ON IMPROVING THE PERFORMANCE OF MOBILE APPLICATIONS USING THE CLOUD

by

Yuan Feng

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

Copyright © 2013 by Yuan Feng
Abstract

On Improving the Performance of Mobile Applications using the Cloud

Yuan Feng
Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto
2013

As a more efficient means of supplying computing resources in the form of a utility, cloud computing platforms have been increasingly used to meet the insatiable demand from mobile applications. With virtualization and statistical multiplexing, cloud computing platforms are able to provide a much higher level of efficiency when it comes to utilizing computing resources, such as CPU cycles and network bandwidth. The research problems we study in this thesis are in the general research area of mobile cloud computing, as we seek to design and implement new algorithms and protocols that straddle the boundary between mobile applications and cloud computing systems, so that their performance can be jointly optimized to provide the best possible user experience, yet operating within the constraints of available resources and operational costs. We consider mobile applications and cloud computing as two sides of the same “coin,” and this thesis takes both sides into consideration.

From the perspective of mobile applications, we show that interactive applications have the need to stream multi-touch gestures among multiple users, and these streams are broadcast sessions in nature, and are delay-sensitive, bursty, with low bit rates in general. Tailored to the nature of multi-touch gesture streams, we propose a new protocol that uses inter-session network coding to reduce the gesture recognizing delays. Towards supporting mobile applications using the cloud infrastructure and its resources, we believe that mobile applications can benefit from a multi-party video conferencing service based on inter-datacenter networks in the cloud.
We apply intra-session network coding to design a new inter-datacenter protocol to maximize the total throughput of all conferencing sessions in the cloud, subject to a latency constraint imposed by the nature of video conferencing. Our real-world experiments have shown that, the abundant available bandwidth in inter-datacenter networks helps to achieve substantially improved throughput, with very similar delays compared to traditional peer-to-peer solutions. From the perspective of cloud service providers, we study the challenges involved when resource utilization is to be maximized in the datacenters, and when operational costs are to be minimized. To maximize resource utilization, we propose a virtual machine (VM) migration algorithm based on Nash bargaining solutions. To minimize operational costs, we present optimal routing and flow assignment algorithms that route traffic across an inter-datacenter network, with and without store-and-forward capabilities in intermediate datacenters. With efficient and cost-effective utilization of resources in the cloud, and by designing new protocols that are applicable to both mobile applications and cloud computing systems, achieving an optimized level of user experience in a large number of interactive mobile application sessions will become a reality.
To my parents
Acknowledgments

I would like to thank all the people who have helped and inspired me during my studies at the University of Toronto.

First and foremost, I would like to express my deeply-felt thanks to my thesis advisor, Professor Baochun Li, for his thoughtful guidance and strong support throughout my studies towards a Ph.D. degree. I deeply appreciate his strict training and precious advice in all aspects of my academic development. I am also very grateful to Professor Bo Li, at the Hong Kong University of Science and Technology, for giving me instructions and sharing his knowledge. I feel very privileged to have the chance to collaborate with such a great researcher over the past four and half years.

I would like to express my sincere appreciation towards my examination committee members, Professor Hans-Arno Jacobson, Professor Jianwen Zhu, Professor Ben Liang, and Professor Guoliang Xue, for their time and efforts devoted to this thesis.

I would like to thank all my colleagues in the iQua research group, including Di, Chen, Henry, Wei, Zimu, Yuefei, Yiwei, Boyang, Zhi, Jun, Li, and Liyao, for valuable discussions on research and precious friendships that supported me throughout the past four and half years. They make BA 4176 a convivial place to work. I wish them all the best in pursuing their dreams in the future. I also would like to thank my friends Ella, Amber, Jin, Weiwei, Sherry, Yan, and Vladi, for their encouragement and unconditional support. I would like to send a special “thank you” to Zimu, my beloved boyfriend, for his love and tolerance through all these years.

My deepest gratitude goes to my parents for their unflagging love and support throughout my life. I remember their constant support when I encountered difficulties. Their unreserved love has been the most important power that keeps my perseverance throughout the past four and half years. The spirit of this thesis is dedicated to them.
# Contents

1 Introduction

1.1 The Emergence of Cloud Computing ................................. 2
1.2 Motivation: Mobile Cloud Computing ................................. 4
1.3 Main Contributions ...................................................... 6
   1.3.1 Streaming Multi-Touch Gestures across Mobile Devices .......... 6
   1.3.2 Multi-party Video Conferencing as a Cloud Service ............... 8
   1.3.3 Improving Resource Utilization with VM Migration ............... 10
   1.3.4 Minimizing Operational Costs on Inter-Datacenter Traffic .......... 11
1.4 Organization .......................................................... 13

2 Background and Related Work ........................................... 15

2.1 Network Coding ....................................................... 15
2.2 Multi-party Video Conferencing ....................................... 19
2.3 Nash Bargaining Games and Resource Utilization in the Cloud .......... 21
2.4 Multi-Commodity Flow Problems ..................................... 23
2.5 Inter-Datacenter Networks ............................................. 25

3 Streaming Multi-Touch Gestures ........................................ 27

3.1 Overview ............................................................... 27
3.2 Gesture Streaming: Motivation and Quality of Experience ............ 29
   3.2.1 Streaming Multi-Touch Gestures .................................. 30
3.2.2 Quality of Experience ........................................... 32
3.2.3 Presenting Multi-Touch Gestures ............................. 35
3.3 Transporting Gesture Streams .................................... 37
  3.3.1 Design Overview ............................................ 38
  3.3.2 GestureFlow: Protocol Design .............................. 40
3.4 Experiences with GestureFlow .................................. 48
  3.4.1 Performance Evaluation .................................... 49
  3.4.2 A Subjective Evaluation of the User Experience .......... 52
  3.4.3 Performance of Network Coding ............................ 54
3.5 The Problem of Linear Dependence .............................. 58
  3.5.1 Analyzing the Effects of Linear Dependence on QoE ...... 58
  3.5.2 Mitigating Linear Dependence .............................. 60
  3.5.3 Evaluating the Use of the Vandermonde Matrix .......... 61
3.6 Summary ......................................................... 63

4 Airlift: Video Conferencing as a Cloud Service ............... 66
  4.1 Overview ....................................................... 66
  4.2 Motivation and Design Objectives ............................ 69
    4.2.1 Conferencing via the Cloud: Motivation .................. 69
    4.2.2 Airlift: Design Objectives and Choices ................. 71
  4.3 Maximizing Total Throughput in the Cloud .................... 73
    4.3.1 Feasible Paths Satisfying a Delay Bound ................ 73
    4.3.2 The Problem of Maximizing Total Throughput .......... 75
  4.4 Airlift: Protocol Design ...................................... 79
    4.4.1 Transport with Network Coding .......................... 79
    4.4.2 Bandwidth Overhead vs. Delay: A Dilemma .............. 80
    4.4.3 The Protocol at Source and Destination Datacenters .... 82
    4.4.4 Realizing Conceptual Flows with Source Routing ... 84
4.4.5 A Full-Service Broker to Conference Participants .................. 86
4.5 Real-World Implementation and Evaluation ............................... 87
  4.5.1 Airlift vs. Celerity: A Performance Comparison ..................... 88
  4.5.2 Performance with Multiple Aggregated Sessions ..................... 90
  4.5.3 A Packet’s Life: A Microscopic View ................................. 92
4.6 Summary ........................................................................... 94

5 Maximizing Resource Utilization in the Cloud ............................ 96
  5.1 Overview ......................................................................... 96
  5.2 The Problem of Maximizing Resource Utilization ....................... 99
    5.2.1 Benefits of VM Migration ............................................ 99
    5.2.2 Maximizing Resource Utilization in a Generalized Form .......... 102
    5.2.3 Lagrangian Heuristic vs. the Nash Bargaining Solution .......... 106
  5.3 VM Migration Algorithm based on the Nash Bargaining Solution .... 106
    5.3.1 The Nash Bargaining Solution ...................................... 107
    5.3.2 VM Migration in the Bargaining Game .............................. 109
  5.4 Experimental Evaluation .................................................. 115
  5.5 Summary ......................................................................... 120

6 Jetway: Minimizing Costs on Inter-Datacenter Traffic .................. 121
  6.1 Overview ......................................................................... 121
  6.2 Rationale and Challenges .................................................. 125
  6.3 System Model .................................................................... 127
  6.4 Problem Formulation ........................................................ 130
  6.5 Splitting and Routing Flows Optimally to Minimize Costs ............ 132
    6.5.1 Fully Utilizing the Already Paid Portion of Traffic Volume ..... 133
    6.5.2 Minimizing the Additional Cost ..................................... 135
    6.5.3 Implementation Issues ............................................... 138
6.6 Performance Evaluation
   6.6.1 Implementation
   6.6.2 Experiments in the Amazon EC2 Cloud
   6.6.3 Simulation Results
6.7 Summary

7 Postcard: Minimizing Costs with Store-and-Forward
   7.1 Overview
   7.2 Rationale and Challenges
   7.3 Jetway Review: A Flow-Based Approach
   7.4 System Model
   7.5 Formulating the Problem: the First Try
   7.6 Problem Formulation on a Time-Expanded Graph
   7.7 The Time Expansion Approach
   7.8 Performance Evaluation
   7.9 Summary

