other strategies to reduce oscillations. Currently, when oscillations are detected, all clients make their individual decision of proposing reduced bitrates which could be very different from one another. The more these clients vary in their pro-

posals, the more reduced is the benefit drawn from in-network caches. Alternatively, we propose some server-client based communication strategies to reduce oscillations along with the reason why they were not chosen.

When oscillations are detected, if one client proposes reduced bitrates based on its current average bitrate, it could communicate with the server to change the MPD (to one with reduced bitrates) for the remaining clients for the rest of the streaming session. On the other hand, the first client that detects oscillation and calculates a bitrate proposal could also communicate its proposal to all other clients. Both of these solutions lead to following unwanted outcomes:

- We force other clients to stream at bitrates that are ideal for one client. In a more heterogeneous and realistic network where clients have different bandwidth capacities, we cannot force all clients to stream at similar bitrates.
- Communication between clients during live-streaming is not only unnecessary but also forces all clients to choose from bitrates that are ideal for the earliest client to detect an oscillation.
- Communicating with the server to change its MPD permanently affects clients that are not suffering from oscillation effects.

Because of the above-mentioned reasons, we presented  $OR_d$  as a client-based solution to the problem where no two live streaming clients have to communicate with each other.

## 6.6 Conclusion and Future Work

In this thesis work, we presented how ABR streaming with in-network cache causes oscillations that can impede end-users' QoE as well as proposed a dynamic solution to reduce oscillations.  $OR_d$  is the algorithm proposed to detect and reduce oscillations

dynamically during the course of streaming.  $OR_d$  can act as an add-on to any adaptive bitrate streaming algorithm. The idea behind reducing oscillations is borrowed from  $OR_s$ , a static solution to oscillations provided in the previous chapter. Reducing bitrates decreases the variation in bitrates requested by all clients, thus leading to more cache hits and improved QoE. However,  $OR_s$  is not practical in real-time scenarios as it requires pre-collected information regarding the reduced set of bitrates for ideal oscillation reduction. The non-ideal choice of bitrates can lead to a QoE worse than when oscillations are not handled at all (see  $OR_s(\text{worst})$  in table 6.1). To solve this, we proposed  $OR_d$ , which monitors the average quality downloaded so far and leverages this information to propose reduced bitrates around the current average bitrate. This method not only outperforms clients with no oscillation reduction in terms of QoE but also provides a more practical approach to  $OR_s$ , and can dynamically adapt to the fluctuating network conditions as well as varying bandwidth connectivity. It was also found that the choice of bitrates proposed by  $OR_d$  plays a significant role in affecting playback QoE metrics. From experimental data, it was also concluded that proposed bitrates close to the average observed bitrate deliver optimum QoE.

The future scope of this algorithm could include testing the QoE of clients with varying bandwidth requirements. Also, evaluating  $OR_d$  on a different setup could also validate its robustness.

## CHAPTER 7

### CONCLUSION & FUTURE SCOPE

#### 7.1 Conclusion

With the evolution of high-resolution data being delivered by multiple systems on the cloud, it is of utmost importance that we identify the existing challenges in the process. The end-goal for any cloud service is to optimize the quality of experience for the end-user or customer. The efficiency of a system can be defined by a set of metrics and we can leverage latest technologies to optimize these metrics, which in turn can achieve the end-goal. In this dissertation, we propose our work on improving end-user experience in two different cloud services: weather systems and live video streaming systems. Even though these systems vary in their challenges and their end-goal, we utilize Named Data Networking and virtualization to make improvements in both these systems. We summarize our contribution to optimizing weather and live video streaming system as follows:

##### 7.1.1 Optimizing Flood Forecasting Hydrologic Models

This thesis work presents more readily available cloud VMs as an alternative to HPCs for running hydrologic models that predict flood (HL-RDHM). For this approach to be feasible, the hydrologic model has to perform close to real-time predictions. For faster predictions, we propose parallelization techniques to modify the existing workflow of the HL-RDHM model. This model takes rainfall data as input and outputs discharge and runoff information which is used to calculate the possibility of a flood event. We suggest two parallelization approaches to lower the rainfall

processing runtime: parallel and interleaved approach and propose three new models based on these approach. Our interleaved outperforms all other models and improves rainfall processing time on the cloud up to 34% when compared to the baseline, although parallel model still performs better than the baseline. Even with different data ingest-speed, our model processed rainfall data faster than the baseline, although the improvement decrease with longer intervals of data ingest. We tested our system for robustness and found that our proposed model is consistent with significant runtime improvement across multiple rainfall scenarios. The approach of dividing the region of interest into headwater and downstream basins and parallelizing their workflow made the existing model more efficient. With this thesis work, we improve flood forecasting on the cloud such that these warning systems are not dependent on less readily available and more expensive HPC clusters.

### **7.1.2 Optimizing Weather Sensing using NDN**

In this dissertation, we also improve upon the data retrieval process from CASA radars in DFW region. The accuracy of this data sent to the collection facility ensures accurate warnings for weather hazards. We discuss that existing weather sensing workflow with a sender-driven approach is not suitable as they result in data that is not temporally synchronized and lead to inaccurate predictions. We utilize NDN and present a hierarchical naming scheme for the radar generated data such that a pull-based approach can be implemented where consumer can request the exact required information. With our approach, we demonstrate that NDN's pull based model enables more accurate weather predictions than current time-based windowing mechanism used on top of TCP/IP. We also implement a piggybacking scheme so that clients are up-to-date with any changes on the radars.

### 7.1.3 Optimizing Video Streaming using NDN

In this thesis work, we optimize the quality of performance in video streaming systems. Live video streaming systems need to keep up with the evolution of new technologies and latest customer requirements. There exist multiple Internet architectures suited for live video streaming and in this work, we propose a seamless hybrid architecture that doesn't require immediate replacement of TCP/IP. NDN is a future Internet architecture well suited for live streaming due to its in-network caching and multicast properties. We leverage the benefits of NDN along with SDN, NFV, kernel virtualization and containerization to improve QoE in live adaptive bitrate video streaming applications. With this work, we demonstrate that our hybrid approach can outperform traditional IP in terms of average bitrate. We identify the benefits of using NDN and the challenges of oscillation effect in combination with adaptive bitrate streaming. We propose and test our approach of removing oscillation effect which results in superior playback quality as well as visual quality in NDN clients when compared to traditional IP clients.

### 7.1.4 Optimizing ABR streaming with In-network caching

In this thesis work, we identify the demerits of live video streaming when in-network cache comes into play. We identify bitrate oscillations in ABR clients caused by cache misses that directly impede end-users' QoE. As a result, we propose client-based dynamic oscillation reduction algorithm that can not only reduce these bitrate oscillations but can adapt to fluctuating network environment as well. Our proposed algorithm can be implemented with any adaptation bitrate streaming algorithm. We also show that end-users experience improved QoE with our oscillation reduction algorithm as compared to scenarios when oscillations are undetected and not handled.

## 7.2 Future Scope

Our approach optimizes the rainfall processing on the cloud by HL-RDHM using parallelization. A future scope of this work is to test the generalization and scalability of our approach. Generalization can be tested by extending this concept of distributing the workflow parallelly to similar weather models such as WRF-Hydro that are also readily used by NWS. The scalability of our approach can be tested with a higher-resolution rainfall data over longer duration. Furthermore, our parallelization technique could be used in conjunction with MPI to further distribute our workflow and possibly further decrease processing times.

In the direction of improving weather forecasting, we also better the data retrieval process from CASA radars. For the future scope of this work, we can implement our approach on a real prototype, deploy it in the real-world CASA system and compare the accuracy of our data retrieved.

In the world of live video streaming, it is our goal to advance ABR algorithms that are cognizant of the fact that NDN provides in-network caching. Keeping this in mind, we propose and evaluate strategies to further improve QoE for such an environment in Sect. 5.4.2.4 and 6.4. With the first version of this solution, it would be interesting to see if there are other information during streaming that can be utilized to make best use of in-network caches and can have a greater impact on QoE.