8 Concluding Remarks
   8.1 Conclusions
   8.2 Future Work

Bibliography
List of Tables

3.1 Examples of multi-touch gestures in GestureFlow. 34
3.2 Examples of multi-touch operations in typical interactive applications. 36
3.3 Comparisons of Gesture Recognizing Delays (msec) using GestureFlow and TCP Relay. 51
3.4 Gesture Recognizing Delays (msec) at Wi-Fi 1 User with New GestureFlow Design. 63
4.1 Conferencing with P2P overlays or via the cloud? A comparison of throughput and end-to-end delay. 71
4.2 Important mathematical notations 77
4.3 Airlift vs. Celerity: a comparison with respect to the total throughput and the maximum end-to-end delay across all 4 sessions in the same conference. 89
5.1 Required resources to accommodate one request in each VM, as well as the resource capacity in each server. 100
5.2 Notations and Definitions 103
5.3 Detailed Information About the Trace 116
6.1 Notations and Definitions 128
6.2 Link Capacities and Costs per Traffic Unit in the Amazon EC2 Inter-Datacenter Network 142
7.1 Notations and Definitions 156
List of Figures

1.1 Cloud computing services: Infrastructure, Platform, and Software as a Service. 4
2.1 Network coding helps to achieve a multicast throughput of 2. 16
2.2 Set of bargaining alternatives for two players and the Nash equilibrium. 21
3.1 A screenshot of running MusicScore on the iPad. 30
3.2 A “picture in picture” design as multi-touch gestures are streamed between two users, who are collaborating to compose the same piece of music. 31
3.3 Bit rates of a typical gesture broadcast session in MusicScore over time. 32
3.4 The replay of gestures from Alice to Bob, using an initial startup delay $\delta$ to mask varying end-to-end delays $\tau_i$ over the Internet, as well as gesture recognizing delays $v_i$ in the application. 35
3.5 Streaming a raw stream of touch objects. 37
3.6 Streaming coded blocks from Node 1 to all other participating nodes along multiple paths. Data blocks from Node 1 are being transmitted to Node 4 in coded form, either using a direct link, or relayed by Node 2 and Node 3 after being recoded with their own data blocks. 39
3.7 The inter-session network coding engine in node 2, in the context of the four-node topology shown in Figure 3.6, with 4 “all-to-all” broadcast sessions considered. 40
3.8 Block 2 \((b_2)\) is decoded after receiving the third coded block, using Gauss-Jordan elimination. ........................................ 41

3.9 A receiver sends a cumulative acknowledgment only when all earlier blocks have been decoded. ......................................................... 42

3.10 The coding window at a source node advances itself over time. .............. 44

3.11 The source node adjusts its maximum coding window size \(W\) when a gesture is recognized by its gesture recognizer. The number in the circle indicates the size of the coding window; and the dark circle indicates that the maximum coding window size has been reached. ........................................ 45

3.12 Removing data blocks from the coding window of a node in its role as a relay (node 4). ................................................................. 47

3.13 *MusicScore* in action: two users are collaboratively composing a musical piece with support from the *GestureFlow* framework. .................. 48

3.14 CDF of gesture recognizing delays between Wi-Fi users using *GestureFlow* and *TCP Relay*. ......................................................... 50

3.15 CDF of gesture recognizing delays between 3G users using *GestureFlow* and *TCP Relay*. ................................................................. 50

3.16 The percentage of blocks from relaying nodes over all received blocks in *GestureFlow* and *TCP Relay*. ........................................ 51

3.17 The average of gesture recognizing delays with different network sizes using *GestureFlow* and *TCP Relay*. ........................................ 51

3.18 Bandwidth overhead per user. .................................................. 53

3.19 User experience ratings of different gesture recognizing delays in *MusicScore* using *GestureFlow*. ........................................ 53

3.20 User experience ratings of different gesture recognizing delays in *MusicScore* using *TCP Relay*. ........................................ 53
3.21 The average delays along with 95% confidence intervals in different experiment settings. .................................................. 54
3.22 A comparison of gesture recognizing delays between GestureFlow with and without the adaptive coding window size adjustment mechanism. .................. 55
3.23 CDF of the number of original blocks to be coded at a node with different choices of the maximum coding window size (W). ........................................ 55
3.24 CDF of the number of relayed blocks to be coded at a node. ........................ 56
3.25 CDF of linear dependence in different settings of the maximum coding window size. .................................................. 56
3.26 The CPU load and memory usage of network coding in an iPhone 3GS device. 57
3.27 CDF of linear dependence with the Vandermonde matrix used at the source nodes (W = 8). .................................................. 62
3.28 Linear dependence with different coding window sizes. ........................ 62

4.1 Packets from User 1 in a 5-party conference are being transmitted in an inter-datacenter network with 5 datacenters, via a combination of direct paths (e.g., D1 \(\rightarrow\) D4) and relay paths (e.g., D1 \(\rightarrow\) D3 \(\rightarrow\) D5). ........................................ 68
4.2 With P2P conferencing, more than the minimum number of overlay links may compete for the same physical bottleneck. ................................. 70
4.3 Conferencing via the cloud: flow rates over the physical bottleneck are aggregated and minimized. .................................................. 70
4.4 Maximizing the total throughput in an inter-datacenter network: an example of the optimal solution obtained by solving the linear program. .................. 78
4.5 Bandwidth overhead vs. decoding delay: a dilemma over the choice of the generation size. .................................................. 81
4.6 The Airlift protocol at the source and each destination: an example. ............ 84
4.7 Three cases of realizing conceptual flows passing through an intermediate datacenter. .................................................. 85
4.8 *Celerity* takes 15 seconds to ramp up its per-session throughput to the steady state of 0.9 Mbps on average. .................................................. 90

4.9 *Airlift* takes less than 5 seconds to ramp up its per-session throughput to the steady state of 3.5 Mbps on average, a three-fold improvement. ................. 90

4.10 Observed throughput values in 10 of the aggregated sessions. .......................... 91

4.11 Observed maximum end-to-end delays from the source to the destinations in 10 of the aggregated sessions. ................................................................. 91

4.12 Observed queue lengths on each hop along a path (Sao Paulo → Ireland → Virginia → Oregon). ................................................................. 93

4.13 Observed decoding delays at both destinations (Oregon and Ireland) over time, measured every 15 seconds. ................................................................. 93

4.14 A packet’s life as measured from the source datacenter (Sao Paulo) to both destination datacenters (Oregon and Ireland). ........................................... 94

5.1 The normalized population of two videos from August 12 to 14, 2008. .......... 97

5.2 The initial fit of three videos on two servers. ................................................. 101

5.3 Improved video fit after VM migration. ......................................................... 101

5.4 Spacial representation of a 2-player game. ................................................... 111

5.5 The utility-distance product of commodities of Player i. ................................ 112

5.6 Finding the pivot point in a lever system. ..................................................... 112

5.7 Finding the pivot point in a 3-player game according to the utility-distance products. ..................................................... 114

5.8 The demand pattern in 200 hours. ............................................................ 117

5.9 Average of improvement on resource utilization ratios with the VM migration algorithm. ............................................................. 117

5.10 Average of improvement on resource utilization ratios vs. the number of requests. 117

5.11 Improvement on the number of requests successfully handled by the datacenter. 118
5.12 Improvement on the number of request successfully handled vs. the total number of requests. ................................................................. 118
5.13 Reduction in standard deviations of resource utilization ratios. .................... 119
5.14 CDF of reductions in standard deviations as a percentage. .......................... 119
5.15 The number of VM migrations after bargaining. .................................... 119
5.16 CDF of the number of VM migrations after bargaining. ............................ 119

6.1 Selecting the best paths for inter-datacenter video traffic in a cloud with 5 datacenters. Based on differing traffic costs on inter-datacenter links and varying transmission rates of videos, the video from Datacenter 2 (D_2) to Datacenter 5 (D_5) is best routed along path \{D_2 \rightarrow D_1 \rightarrow D_5\}, \{D_2 \rightarrow D_3 \rightarrow D_5\}, and \{D_2 \rightarrow D_4 \rightarrow D_5\}, represented by red (light gray) flows; the video from Datacenter 3 (D_3) to Datacenter 1 (D_1) is best routed along path \{D_3 \rightarrow D_2 \rightarrow D_1\} and \{D_3 \rightarrow D_1\}, represented by blue (dark gray) flows. ......................... 124
6.2 How traffic costs can be reduced with optimal routing: a motivating example. 126
6.3 An illustration of the operational costs on link \{i,j\}. ................................. 132
6.4 Jet: architectural components and design. ............................................ 140
6.5 The network topology used in our experiments. ..................................... 141
6.6 Cost per traffic unit with the 100-th percentile charging model. .................. 143
6.7 Cost per traffic unit with the 95-th percentile charging model. .................... 143
6.8 Routing and flow assignment for Video 20 in Jetway. ............................. 144
6.9 Normalized receiving rates for 5 video flows using Jetway. ....................... 145
6.10 CDF of the normalized receiving rates for all video flows using Jetway. ....... 145
6.11 CDF of the end-to-end delays for all video flows using Jetway. ................. 146
6.12 Average end-to-end delays to the Singapore datacenter (6). ..................... 146
6.13 Average percentage of cost reduction with the 100-th percentile charging model. 147
6.14 Average percentage of cost reduction with the 95-th percentile charging model. 147
6.15 Average percentage of flows carried by utilizing the already paid portion of traffic volume with the 100-th percentile charging model. 148

6.16 Average percentage of flows carried by utilizing the already paid portion of traffic volume with the 95-th percentile charging model. 148

7.1 How traffic costs can be reduced with routing and scheduling strategies: a motivating example. 153

7.2 An illustration of the traffic volume to be charged up to time $t$. 159

7.3 An example inter-datacenter network and its corresponding time-expanded graph. 163

7.4 Average costs per $\bar{t}$ with $c_{ij} = 100$ GB/$\bar{t}$ and $\max_k T_k = 3$. 168

7.5 Average costs per $\bar{t}$ with $c_{ij} = 100$ GB/$\bar{t}$ and $\max_k T_k = 8$. 168

7.6 Average costs per $\bar{t}$ with $c_{ij} = 30$ GB/$\bar{t}$ and $\max_k T_k = 3$. 169

7.7 Average costs per $\bar{t}$ with $c_{ij} = 30$ GB/$\bar{t}$ and $\max_k T_k = 8$. 169
Chapter 1

Introduction

As the world migrates to the digital age, millions of people routinely use smart mobile devices, such as tablet computers and smartphones, to create and consume content with mobile applications. These mobile devices, however, are limited in their computing capabilities and energy. Further, they are connected to the Internet sporadically and spontaneously over wireless networks, which are limited in bandwidth availability. On the other end of the spectrum, however, computers with multi-core or many-core processors are readily available off-the-shelf, and can be deployed in large-scale datacenters in the “cloud” as a cloud computing platform, typically provided by a cloud service provider. As a more efficient means of supplying computing resources in the form of a utility, cloud computing platforms have been increasingly used to meet the insatiable demand from mobile applications, and to improve the overall efficiency of using resources.

Our research work in this dissertation is motivated by a close examination of the possibility of using cloud computing platforms for the benefit of mobile applications. The primary goal is to design and implement new protocols and algorithms that carefully take mobile application needs into account, and that optimally allocate resources in the cloud to meet such application needs. In this sense, the new protocols and algorithms we design in this dissertation straddle the boundary between mobile applications and cloud computing systems, as they not only improve
the performance of mobile applications by tapping into abundant resources in the cloud, but also optimize the efficiency and minimize the costs of using such cloud resources. In essence, our work brings the advantage of jointly considering mobile applications and cloud computing to the forefront of research attention, by making important research advances in the emerging area of mobile cloud computing. In this chapter, we introduce the background of our research in the context of cloud computing systems and mobile applications, discuss motivations of our work, present important highlights of our original contributions, and outline the remainder of this dissertation.

1.1 The Emergence of Cloud Computing

Cloud computing has recently emerged as a new paradigm for hosting and delivering services that are resource-intensive, in the dimensions of CPU cycles, network bandwidth, and storage space. Based on the definition given by The National Institute of Standards and Technology (NIST), cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [2]. Such a definition implies that cloud computing platforms are typically provided by cloud service providers, or cloud providers, as a service to enterprises and consumers.

To provide such cloud computing services, these cloud providers operate and manage multiple datacenters that are geographically distributed around the world, and interconnected via inter-datacenter networks that are typically provided by Internet Service Providers (ISPs). Each of these datacenters operate and manage a large number of physical machines, interconnected via high-bandwidth network topologies, typically called datacenter networks. Each physical machine, as a commodity off-the-shelf server in a rack-mounted environment, is further virtualized using virtual machine managers (also called hypervisors), such as Xen
Server [5] and VMWare vSphere [4]. Such virtualization makes it feasible for each physical machine to host multiple virtual machines (VMs) in a multiplexed fashion. As a result of such multiplexing, resources, such as CPU and bandwidth, can be shared and managed by multiple cloud computing services in a more efficient manner.

By migrating from traditional in-house server infrastructures to cloud computing, cloud computing users (also called tenants), such as content service providers, can save a significant amount of up-front investment costs on infrastructure acquisition, as well as ongoing maintenance costs on hardware upgrades and software licensing. This is due to the significant difference between average and peak demand for resources when enterprise applications are hosted and run, such that planning infrastructure acquisition to accommodate peak demand is not economical over the long run. On the other hand, cloud providers are able to charge their tenants for using computing resources on a “pay-as-you-go” basis, and resources are provisioned on-demand. As a result of statistical multiplexing, a cloud provider is also able to meet peak demand from one of its tenants with relative ease, by allocating more resources in the cloud.

Today’s cloud computing services are provided in a layered structure, where each layer can be perceived as a customer of the layer below it. Based on the layer in the architecture that receives focused attention, cloud services can be categorized into three types: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). An architectural overview of cloud computing services and examples of today’s cloud service providers are shown in Fig. 1.1. A SaaS cloud, such as Google Apps, iCloud, and Microsoft Office 365, provides on-demand applications over the Internet, such as word processing and media streaming. With PaaS, cloud computing services are provided as a computing platform and a solution stack, based on an application programming interface (API) that developers can use to develop cloud-based applications. A prime example is Google App Engine, where tenants deploy software services to be run in the cloud using a published Python based API, and Google Inc. supplies low-level services as a cloud provider, such as transparent support for a highly
CHAPTER 1. INTRODUCTION

1.1 Cloud Computing Services

<table>
<thead>
<tr>
<th>Hardware resources</th>
<th>Infrastructure as a Service (IaaS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(CPU, memory, network bandwidth)</td>
<td>(Amazon EC2, Microsoft Azure)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Software as a Service (SaaS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Web services, multimedia)</td>
<td>(Google Apps, iCloud, Microsoft Office 365)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Platform</th>
<th>Platform as a Service (PaaS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Application programming interfaces)</td>
<td>(Google App Engine)</td>
</tr>
</tbody>
</table>

Figure 1.1: Cloud computing services: Infrastructure, Platform, and Software as a Service.

Available and fully managed relational database. Finally, most existing cloud providers provide Infrastructure as a Service (IaaS), including Amazon EC2, Microsoft Azure, and RackSpace Cloud. With IaaS, resources are leased to tenants in the form of Virtual Machines (VMs).

1.2 Motivation: Mobile Cloud Computing

New mobile devices with multi-touch displays, such as smartphones and tablets, have brought revolutionary changes to ways users interact with their devices, and perhaps no less actively, with other users. With mobile applications running on these devices, users interact with one another socially and collaboratively, and such interactions are increasingly conducted in an online fashion in real-time. We believe that application-driven interactions among mobile devices have spurred an upward momentum in mobile traffic volume, and such traffic is mostly multimedia, latency-sensitive, originating from an extensive array of mobile applications. As the total number and the variety of mobile applications continue to increase at an explosive pace, the volume of mobile traffic across mobile devices — as well as between mobile devices and the cloud — will grow with a similar trend.

As the volume of mobile traffic surged in recent years, it is imperative to examine how the performance of mobile applications can be further improved by optimizing the way we
deliver mobile traffic across the Internet, so that users who interact with others in a socially collaborative fashion can enjoy the best possible user experience. Given the need for better performance and the abundant resource availability in the cloud, a natural next step is to turn to possibilities of using cloud computing platforms to improve mobile application performance. We believe that such a combination between mobile applications and cloud computing represents more than just using cloud computing services within mobile applications, by tapping into high-level APIs of SaaS (such as the iCloud) or PaaS (such as the Google App Engine). Instead, the rapid growth of mobile traffic volume demands a more fundamental redesign of protocols and algorithms that transport application data in live sessions across mobile devices, as well as an in-depth examination on how resources are to be optimally utilized in the cloud computing platforms themselves.

Intuitively, any performance improvement would potentially use more resources in the form of CPU cycles and bandwidth, and resources in cloud computing platforms are typically associated with operational costs. We believe that such costs cannot be overlooked when resources are allocated to improve performance; they have to be explicitly considered and accounted for. For example, when inter-datacenter networks are used to carry application traffic, the cloud provider may incur traffic costs that are charged by Internet Service Providers (ISPs), and such costs can easily escalate as the volume of inter-datacenter traffic grows rapidly.

In such an emerging research area of mobile cloud computing, our research presented in this dissertation seek to make progress towards answering a simple opening question: Can we design new algorithms and protocols to be deployed in both mobile applications and cloud computing systems, so that their performance can be improved to provide better user experiences within the budgetary constraints of available resources and operational costs? Research results presented in this dissertation have applied theoretical insights from fields as diverse as game theory, network coding theory, optimization, and combinatorial algorithms, with the objective of designing simple yet practically effective protocols as answers to our opening question.

Despite the fact that our proposed algorithms and protocols are based upon a solid theoret-
ical foundation with rigorous reasoning, one of the highlights of the work in this dissertation is its real-world touch: rather than presenting a theoretical treatment based on assumptions and system models that do not necessarily reflect reality, real-world mobile applications and large-scale software frameworks in the cloud have been implemented and deployed in practice, in order to evaluate the effectiveness and performance of our proposed algorithms and protocols.