## BIBLIOGRAPHY

- [1] Ahmed, Syed Hassan, Bouk, Safdar Hussain, and Kim, Dongkyun. *Content-Centric Networks: An Overview, Applications and Research Challenges*, 1st ed. Springer Publishing Company, Incorporated, 2016.
- [2] Anderson, Tom, Birman, Ken, Broberg, Robert, Caesar, Matthew, Comer, Douglas, Cotton, Chase, Freedman, Michael J, Haeberlen, Andreas, Ives, Zachary G, Krishnamurthy, Arvind, et al. The nebula future internet architecture. In *The Future Internet Assembly* (2013), Springer, pp. 16–26.
- [3] CASA. CASA radar installation at UT Arlington,. ”<http://www.casa.umass.edu/main/research/UTARlington/>”, 2020.
- [4] Chandrasekar, V., et al. Dfw urban radar network observations of floods, tornadoes and hail storms. In *2018 IEEE Radar Conference (RadarConf18)* (2018), pp. 0765–0770.
- [5] Change, Intergovernmental Panel On Climate. Climate change 2007: the physical science basis. *Agenda 6*, 07 (2007), 333.
- [6] Cisco Systems, Inc. Cisco. 2020. cisco visual networking index report.
- [7] Dasgupta, Ishita. Parallel and staggered HLRDHM code,. ”[https://github.com/ISHITADG/rdhm\\_backup](https://github.com/ISHITADG/rdhm_backup)”, 2020.
- [8] Dasgupta, Ishita, Shannigrahi, Susmit, and Zink, Michael. A hybrid ndn-ip architecture for live video streaming: A qoe analysis. In *2021 IEEE International Symposium on Multimedia (ISM)* (2021), pp. 148–157.
- [9] Dasgupta, Ishita, Shannigrahi, Susmit, and Zink, Michael. A hybrid ndn-ip architecture for live video streaming: From host-based to content-based delivery to improve qoe. *International Journal of Semantic Computing* 16, 02 (2022), 163–187.
- [10] de-las Heras-Quirós, Pedro, Castro, Eva M, Shang, Wentao, Yu, Yingdi, Mastorakis, Spyridon, Afanasyev, Alexander, and Zhang, Lixia. The design of roundsync protocol. *Technical Report NDN-0048, NDN, Tech. Rep.* (2017).
- [11] Docker. Docker. ”<https://www.docker.com/>”, 2020.
- [12] Docker. Host networking. ”<https://docs.docker.com/network/host/>”, 2020.

- [13] Docker. Macvlan networks. ”<https://docs.docker.com/network/macvlan/>”, 2020.
- [14] Duan, Q., Ansari, N., and Toy, M. Software-defined network virtualization: an architectural framework for integrating sdn and nfv for service provisioning in future networks. *IEEE Network* 30, 5 (2016), 10–16.
- [15] Duplyakin, Dmitry, Ricci, Robert, Maricq, Aleksander, Wong, Gary, Duerig, Jonathon, Eide, Eric, Stoller, Leigh, Hibler, Mike, Johnson, David, Webb, Kirk, Akella, Aditya, Wang, Kuangching, Ricart, Glenn, Landweber, Larry, Elliott, Chip, Zink, Michael, Cecchet, Emmanuel, Kar, Snigdhaswin, and Mishra, Prabodh. The design and operation of cloudlab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (Renton, WA, July 2019), USENIX Association, pp. 1–14.
- [16] Fan, Chengyu, Shannigrahi, Susmit, DiBenedetto, Steve, Olschanowsky, Catherine, Papadopoulos, Christos, and Newman, Harvey. Managing scientific data with named data networking. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management* (2015), ACM, p. 1.
- [17] Foundation, Linux. tc. ”<https://linux.die.net/man/8/tc>”, 2020.
- [18] Ghasemi, Chavoosh, Yousefi, Hamed, and Zhang, Beichuan. Far cry: Will cdns hear ndn’s call? In *Proceedings of the 7th ACM Conference on Information-Centric Networking* (New York, NY, USA, 2020), ICN ’20, Association for Computing Machinery, p. 89–98.
- [19] Google. Google Cloud,. ”<https://cloud.google.com/>”, 2020.
- [20] Grandl, R., Su, K., and Westphal, C. On the interaction of adaptive video streaming with content-centric networking. In *2013 20th International Packet Video Workshop* (2013), pp. 1–8.
- [21] Gusev, Peter, Wang, Zhehao, Burke, Jeff, Zhang, Lixia, Yoneda, Takahiro, Ohnishi, Ryota, and Muramoto, Eiichi. Real-time streaming data delivery over named data networking. *IEICE Transactions on Communications* 99, 5 (2016), 974–991.
- [22] Ha, Sangtae, Rhee, Injong, and Xu, Lisong. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008), 64–74.
- [23] Habibi, Hamideh, Dasgupta, Ishita, Noh, Seongjin, Kim, Sunghee, Zink, Michael, Seo, Dong-Jun, Bartos, Matthew, and Kerkez, Branko. High-resolution hydrologic forecasting for very large urban areas. *Journal of Hydroinformatics* 21, 3 (02 2019), 441–454.