1.3 Main Contributions

While we have presented the motivation to our work in this thesis at a high level, such a motivational story is not sufficient. Problems will need to be rigorously defined so that solutions can be pursued. In this dissertation, we begin with a detailed study of mobile applications that involve interactive sessions, and then move on to how the cloud can be used to support multi-party video conferencing sessions among mobile users. From the perspective of a cloud provider, we will examine how resources along multiple dimensions can be optimally utilized via VM migration, and how incurred costs of inter-datacenter traffic can be minimized. Combining our studies from the perspectives of both mobile applications and cloud providers in this dissertation, we make substantial advances towards improving the performance of mobile applications with better utilization of cloud resources.

1.3.1 Streaming Multi-Touch Gestures across Mobile Devices

Multi-touch mobile devices, such as smartphones and tablets, share one common feature: users interact with these mobile devices in the form of multi-touch gestures. It has been widely observed in a large number of mobile applications (especially games) that collaborative interaction among multiple participating users becomes a norm, rather than an exception. To support such collaboration among multiple users in real time, we propose that gestures themselves should be streamed directly in multiple broadcast sessions with an “all-to-all” broadcast nature, with each session corresponding to one of the users as a source of a gesture stream.
Streaming gestures themselves, rather than application-specific data, make it possible to design a new gesture broadcast protocol that can be reused by any gesture-intensive application that needs to support multi-party collaboration. Rather than streaming application-specific data, this is a more elegant solution to serve the needs of an entire category of gesture-intensive interactive applications. Once received, a gesture stream can be rendered in real time by a live instance of the same application on a receiving mobile device. Due to the broadcast nature of such collaborative interactions, multiple gesture broadcast sessions need to be supported concurrently, so that any participating user can be the source of a gesture stream. Such gesture streams are especially interesting as we study how new protocols can be designed to transport traffic generated by mobile applications.

In Chapter 3, we propose *GestureFlow*, a new multi-party gesture streaming protocol specifically designed for multiple concurrent broadcast sessions. Since only a subset of raw touch events can be recognized as multi-touch gestures, the source of a broadcast session needs to send raw touch events, which will be recognized at each of the receivers. Each recognized gesture will consist of a number of network-layer packets, all of which need to be received for the gesture to be correctly recognized. Unlike traditional media streams, gesture streams typically incur low but bursty bit rates. Such low but bursty bit rates, combined with the requirement of receiving and recognizing these streams reliably and with the lowest possible delay, have posed significant challenges to the design of the new protocol.

In order to support multiple broadcast sessions while minimizing gesture recognizing delays and guaranteeing reliable packet delivery, we motivate the effectiveness and practicality of using *inter-session network coding* in the *GestureFlow* protocol, and tackle a number of open challenges in our protocol design, such as what the best size of the coding window should be, how the coding window should be advanced, and how packets from different sessions can be coded together. We present a detailed design that takes advantage of inter-session network coding to support low latencies across multiple broadcast sessions, and address challenges introduced by linear dependence, discovered in our extensive experiments involving a gesture-
intensive iPad application [32].

1.3.2 Multi-party Video Conferencing as a Cloud Service

Our GestureFlow protocol with inter-session network coding is designed to transport low-bit-rate gesture streams in broadcast sessions, but it has not yet taken advantage of available resources in the cloud, with its readily available computing and network bandwidth resources. Naturally, we wonder if we can use the cloud as a service to improve the performance of state-of-the-art mobile applications that are resource demanding. Towards this goal, we first study the feasibility of using the cloud to support mobile applications.

In particular, multi-party video conferencing is one of the most demanding multimedia applications from the perspective of both bandwidth and end-to-end delay constraints. When more users join multi-party video conferencing sessions using their mobile devices, bandwidth limitations in wireless networks may affect the performance of video quality negatively. For example, a direct link between two mobile users may have very poor quality as bottlenecks can exist anywhere in the wireless network. Existing conferencing solutions in the literature have traditionally focused on the use of peer-to-peer (P2P) (e.g., Celerity) or client-server architectures (e.g., Microsoft Lync). Yet, it was found to be difficult for these solutions to provide consistently satisfactory conferencing quality.

On the other hand, nowadays, it is typical for enterprises to rely on services from cloud providers in order to build a scalable platform with abundant available resources to satisfy user demand, and for cloud providers to deploy a number of datacenter inter-connected with high-capacity links, across different geographical regions. One may conceive the possibility of taking advantage of inter-datacenter networks in the cloud to support higher bit rates in multi-party video conferences, yet still maintaining acceptable delays.

In Chapter 4, we promote the use of inter-datacenter networks to support live multi-party video conferencing as a cloud service. Our protocol and real-world implementation, collectively referred to as Airlift, is designed from the ground up to support multiple live conferences.
with an inter-datacenter network operated by a cloud provider. As its name suggests, the unique advantage of Airlift is to provide low-latency end-to-end paths among participants in multiple conferences, yet without the “hustle and bustle” of the public Internet.

Our main research goal in Chapter 4 is to design a new application-layer protocol for inter-datacenter networks, with the objective of maximizing the total throughput in all conferencing sessions subject to delay constraints. To be more specific, its original features are two-fold: First, to be more scalable, our protocol aggregates user-initiated conferences to a smaller number of multicast sessions among datacenters. Second, our protocol is designed to maximize the total throughput across all sessions, while maintaining basic fairness across different conferences, and with stringent delay constraints taken into consideration.

Due to the multicast nature of aggregate sessions, maximizing the total throughput is a NP-Complete problem of Steiner tree packing. With a large number of conferences served concurrently in an inter-datacenter network, packing Steiner trees for each source and in each conference is computationally prohibitive, even with trees of limited depth. Instead of Steiner tree packing, we propose to use intra-session network coding as an integral part in both our protocol design and our real-world implementation. Thinking from the perspective of conceptual flows, the power of network coding lies in its ability of resolving conflicts competing for bandwidth resources in bottleneck links. With the help of conceptual flows, we are now able to form the problem of maximizing the total throughput across multiple multicast sessions as a linear program, which can be solved by any standard LP solver. The optimal solution of the linear program serves as the foundation of our protocol design.

To evaluate and validate the performance of Airlift as a cloud-based multi-party video conferencing solution, we have implemented it in the Amazon EC2 cloud. Our real-world experimental results over PlanetLab and the Amazon EC2 cloud have indicated up to 24 times higher throughput than compared to Celerity, a state-of-the-art P2P conferencing solution, yet with very similar delay performance. To the best of our knowledge, we are the first to present the design and implementation of a cloud service designed to support multi-party video conferenc-
ing sessions from mobile applications [28].

1.3.3 Improving Resource Utilization with VM Migration

We now turn our attention to improving resource utilization within datacenters in the cloud, as an increasing number of mobile applications, such as multi-party video conferencing and on-demand video streaming, are supported by cloud services to achieve an improved level of performance. As better operational efficiency is achieved by multiplexing resources on the same physical server in the cloud using virtualization technologies, migrating virtual machines (VMs) live from physical servers that are overloaded to those that are under-utilized becomes an intuitive solution to handle a flash crowd of requests. With live VM migration, the number of requests being handled at the same time may be effectively increased by migrating VMs with additional resource needs from resource-deficient to resource-rich servers.

Nevertheless, we believe that such migration should be conducted with a well-designed mechanism, in order to fully explore possibilities of utilizing available resources. This is due to the fact that VMs are demanding resources in multiple dimensions — including CPU cycles, bandwidth, and storage space — in a tightly coupled manner. It is usually the case that over-utilization in one dimension leads to under-utilization in other dimensions even as resources are still available, which then leads to a lower efficiency of utilizing resources across all the physical servers in a datacenter. Since resource demands in multiple dimensions are tightly coupled, as VMs are being migrated, utilization of resources may gradually become severely unbalanced across different dimensions, which implies unnecessary waste of available resources.

In Chapter 5, we design an efficient and practical algorithm to maximize resource utilization, with all three dimensions considered, by migrating VMs across servers with awareness of multiple resource dimensions. Using the video streaming cloud service as an example, our study reveals that the challenge of maximizing resource utilization in a VM-based video streaming datacenter is equivalent to maximizing the joint profit in the context of Nash bargaining solutions, as long as utility functions are properly defined. Having servers participating as
players to bargain with each other, and VMs as commodities in the Nash bargaining game, trades conducted after bargaining govern VM migration decisions in each server. Conducted in a laissez-faire manner from time to time, we believe that such a bargaining-based VM migration strategy is helpful to improve the overall resource utilization in the cloud, and ultimately to deliver better performance to mobile applications using these cloud services, as more requests are being handled concurrently [29].

1.3.4 Minimizing Operational Costs on Inter-Datacenter Traffic

In addition to the objective of improving resource utilization within a datacenter, cloud providers have typically deployed multiple datacenters across different geographical regions, and connect them with a dedicated, high-capacity inter-datacenter network to provide better performance. With substantial upfront investments to construct these datacenters, it is certainly to the advantage of cloud providers to minimize operational costs, as inter-datacenter traffic is charged by Internet Service Providers (ISPs). Recent research reveals that traffic costs amount to around 15% of operational costs incurred to a cloud provider, a percentage that is similar to energy costs.

Inter-datacenter traffic is usually charged by the ISPs based on the percentile-based charging model, with which cloud providers pay based on the $q$-th percentile of traffic volumes measured in a short time interval, over a number of such intervals in a charging period. We believe that it is feasible to reduce or even minimize cloud providers’ costs by designing new routing and flow assignment algorithms for inter-datacenter traffic. This is motivated by the observation that, with the percentile-based charging model, if traffic has already been generated during one time interval, up to the same volume of traffic may be carried free of charge in subsequent time intervals.

Again, we take inter-datacenter video traffic as an example. In Chapter 6, we proposed Jetway, a new set of algorithms designed to minimize operational costs on inter-datacenter video traffic in an efficient and simple way. To guide the design of our algorithms in Jetway,
we present a methodical and in-depth analytical study on how inter-datacenter video traffic costs are to be minimized by routing video flows via multiple multi-hop paths in an optimized fashion. With Jetway, video flows across inter-datacenter links can be split and transmitted along multiple multi-hop paths, each of which can be optimally and dynamically computed over time. It takes full advantage of our key observation that some of the traffic volumes can be transferred free of charge, while taking into account practical constraints of limited link capacities, as well as different desired transmission rates of videos, representing their delay tolerance. Our algorithm will split and route the video flows in an online fashion by solving two subsequent minimum-cost multicommodity flow and maximum concurrent flow problems.

To evaluate the performance of Jetway towards minimizing costs on inter-datacenter video traffic, we have again implemented it in the Amazon EC2 cloud, along with extensive performance evaluations using simulations. Both our real-world experimental results and simulations have shown that, by routing video flows optimally, Jetway is capable of reducing costs on inter-datacenter video traffic. It is important to emphasize that, although we use video streaming datacenters and video inter-datacenter traffic as examples in our discussions on maximizing the resource utilization ratio in the cloud and minimizing costs on inter-datacenter traffic, respectively, our solutions can be readily extended to more general scenarios [30].

Last but not the least, beyond the flow-based model that we have used in Jetway, we have also explored a store-and-forward approach in minimizing the costs incurred by inter-datacenter traffic, and we call this approach Postcard, presented in detail in Chapter 7. In Postcard, intermediate nodes are able to store incoming traffic flows, and to forward them at a later time to reduce peak traffic demand. Postcard is formulated as a tractable convex optimization problem with linear constraints only (e.g., on link capacity and traffic conservation), to be solved with a standard solver. The key idea in the design of Postcard is to construct a time-expanded graph over multiple time intervals, where a data file starts its transmission at the beginning of a time interval, and finishes at the end of it.

With extensive simulations, we compare our results from solving Postcard to those from
solving Jetway, and present the advantages and drawbacks of store-and-forward when it comes to minimizing costs on inter-datacenter traffic. *Postcard* represents a first attempt in the literature to systematically study and formulate the problem of minimizing operational costs on inter-datacenter traffic, with the ability of intermediate nodes to store incoming data and forward them at a later time [31].

### 1.4 Organization

The remainder of this dissertation is organized as follows.

In Chapter 2, we will first present the necessary theoretical background that is needed for presenting our contributions, along with a detailed literature review in the area of mobile applications and cloud computing related to this dissertation.

In Chapter 3, we will take the perspective of mobile applications. We first discuss the motivation and challenges of streaming multi-touch gestures among users, and then describe our detailed system design in *GestureFlow*, using inter-session network coding. After that, we present a thorough analysis of measurement results using our implementation of *GestureFlow*. From our measurement results, we find out that coded blocks are linearly dependent with an alarmingly high probability. To mitigate such linear dependence, we propose a solution to use systematic Reed-Solomon codes at each source, and evaluate the performance of our revised solution.

In Chapter 4, we begin with the motivation of providing multi-party video conferencing as a cloud service, and discuss our design objectives and choices. Serving as the foundation of our protocol design, we present our analytical study on maximizing the total throughput in an inter-datacenter network using intra-session network coding, with constraints on end-to-end delays. We then present details of our protocol, called *Airlift*, designed based on results from our theoretical analysis. Finally, we present our real-world implementation of *Airlift* and evaluate its validity and performance in the Amazon EC2 cloud.
Starting from Chapter 5, we switch gears to optimizing resource utilization and minimizing operational costs in the cloud, taking the perspective of cloud providers. In Chapter 5, to maximize the resource utilization in datacenters, we first formulate the challenge of optimal VM migration into an optimization problem, and motivate the use of Nash bargaining solutions to solve this problem. We show that the formulation is equivalent to the joint profit maximization problem in a Nash bargaining solution, and describe the VM migration algorithm. The effectiveness of our VM migration algorithm is validated by an in-depth simulation-based analysis, driven by real-world traces from UUSee Inc.

In Chapter 6, we focus on the challenge of minimizing operational costs on inter-datacenter traffic, which are charged by the ISPs. We take inter-datacenter video traffic as an example. We first formulate the optimal routing and flow assignment problem in an online fashion, and then propose algorithms that seek to minimize the traffic costs by splitting and routing video flows optimally. Similar to Airlift, we have implemented Jetway in the Amazon EC2 cloud, and evaluate its performance using both real-world experiments and simulations.

In Chapter 7, we present a new solution on minimizing inter-datacenter traffic costs, based on the ability for intermediate datacenters to store-and-forward traffic. In our alternative model, intermediate nodes have the ability of storing incoming data, and of forwarding it to other datacenters during later time periods. To formulate a tractable problem, we construct a time-expanded graph for the inter-datacenter network, which transforms the problem into a traditional convex problem that can be solved by a standard solver. We compare our simulation results between Postcard and Jetway, and discuss the benefits and drawbacks of the store-and-forward model in detail.

Finally, Chapter 8 summarizes the conclusions of this dissertation. It ends with a brief outlook and directions to potential future work in the general area of mobile cloud computing.
Chapter 2

Background and Related Work

In order to improve the performance of mobile applications with cloud services, we have re-sorted to a variety of theoretical tools in this dissertation, including inter-session and intra-session network coding, Nash bargaining games, and multi-commodity flows. In this chapter, we present a brief survey of the necessary background and related work in the literature, in order to facilitate a better understanding of the remainder of this dissertation.

2.1 Network Coding

Network coding has attracted researchers from information theory and data networking in the past decade of research. The essential idea is that it allows an intermediate node in the network to perform coding on the received data blocks, rather than just simply forwarding (routing) and replicating them. Data blocks, therefore, can be “mixed” during the course of routing. In contrast to source coding, the encoding and decoding operations in network coding are not restricted to sources and destinations only, and may occur at any node across the network. It has been shown that network coding maximizes multicast throughput, as compared to routing only.

More formally, it has been proved that for a given multicast session in a directed network with network coding, if a unicast rate $x$ is feasible from the sender to each receiver indepen-
dently, then it is feasible as a multicast rate to all the receivers simultaneously [10]. What this implies is that network coding is able to maximize the throughput in multicast sessions. The power of network coding can be appreciated with the following classic example, using a simple “butterfly” network topology. Shown in Fig. 2.1, there is a single multicast transmission session from the sender to two receivers simultaneously, and the remaining four nodes are relay nodes. Assuming each link in the network has the same unit capacity of 1, a multicast transmission scheme with network coding is shown in the figure. Here $a$ and $b$ are two data flows, each with a rate of 1. Replication occurs at nodes $A$, $B$, and $D$. Coding takes place at the relay node $C$, which computes the bitwise exclusive-or of $a$ and $b$, i.e., it computes $a + b$. When receiver 1 receives $a$ and $a + b$, it is able to recover $b$. Similarly, receiver 2 can recover $a$ by receiving $b$ and $a + b$. Both receivers are therefore receiving data packets at a rate of 2, leading to a multicast throughput of 2. It is not difficult to find out that without network coding, the throughput of 2 cannot be achieved.

With the inception of network coding [10] and random network coding [24, 43] in information theory, the topic has attracted a substantial amount of research attention. Analytical studies [10, 43] have shown that network coding is able to maximize information flow rates in multicast sessions in direct acyclic graphs. In more practical systems, Gkantsidis et al. [37, 38]
have shown that the use of *random network coding* in peer-to-peer file sharing systems can reduce the time to download files. Annapureddy *et al.* [13, 14] have evaluated the use of network coding in experimental peer-to-peer on-demand streaming systems, and have shown that network coding helps to achieve good performance with respect to the sustainable playback rate and system throughput. UUSee, Inc. has successfully adopted network coding within its commercial peer-assisted on-demand streaming protocol [54].

With random network coding, a generation of $k$ original data blocks (called the generation size) $b = [b_1, b_2, \ldots, b_k]^T$, each with a fixed number of $s$ bytes, are to be transmitted from the source to multiple receivers in a network topology. A source of a broadcast session generates coded blocks $x_j = \sum_{i=1}^{k} c_{ji} \cdot b_i$ as a linear combination of original data blocks in a finite field (typically $GF(2^8)$), where the set of coding coefficients $c_j = [c_{j1}, c_{j2}, \ldots, c_{jk}]$ is randomly chosen. A relay node is able to perform similar random linear combinations on received coded blocks with random coefficients. Coding coefficients related to original blocks $b_i$ are transmitted together with a coded block, typically in its header. If recoding at intermediate relay nodes is restricted to blocks belonging to the same broadcast session, it is referred to as intra-session network coding; otherwise, if there is no restriction on which session the recoded blocks come from, it is called inter-session network coding.