- [24] Handigol, Nikhil, Heller, Brandon, Jeyakumar, Vimalkumar, Lantz, Bob, and McKeown, Nick. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2012), CoNEXT '12, Association for Computing Machinery, p. 253–264.
- [25] IBM. What is apache mapreduce. ”<https://www.ibm.com/topics/mapreduce>: :text=Get2020.
- [26] Inc., Apple. Apple HTTP Live Streaming. ”<https://developer.apple.com/streaming/>”, 2020.
- [27] Incorporated, Adobe Systems. Adobe HTTP Dynamic Streaming. ”<http://www.adobe.com/products/hds-dynamic-streaming.html>”, 2020.
- [28] Juluri, Parikshit. Astream. ”<https://github.com/pari685/AStrea>”, 2020.
- [29] Kanada, Y., and Nakao, A. Development of a scalable non-ip/non-ethernet protocol with learning-based forwarding method. In *2012 World Telecommunications Congress* (2012), pp. 1–6.
- [30] Kong, Zhaoning, Ma, Xinyu, Zhang, Yufeng, Zhang, Zhiyi, Pesavento, Davide, Shannigrahi, Susmit, Dulal, Saurab, and Shi, Junxiao. ndn-python-repo. ”<https://github.com/UCLA-IRL/ndn-python-repo>”, 2020.
- [31] Koren, Victor, Reed, Seann, Smith, Michael, Zhang, Ziya, and Seo, Dong-Jun. Hydrology laboratory research modeling system (hl-rms) of the us national weather service. *Journal of Hydrology* 291 (06 2004), 297–318.
- [32] LBL. Singularity. ”<https://sylabs.io/docs/>”, 2020.
- [33] Lee, Danny H., Dovrolis, Constantine, and Begen, Ali C. Caching in http adaptive streaming: Friend or foe? In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop* (New York, NY, USA, 2014), NOSSDAV '14, Association for Computing Machinery, p. 31–36.
- [34] Li, M., Pei, D., Zhang, X., Zhang, B., and Xu, K. Ndn live video broadcasting over wireless lan. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)* (2015), pp. 1–7.
- [35] Li, Zhi, Aaron, Anne, Katsavounidis, Ioannis, Moorthy, Anush, and Manohara, Megha. Toward a practical perceptual video quality metric. ”<https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652>”, 2016.
- [36] Liu, Ronghua, Wei, Jiahua, Ren, Yan, Liu, Qi, Wang, Guangqian, Shao, Songdong, and Tang, Shuang. HydroMP – a computing platform for hydrodynamic simulation based on cloud computing. *Journal of Hydroinformatics* 19, 6 (09 2017), 953–972.

- [37] Lyons, Eric, Papadimitriou, George, Wang, Cong, Thareja, Komal, Ruth, Paul, Villalobos, J.J., Rodero, Ivan, Deelman, Ewa, Zink, Michael, and Mandal, Anirban. Toward a dynamic network-centric distributed cloud platform for scientific workflows: A case study for adaptive weather sensing. In *2019 15th International Conference on eScience (eScience)* (2019), pp. 67–76.
- [38] Mai, H. L., Aouadj, M., Doyen, G., Mallouli, W., de Oca, E. M., and Fester, O. Toward content-oriented orchestration: Sdn and nfv as enabling technologies for ndn. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (2019), pp. 594–598.
- [39] Mair, Michael, Sitzenfrei, Robert, Kleidorfer, Manfred, and Rauch, Wolfgang. Performance improvement with parallel numerical model simulations in the field of urban water management. *Journal of Hydroinformatics* 16, 2 (05 2013), 477–486.
- [40] Martínez, R., Mayoral, A., Vilalta, R., Casellas, R., Muñoz, R., Pachnicke, S., Szyrkowicz, T., and Autenrieth, A. Integrated sdn/nfv orchestration for the dynamic deployment of mobile virtual backhaul networks over a multilayer (packet/optical) aggregation infrastructure. *IEEE/OSA Journal of Optical Communications and Networking* 9, 2 (2017), A135–A142.
- [41] Mastorakis, Spyridon, Afanasyev, Alexander, and Zhang, Lixia. On the evolution of ndnSIM: an open-source simulator for NDN experimentation. *ACM Computer Communication Review* (July 2017).
- [42] Mastorakis, Spyridon, Afanasyev, Alexander, and Zhang, Lixia. On the evolution of ndnsim: An open-source simulator for ndn experimentation. *SIGCOMM Comput. Commun. Rev.* 47, 3 (Sept. 2017), 19–33.
- [43] Matias, J., Garay, J., Toledo, N., Unzilla, J., and Jacob, E. Toward an sdn-enabled nfv architecture. *IEEE Communications Magazine* 53, 4 (2015), 187–193.
- [44] McLaughlin, David, Pepyne, David, V.Chandrasekar, Philips, Brenda, Kurose, James, and et al., Michael Zink. Short-Wavelength Technology and the Potential for Distributed Networks of Small Radar Systems. *Bulletin of the American Meteorological Society (BAMS)* 90, 12 (December 2009), 1797–1817.
- [45] Merkel, Dirk. Docker: lightweight linux containers for consistent development and deployment. *Linux journal* 2014, 239 (2014), 2.
- [46] Microsoft. Microsoft Smooth Streaming, 2020.
- [47] Mohammed, S., and Xie, M. A measurement study on media streaming over wi-fi in named data networking. In *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems* (2015), pp. 543–548.

- [48] Moiseenko, Ilya, and Oran, Dave. Tcp/icn: Carrying tcp over content centric and named data networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking* (New York, NY, USA, 2016), ACM-ICN '16, Association for Computing Machinery, p. 112–121.
- [49] Moya Quiroga, Vladimir, Popescu, I., Solomatine, Dimitri, and Bociort, L. Cloud and cluster computing in uncertainty analysis of integrated flood models. *Journal of Hydroinformatics* 15 (01 2013), 55–70.
- [50] nginx. Reverse proxy caching with nginx. "https://www.nginx.com/resources/wiki/start/topics/examples/reverseproxycachingexample/2020.
- [51] Nguyen, V., Brunstrom, A., Grinnemo, K., and Taheri, J. Sdn/nfv-based mobile packet core network architectures: A survey. *IEEE Communications Surveys Tutorials* 19, 3 (2017), 1567–1602.
- [52] Nour, Boubakr, Li, Fan, Khelifi, Hakima, MOUNGLA, Hassine, and Ksentini, Adlen. Coexistence of icn and ip networks: an nfv as a service approach. In *2019 IEEE Global Communications Conference (GLOBECOM)* (2019), IEEE, pp. 1–6.
- [53] Olschanowsky, Catherine, Shannigrahi, Susmit, and Papadopoulos, Christos. Supporting climate research using named data networking. In *2014 IEEE 20th International Workshop on Local & Metropolitan Area Networks (LANMAN)* (2014), IEEE, pp. 1–6.
- [54] Omnes, N., Bouillon, M., Fromentoux, G., and Grand, O. L. A programmable and virtualized network it infrastructure for the internet of things: How can nfv sdn help for facing the upcoming challenges. In *2015 18th International Conference on Intelligence in Next Generation Networks* (2015), pp. 64–69.
- [55] Pauly, Rini, Ogle, Cameron, Mcknight, Cole, Reddick, David, Presley, Justin, Shannigrahi, Susmit, and Feltus, Alex. NDN-TR68: Utilizing NDN for Domain Science Applications - a Genomics Example - Named Data Networking (NDN), Mar 2021. [Online; accessed 8. Mar. 2021].
- [56] Pfaff, Ben, Pettit, Justin, Koponen, Teemu, Jackson, Ethan J., Zhou, Andy, Rajahalme, Jarno, Gross, Jesse, Wang, Alex, Stringer, Jonathan, Shelar, Pravin, Amidon, Keith, and Casado, Martín. The design and implementation of open vswitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (USA, 2015), NSDI'15, USENIX Association, p. 117–130.
- [57] Quinlan, Jason J., Zahran, Ahmed H., and Sreenan, Cormac J. Datasets for avc (h.264) and hevc (h.265) evaluation of dynamic adaptive streaming over http (dash). In *Proceedings of the 7th International Conference on Multimedia Systems* (New York, NY, USA, 2016), MMSys '16, Association for Computing Machinery.