A receiver is able to decode all $k$ data blocks when it has received $k$ linearly independent coded blocks $x = [x_1, x_2, \ldots, x_k]^T$, either from the source or from a relay. It first forms a $k \times k$ coefficient matrix $C$, in which each row corresponds to the coefficients of one coded block. It then decodes the original blocks $b = [b_1, b_2, \ldots, b_k]^T$ as $b = C^{-1}x$. Such a decoding process can even be performed progressively as coded blocks arrive, using Gauss-Jordan elimination to reduce $C$ to its reduced row-echelon form (RREF). The linear independence of received coded blocks are critically important, as the inversion of $C$ is only possible when its rows are linearly independent, i.e., $C$ is full rank.

Different from existing work in the literature where intra-session network coding has been used [13, 14, 54], in Chapter 3, we choose to use inter-session network coding in our design of
a new broadcast protocol for streaming gestures with reliable delivery and short delays. From
a more theoretical perspective, inter-session network coding has drawn some recent research
attention in the literature. Eryilmaz et al. provided a theoretical framework in which a dynamic
routing-scheduling-coding strategy is proposed to decide whether blocks from two sessions
should be coded together at a node [27]. Yang et al. proposed to divide multiple sessions
into groups and to construct a linear network code for each group, with the goal of improv-
ing the system’s benefits on bandwidth and throughput [81]. Focused on directed networks
with two multicast sessions, Wang et al. discussed various aspects of pairwise inter-session
network coding, including the sufficiency of linear codes and the complexity advantages of
identifying coding opportunities [75]. I²NC combined inter-session and intra-session network
coding to improve the throughput in lossy wireless environments [69]. In contrast to previ-
ous research, the primary objective of applying inter-session network coding in Chapter 3 is
to reduce the gesture recognizing delay as a Quality of Experience (QoE) metric in interactive
multimedia applications, which has not been the focus of study in previous work. We will show
how inter-session network coding achieves reduced delays in all-to-all broadcast sessions, with
reasonably low bit rates in each session.

In Chapter 4, we have resorted to intra-session network coding in our Airlift protocol to
maximize the total throughput of multiple video conferencing sessions in an inter-datacenter
network, which are both bandwidth demanding and delay sensitive. In Airlift, intra-session net-
work coding is used at the transport layer of the protocol stack, supporting multiple concurrent
multicast sessions.

With respect to the design of transport protocols, RTP [67] and RTSP [68] were able to
provide one-to-all delivery of data with real-time properties over IP multicast. Since RTP
and RTSP were originally designed for IP unicast, reliable multicast protocols [35, 61] were
proposed to improve the performance of multi-party streaming, by reducing ACK/NAK implo-
sion in back traffic and optimizing retransmissions on multicast channels. In addition, intra-
session network coding has been applied to the design of new transport protocols. For exam-
ple, Chachulski et al. [20] have proposed MORE, which improved the unicast throughput in the context of wireless opportunistic routing by using intra-session network coding. CodeCast, presented by Park et al. [60], was a network coding based multicast protocol for low-latency multimedia streaming. Sundararajan et al. [72] proposed a modified acknowledgment mechanism to incorporate network coding into TCP, with the objective of providing better support to unicast sessions. In their solution, the number of blocks involved in the sender’s sliding window (i.e., the generation size) was completely controlled by TCP. The receiver acknowledges the degree of freedom of the coefficient matrix of coded blocks received so far. Network coding was used in a separate underlying layer as a rateless erasure code, and was decoupled from window-based flow control in TCP.

In contrast, when we design Airlift as a transport protocol in Chapter 4, we have considered a number of more practical challenges that emerged from our experiments, all of which were not discussed in the literature. For example, challenges of determining the appropriate size of a generation to be used in network coding has prompted us to design a new protocol that saturates the delay-bandwidth product between the source and its destinations of a multicast session, yet without adding significant decoding delays.

2.2 Multi-party Video Conferencing

In Chapter 4, we seek to provide a cloud service to support multi-party video conferencing in mobile applications. In the literature, multi-party video conferencing solutions were typically designed in a peer-to-peer (P2P) fashion, due to the lack of scalability of using a server-centric architecture.

The most recent (and most elaborate) P2P conferencing solution is Celerity [22], which held the view that bandwidth bottlenecks may not be limited to the last-mile uplink bandwidth, and instead may reside in the core of the Internet. In this spirit, Celerity represented an advance as compared to other existing works on P2P conferencing protocols in the litera-
ture [21, 50, 52, 62], which sought to optimize the aggregated utility of conference participants, as well as the utilization of peer uplink bandwidth. Designed as a pure P2P solution, Celerity eliminates the existence of centrally administered servers. In contrast to commonly assumed P2P scenarios where bandwidth bottlenecks reside only at the edge of the network, Celerity made the assumption that available network resources are over arbitrary network topologies where bottlenecks can be anywhere in the network. It proposed a distributed and adaptive rate control protocol, which can help the source in each broadcast session to discover and adapt to network topologies and various network conditions in an iterative fashion.

In contrast, the design objective of the Airlift protocol in Chapter 4 is to provide video conferencing as a cloud service that routes video flows through the inter-datacenter network in the cloud. In contrast to Celerity that focuses on each broadcast session in just one conference, our protocol design supports multiple conferences by aggregating video flows to aggregated sessions (each with its own subset of datacenters). Granted, there exist previous works in the literature (e.g., Liang et al. [52]) that considered multiple conferences (called swarms) in a P2P system, and focused on optimal bandwidth sharing of resources at both peers and helpers using a utility maximization framework. Since Airlift handles aggregated video traffic, it is designed to maintain basic fairness among all the video sources with respect to their flow rates, without the use of utility functions.

Network coding has been used in both Airlift in Chapter 4 and Celerity. It was used as a rateless erasure code in Celerity [22], and detailed implementation challenges have not been addressed. In Airlift, network coding serves as the foundation throughout our problem formulation and our protocol design. Only with network coding can we design a new protocol that achieves throughput optimality in theory, yet lends itself to a feasible real-world implementation.
2.3 Nash Bargaining Games and Resource Utilization in the Cloud

In a Nash bargaining game, participating players are performing barter trade such that their utility gains are maximized. Bargaining considers the situation when multiple players are in a position to barter goods but have no money to facilitate the exchange [57]. Take a simple two-player game as an example. A compact convex metrical space \( S_i \) of mixed strategies \( \theta_i \) pertains to player \( i, i \in \{1, 2\} \). These mixed strategies represent the courses of action the player can take independently of the other players. The randomization process of establishing all possible strategy alternatives illustrates the possible joint courses of action by the players. This set of alternatives can be represented by a convex polytope in the plane with the dimensions of utility gains for the players, which is shown in Fig. 2.2. For each pair of mixed strategies \((\theta_1, \theta_2)\) from \((S_1, S_2)\), the payoffs for the deployment of these strategies are denoted by \( \Phi_1(\theta_1, \theta_2) \) and \( \Phi_2(\theta_1, \theta_2) \), respectively, which corresponds to a point in the convex polytope in Fig. 2.2.

![Figure 2.2: Set of bargaining alternatives for two players and the Nash equilibrium.](image)

An outcome is in equilibrium if there is no other possible agreement that allows both players to have higher payoffs simultaneously, and the barter trade such that maximum utility gain \( G_i \) is achieved, which is also known to be Pareto-optimal. Nash has shown in his seminal paper that obtaining the maximum of the product of the two utility gains from the set of alternatives,
known as the Nash product $G_1 G_2$, will attain the Pareto-optimal solution for the bargaining game [57]. As shown in the figure, the super set of alternatives results in a convex polygon whereby the product of maximum utility gains is maximized at its vertex. In practice, players take the action that optimizes the Nash product $G_1 G_2$.

In this dissertation, we seek to explore possibilities of using Nash bargaining game as an approach to tackle virtual machine (VM) exchanging problems in large-scale datacenter networks, where utilities are generated to model each participating player’s satisfaction. Intuitively, if we consider one or a collection of physical servers in the datacenter network as participating players and VMs to be exchanged as commodities, the datacenter network can be envisioned as a bargaining game market. Having each player to form its utility based on its preference to that commodity, barter trades are dynamically controlled by these preferences. If the utility of a player is defined intelligently, trading conducted to maximize the Nash product in the Nash bargaining game can result in “optimal” VM exchange results naturally in a distributed fashion.

Researchers are starting to find possibilities of moving video streaming services to virtualized cloud platforms. For example, Aggarwal et al. investigated the potential of utilizing virtualization to deliver IPTV services, while their focus is on how the number of servers required is efficiently minimized, with only the deadline constraint considered [9]. Prior to direct video streaming, video transcoding is commercially available in cloud environments. For example, HDCloud and Encoding have provided flexible but proprietary cloud based video transcoding services integrated with Amazon EC2, S3, and CloudFront CDN services [36].

There exist research results that showed how VMs are to be migrated to alleviate “hotspots” in datacenters. Wang et al. proposed an autonomic provisioning framework, so that resources can be dynamically provisioned to different applications according to their demand on CPU capabilities [76]. Meng et al. presented a placement algorithm that focuses on bandwidth consumption of each VM [56]. Unlike prior works that only considers a single resource dimension, this dissertation addresses challenges of maximizing resource utilization across multiple
resource dimensions, which is much more challenging.