- [58] Rainer, B., Posch, D., and Hellwagner, H. Investigating the performance of pull-based dynamic adaptive streaming in ndn. *IEEE Journal on Selected Areas in Communications* 34, 8 (2016), 2130–2140.
- [59] Refaei, Tamer, Ma, Jamie, Ha, Sean, and Liu, Sarah. Integrating ip and ndn through an extensible ip-ndn gateway. In *Proceedings of the 4th ACM conference on information-centric networking* (2017), pp. 224–225.
- [60] Ren, Yongmao, Li, Jun, Shi, Shanshan, Li, Lingling, Wang, Guodong, and Zhang, Beichuan. Congestion control in named data networking – a survey. *Computer Communications* 86 (2016), 1–11.
- [61] Roy, Tirthankar, Serrat-Capdevila, Aleix, Valdes, Juan, Durcik, Matej, and Gupta, Hoshin. Design and implementation of an operational multimodel multiproduct real-time probabilistic streamflow forecasting platform. *Journal of Hydroinformatics* 19, 6 (08 2017), 911–919.
- [62] Rukmana, F. B., and Sari, R. F. Performance evaluation of video streaming application via named data network (ndn). In *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)* (2019), pp. 141–144.
- [63] Saltarin, J., Bourtsoulatze, E., Thomos, N., and Braun, T. Adaptive video streaming with network coding enabled named data networking. *IEEE Transactions on Multimedia* 19, 10 (2017), 2182–2196.
- [64] Samain, J., Carofiglio, G., Muscariello, L., Papalini, M., Sardara, M., Tortelli, M., and Rossi, D. Dynamic adaptive video streaming: Towards a systematic comparison of icn and tcp/ip. *IEEE Transactions on Multimedia* 19, 10 (2017), 2166–2181.
- [65] Samain, Jacques, Carofiglio, Giovanna, Muscariello, Luca, Papalini, Michele, Sardara, Mauro, Tortelli, Michele, and Rossi, Dario. Dynamic adaptive video streaming: Towards a systematic comparison of icn and tcp/ip. *IEEE Transactions on Multimedia* 19, 10 (2017), 2166–2181.
- [66] Satria, M. N. D., Ilma, F. H., and Syambas, N. R. Performance comparison of named data networking and ip-based networking in palapa ring network. In *2017 3rd International Conference on Wireless and Telematics (ICWT)* (2017), pp. 43–48.
- [67] Seskar, Ivan, Nagaraja, Kiran, Nelson, Sam, and Raychaudhuri, Dipankar. Mobilityfirst future internet architecture project. In *Proceedings of the 7th Asian Internet Engineering Conference* (2011), pp. 1–3.
- [68] Shannigrahi, S., Fan, C., and White, G. Bridging the icn deployment gap with ipoc: An ip-over-icn protocol for 5g networks. In *ACM SIGCOMM 2018 Workshop on Networking for Emerging Applications and Technologies (NEAT)* (2018), ACM.