To our knowledge, there exist two papers that discussed resource challenges along multiple dimensions in datacenters. Korupolu et al. proposed a placement algorithm by applying the theory of stable matching, taking storage, bandwidth, and CPU resources into account [46]. Our work in Chapter 5 differs in that the problem they addressed is a static placement problem, in a sense that once VMs are placed, they are not migrated over time. Singh et al. [70] described VectorDot, a load balancing algorithm for handling multi-dimensional resource constraints in datacenters. However, their objective was to alleviate an overloaded server as much as possible, while ours is to improve the utilization ratios across the board, as required resources increase in an unbalanced fashion and may negatively affect the utilization of available resources in the datacenter.

2.4 Multi-Commodity Flow Problems

Flow networks are widely studied to formulate and solve the network resource management problem. Imagine a commodity, e.g., data blocks, flowing through a system from a source, where the commodity is produced, to a destination, where it is consumed. The source produces the commodity at some steady rate, and the destination consumes the commodity at the same rate. We can interpret the “flow” of the commodity at any point in the system is intuitively the rate at which the commodity moves. When considering this flow network, we can think of each directed link as a conduit for the commodity. Each conduit therefore has a stated capacity, given as a maximum rate at which the commodity can flow through the conduit. Vertices are conduit junctions, and other than the source and the destination, the commodity flows through the vertices without being buffered in any of them. In other words, the rate at which commodity enters a vertex must equal the rate at which it leaves the vertex. This property is called “flow conservation,” and is equivalent to Kirchhoff’s current law when the commodity is the electrical current [26].
In the most basic maximum flow problem, we seek to obtain the highest rate at which we can ship the commodity from its source to its destination, without violating any capacity constraints. Now imagine that there are multiple such commodities in the network, each considered to be unique, referred to as multi-commodity flows. The multi-commodity flow problem considers a directed graph with a capacity constraint and a cost on each link. There is a collection of commodities, each characterized by the following characteristics: the source (where the flow is originated), the destination (where the flow consumed), and the demand (the amount of flow desired to be transferred). Formally, we use \( G = (V, E) \) to denote the directed network, a nonnegative capacity function \( u: E \rightarrow R \), and a specification \( s_i, t_i, d_i \) for each commodity \( i \), where \( s_i \) and \( t_i \) state its source and destination node, and \( d_i \) represents the demand of commodity \( i \). What the flow conservation property states is that a flow \( f_i \) of commodity \( i \) should satisfy

\[
\sum_{(u,v)\in E} f_i(u,v) - \sum_{(v,w)\in E} f_i(v,w) = \begin{cases} 
  d_i & \text{if } v = t_i \\
  -d_i & \text{if } v = s_i \\
  0 & \text{otherwise.}
\end{cases}
\]

The reason we resort to multi-commodity flow problems in this dissertation is that they are classic combinatorial optimization problems, and they directly address a number of practically important issues of bandwidth allocation in a networking environment. For example, if we assume that there is infinite demand, and we need to decide what fraction of the demand can be maximally admitted to go through a network with the link capacity constraints considered, it will be a multi-commodity maximum concurrent flow problem, where the objective is to maximize the sum of all feasible flows in the network. If we take the link cost into consideration, \( e.g., \) prorogation delays on a network link or the price per unit of transmission, and wish to decide how to route all the demands to minimize such cost, it will be a minimum cost multi-commodity flow problem, where the objective is to minimize the total cost of all feasible flows. As we will show in Chapter 6 and Chapter 7, we use the multi-commodity flow formulation to solve the traffic cost minimization problem in inter-datacenter networks.
2.5 Inter-Datacenter Networks

Geographically dispersed datacenters have attracted a significant amount of recent research attention, as they have brought forth a number of new opportunities and challenges. Most recent works focused on dynamic load distribution across datacenters, with the objective of minimizing cloud providers’ operational costs on energy or minimizing the cloud users’ performance penalty. Wu et al. proposed to migrate social media applications into geo-distributed clouds operated by one or more cloud providers, and designed an online content migration and request distribution algorithm [80]. Rao et al. studied the problem of minimizing the total electricity costs in a cloud under multiple electricity markets, with guaranteed quality of service [64]. DONAR, a distributed system, was proposed to direct cloud users’ requests to the best replica location, based on the datacenters’ current loads and the users’ performance penalty [77]. Liu et al. sought to reduce the total energy use of a cloud by geographical load balancing [53]. In terms of improving the operational flexibilities on inter-datacenter communication for cloud providers, GRIPhoN is proposed to offer cost-effective restoration capabilities from the carrier’s perspective [55].

Besides analytical results, there are also a few measurement studies regarding the traffic characteristics across multiple datacenters. [47] and [8] shed some light on good designs of caching and load balancing strategies through measuring traffic dynamics in the Google cloud. Chen et al. presented the first study of inter-datacenter traffic characteristics via five Yahoo! datacenters [23]. Their measurement results motivated our study of reducing costs on inter-datacenter video traffic.

To our knowledge, there exist two recent papers that considered cloud providers’ operational costs on traffic. Zhang et al. designed a routing algorithm to optimize the costs on datacenter-to-client traffic [82]. However, with no consideration of the time dimension, their problem formulation is substantially simplified. Laoutaris et al. proposed NetStitcher, which takes advantage of the already paid traffic volumes at night to reduce costs on inter-datacenter bulk traffic, such as backups [49]. Our work differs in that, instead of conservatively utilizing
leftover bandwidth only for bulk transfers, we argue that such costs can — and should — be minimized by globally optimizing the routing strategies for all inter-datacenter video traffic. In addition, both Jetway and Postcard in this dissertation, which are proposed to minimize the costs on inter-datacenter traffic based on flow-based and store-and-forward models, respectively, have considered the co-existence of multiple traffic flows in their problem formulations, which is far more realistic and complicated than the transfer of a single file in NetStitcher.
Chapter 3

Streaming Multi-Touch Gestures

3.1 Overview

New mobile devices with multi-touch displays have brought revolutionary changes to ways users interact with mobile devices, with multi-touch gestures used as the primary means of interaction. In particular, interactive multimedia applications on mobile devices have made it possible to use gestures intensively to create and consume artistic or musical content in an interactive and collaborative fashion, since gestures are frequently needed to create and manipulate artistic strokes or musical notes. With such media authoring applications, it is desirable to support collaboration among multiple participating users. As an example, it would certainly be exciting if music composition hobbyists may collaborate in real time to work on a musical piece, no matter where they are.

To support such collaboration among multiple users in real time, we propose that gestures are streamed in a broadcast fashion from one user to all participating users, in a broadcast session. Streaming gestures themselves, rather than application-specific data, has made it possible to optimize the design and implementation of a gesture broadcast protocol that can be reused by any mobile multimedia application that needs to support multi-party collaboration. Clearly, it is a more elegant and reusable solution to serve the needs of an entire category of
gesture-intensive media applications. Once received, gestures can be recognized and rendered in real time by a live instance of the same application on a receiver, such that the application-specific states can be precisely reproduced on the receiver. To take such broadcast of gestures a step further, multiple gesture broadcast sessions need to be supported concurrently, so that any participating user can be the source of a gesture stream in each of the broadcast sessions.

With such gesture streaming broadcast sessions in place, a high-quality user experience within an interactive application hinges upon an important Quality of Experience (QoE) metric: the time it takes for a gesture to be recognized at each of the receivers, starting from the time it is recognized at the source of the session. Referred to as the gesture recognizing delay, such a delay is an application-layer QoE metric that directly affects the user-perceived quality of an application session.

In this chapter, we present GestureFlow, a new QoE-aware gesture streaming protocol specifically designed for multiple concurrent broadcast sessions of gestures, with the objective of minimizing gesture recognizing delays in these broadcast sessions. Since only a subset of raw touch events can be recognized as multi-touch gestures, the source of a broadcast session needs to send raw touch events, which will be recognized at each of the receivers. Each recognized gesture will consist of a number of network-layer packets, all of which need to be received for the gesture to be correctly recognized. Unlike traditional media streams, gesture streams typically incur low yet bursty bit rates, but packet losses are not tolerable since each lost packet will severely affect the accuracy of the gesture recognizer on a receiver.

In order to support multiple broadcast sessions while minimizing gesture recognizing delays and guaranteeing reliable packet delivery, we present a detailed design that uses inter-session network coding, and addresses a number of open challenges in our design that have not been discussed in the literature: what the best size of the coding window should be, how the coding window should be advanced, and how packets from different sessions can be coded together.

To validate our design, we have developed a real-world implementation of GestureFlow.
We will use an interactive music composition application, called *MusicScore*, to evaluate the performance of *GestureFlow*. *MusicScore* takes full advantage of our *GestureFlow* implementation to allow composers to enjoy a live collaborative session. During extensive experiments presented in this chapter, we have discovered new challenges in the use of network coding within the *GestureFlow* implementation. It turns out that coded blocks are linearly dependent with one another with an alarmingly high probability, leading to a much higher overhead than what we originally anticipated. We found that it is due to the fact that the coding window size is typically very small, which is required to satisfy stringent delay requirements. We propose to use systematic Reed-Solomon codes on the source to mitigate the overhead due to such linear dependence, and show that the revised QoE-aware gesture streaming protocol has met our needs with the smallest possible gesture recognizing delay.