- [69] Shannigrahi, Susmit, Fan, Chengyu, and Partridge, Craig. What’s in a name? naming big science data in named data networking. In *Proceedings of the 7th ACM Conference on Information-Centric Networking* (2020), pp. 12–23.
- [70] Sodagar, I. The MPEG-DASH standard for multimedia streaming over the Internet. *IEEE MultiMedia* 18, 4 (April 2011), 62–67.
- [71] Spiteri, K., Urgaonkar, R., and Sitaraman, R. K. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications* (2016), pp. 1–9.
- [72] Spiteri, Kevin, Sitaraman, Ramesh, and Sparacio, Daniel. From theory to practice: Improving bitrate adaptation in the dash reference player. *ACM Trans. Multimedia Comput. Commun. Appl.* 15, 2s (July 2019).
- [73] Suresh, B., Bhat, D., and Zink, M. An evaluation of sdn and nfv support for parallel, alternative protocol stack operations. In *2018 IEEE International Conference on Communications (ICC)* (2018), pp. 1–7.
- [74] Tang, Y., Reed, P., van Werkhoven, K., and Wagener, T. Advancing the identification and evaluation of distributed rainfall-runoff models using global sensitivity analysis. *Water Resources Research* 43, 6 (2007).
- [75] Technical Report. Cisco Systems, Inc. Cisco. 2020. cisco visual networking index (vni) complete forecast update.
- [76] Thelagathoti, Rama Krishna, Mastorakis, Spyridon, Shah, Anant, Bedi, Harkeerat, and Shannigrahi, Susmit. Named data networking for content delivery network workflows. *CoRR abs/2010.12997* (2020).
- [77] Thelagathoti, Rama Krishna, Mastorakis, Spyridon, Shah, Anant, Bedi, Harkeerat, and Shannigrahi, Susmit. Named data networking for content delivery network workflows. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)* (2020), 10.1109/CloudNet51028.2020.9335806, pp. 1–7.
- [78] Thompson, Robert, Lyons, Eric, Dasgupta, Ishita, Mastorakis, Spyridon, Zink, Michael, and Shannigrahi, Susmit. An information centric framework for weather sensing data. In *2022 IEEE International Conference on Communications Workshops (ICC Workshops)* (2022), pp. 1–6.
- [79] Unidata LDM. <https://www.unidata.ucar.edu/software/ldm/>.
- [80] Vilalta, R., Mayoral, A., Casellas, R., Martínez, R., and Muñoz, R. Experimental demonstration of distributed multi-tenant cloud/fog and heterogeneous sdn/nfv orchestration for 5g services. In *2016 European Conference on Networks and Communications (EuCNC)* (2016), pp. 52–56.

- [81] Vilalta, R., Mayoral, A., Muñoz, R., Casellas, R., and Martínez, R. Multitenant transport networks with sdn/nfv. *Journal of Lightwave Technology* 34, 6 (2016), 1509–1515.
- [82] Zhang, Lixia, Afanasyev, Alexander, Burke, Jeffrey, Jacobson, Van, claffy, kc, Crowley, Patrick, Papadopoulos, Christos, Wang, Lan, and Zhang, Beichuan. Named data networking. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 66–73.
- [83] Zhang, Lixia, Afanasyev, Alexander, Burke, Jeffrey, Jacobson, Van, claffy, kc, Crowley, Patrick, Papadopoulos, Christos, Wang, Lan, and Zhang, Beichuan. Named data networking. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 66–73.
- [84] Zhang, M., Lehman, V., and Wang, L. Scalable name-based data synchronization for named data networking. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications* (2017), pp. 1–9.
- [85] Zink, M., Schmitt, J., and Steinmetz, R. Layer-encoded video in scalable adaptive streaming. *IEEE Transactions on Multimedia* 7, 1 (2005), 75–84.