### 3.2 Gesture Streaming: Motivation and Quality of Experience

Multi-touch allows users to interact with user interface elements directly with their fingers using *gestures*, and has been proven to be the most intuitive interface for a wide variety of applications. These gestures can be as simple as a one-finger tap, or as complicated as a three-finger swipe. As a running example and experimental testbed, we resort to an iPad application for music composition using multi-touch gestures, called *MusicScore* and shown in Fig. 3.1. *MusicScore* allows a user to create musical notes with double taps, to change the pitch of notes by dragging them vertically, and to select a group of notes by dragging a rectangle around them.
3.2.1 Streaming Multi-Touch Gestures

In MusicScore, there are situations in which a teacher gives her student a tutorial on music composition when she is traveling; or composition hobbyists collaborate to compose a piece of music without being physically together. These examples have shown a clear need to facilitate spontaneous sharing of user experience among multiple users, which substantially improves the utility of multi-touch applications. In a nutshell, rather than streaming application-specific data, we propose that multi-touch gestures are streamed instead, regardless of what the application may be. With the same application running on multi-touch devices belonging to all participating users, the streamed gestures can be precisely rendered on a receiving device, as if they are entered live by the local user.

By streaming multi-touch gestures, we immediately gain a number of important benefits. In contrast to the design of customized state exchange protocols for specific collaborative media applications, such as musical symbols in a score and artistic objects in a canvas, it would
be much more generic to stream multi-touch gestures as representatives of user interactions. By handling the replay of streamed gestures as user input, application states are updated correctly at the receiving device. This implies that gesture streaming can serve as an underlying framework that supports any media application that desires multi-party collaboration.

Furthermore, streaming multi-touch gestures makes it easier for multiple users to interact with the same set of application states at the same time. In *MusicScore*, this implies that multiple users are able to compose the same piece of music together, by composing different voices or musical instruments in the piece.

Since all participating users are able to affect the state of the application, it is a must that everyone can see the exact changes made by other users. While some multi-touch gestures can easily be replayed after being streamed to a different user, such as adding a note in *MusicScore*, other gestures only change the views of the local user, such as zooming, and do not affect the state of the application. If participating users have different views, it will be impossible to show all of them on one display. We solve this problem by adopting a “picture in picture” design: a user’s own gestures interact with the native view using the full-screen display, and the views of participating users are displayed in their respective overlay windows. In the example shown in Fig. 3.2, both Alice and Bob are able to work on their preferred views, and to observe the view of the other party at the same time.

![Diagram of multi-touch streams between Alice and Bob](image)

**Figure 3.2:** A “picture in picture” design as multi-touch gestures are streamed between two users, who are collaborating to compose the same piece of music.
3.2.2 Quality of Experience

Our ultimate design objective is to design a reusable framework, called GestureFlow, from the ground up to stream multi-touch gestures to multiple participating users, with the best possible Quality of Experience achieved. The framework uses a shared set of well-designed presentation and transport mechanisms to support a variety of interactive multimedia applications, including MusicScore. There are a number of challenges when designing the GestureFlow framework.

First, multi-touch gesture streams have a very low, yet bursty, bit rate. In multi-touch applications, it is usually the case that users interact with their devices frequently for a while and then stay idle most of the time. For example, a music composer touches the display in MusicScore to add or remove notes only when she is inspired. Fig. 3.3 shows the bit rates of a typical gesture broadcast session in MusicScore over time. We can observe that the peak bit rate reaches 11 kbps, while the average bit rate is no more than 3 kbps.

![Figure 3.3: Bit rates of a typical gesture broadcast session in MusicScore over time.](image)

Second, multi-touch gestures need to be streamed in an in-order, lossless and error-free fashion, as any lost or erroneously transmitted gesture to any of the participating users makes it difficult to precisely render and reconstruct application states at the receiver. This is different from typical media streams, where a loss or an error is considered an inconvenience that degrades playback quality, but not a catastrophic event.
Chapter 3. Streaming Multi-Touch Gestures

Third, gesture streaming has a stringent delay requirement. Media applications that need the support from a gesture streaming framework are interactive in nature, and demand the smallest possible gesture recognizing delay, from when gestures are recognized by the source, to when they are eventually received and recognized by each of the receivers.

Finally, once the replay of streamed gestures has started at a receiver, the interval between the replay of two consecutive gestures has to be kept identical to the difference between their original timestamps when they are generated at the sender. Otherwise, rendered states of an application may be different from the original. This implies that each gesture has to be recognized at the receiver on time, i.e., before its scheduled replaying time, despite the fact that each gesture may be received with different end-to-end delays over the Internet, and as a result experience different gesture recognizing delays. Similar to live media streaming, an initial startup delay — with a corresponding application buffer at the receiver — can be used to mask varying gesture recognizing delays.

Using our “Bob and Alice” example, we illustrate delays in the session from Alice to Bob in Fig. 3.4. Alice’s gestures are received by Bob with end-to-end delays $\tau_1, \tau_2, \tau_3, \ldots$, and recognized by Bob’s application with gesture recognizing delays $v_1, v_2, v_3, \ldots$. Though a gesture may be recognized by Bob’s application, its replay may be delayed by an initial startup delay $\delta$, such that the intervals between gestures, $\Delta t_1, \Delta t_2, \ldots$, are kept precisely the same during replay. In order to make sure all gestures are recognized on time for replaying, the initial startup delay $\delta$ has to be no shorter than the longest gesture recognizing delay, $v_{\text{max}}$. Therefore, to achieve the best possible Quality of Experience with a short $\delta$, gesture recognizing delays need to be minimized.

With these unique characteristics, the design of the GestureFlow framework is more challenging than conventional media streaming systems. It needs to be designed so that a bursty and low bit-rate stream from each user can be transmitted to all participating users in a reliable and timely fashion, in a set of broadcast communication sessions.
### Table 3.1: Examples of multi-touch gestures in *GestureFlow*.

<table>
<thead>
<tr>
<th>Gesture type</th>
<th>Description</th>
<th>Example in MusicScore</th>
<th>Information needed to replay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tap</td>
<td>Tap a view with one or more fingers, possibly multiple times</td>
<td>Select a note or chord</td>
<td>The number of fingers used, the number of taps, and the location in a given view</td>
</tr>
<tr>
<td>Pinch</td>
<td>Move two fingers away or closer to each other</td>
<td>Zoom into or out of the current view</td>
<td>The locations of two fingers in a given view, the scale factor of the pinch, and the velocity of the pinch in scale factor per second</td>
</tr>
<tr>
<td>Swipe</td>
<td>One or more fingers moving towards a direction for a distance</td>
<td>Scroll up or down</td>
<td>The location of the first touch, the direction of the swipe, its velocity and the distance</td>
</tr>
<tr>
<td>Rotate</td>
<td>Move two fingers in a circular motion</td>
<td></td>
<td>To replay this gesture, receivers need to know the location of each touch in the local coordinate system of a given view, the rotation of gesture in radians since its last change, and the velocity of the rotation gesture in radians per second.</td>
</tr>
<tr>
<td>Touch and hold</td>
<td>Touch with one or more fingers and hold for a short period of time</td>
<td>Trigger a pop-up menu to change the note duration or to add accidentals</td>
<td>The location in a given view</td>
</tr>
<tr>
<td>Lasso</td>
<td>Circle an area with one finger</td>
<td>Select a group of notes or measures</td>
<td>The circled area: the locations of the highest, lowest, leftmost and rightmost positions</td>
</tr>
</tbody>
</table>
Figure 3.4: The replay of gestures from Alice to Bob, using an initial startup delay $\delta$ to mask varying end-to-end delays $\tau_i$ over the Internet, as well as gesture recognizing delays $v_i$ in the application.

3.2.3 Presenting Multi-Touch Gestures

We are now ready to present a detailed design of our GestureFlow framework. The first natural question is how gestures should be presented and packetized, in preparation for streaming to multiple participating users.

In multi-touch applications, gesture recognizers are instances that analyze a raw stream of touch objects in a sequence, and determine the intention of users based on properties of each gesture. It analyzes the number of touches and the number of taps from the raw stream, and compares them with the required ones stored in the recognizer to make a decision. Table 3.1 shows descriptions and examples in MusicScore for a collection of useful gestures.

What information should be streamed for a precise playback at a receiver? There are two alternatives. The first is to use a raw stream, represented by a successive sequence of touch events (e.g., finger-down, finger-up, or location update of touch), and the second is to stream a sequence of gestures recognized by gesture recognizers. Intuitively, one may think streaming recognized gestures would be sufficient for mobile applications. Unfortunately, GestureFlow needs to use the first alternative, and to stream raw touch events rather than recognized gestures.
This is because interactive media applications typically include a mixture of raw touch events and recognized gestures. Take a drawing application as an example, while scaling or moving an object in the canvas can be done by gestures, artistic drawing requires a raw stream of touch events to track every movement of fingers. Table 3.2 summarizes examples of multi-touch operations in typical interactive applications, which require both raw touch events and recognized gestures.

Table 3.2: Examples of multi-touch operations in typical interactive applications.

<table>
<thead>
<tr>
<th>Interactive Application</th>
<th>Brushes (Artistic drawing)</th>
<th>MusicScore (Music composition)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Events</td>
<td>Draw a curve with varying speed</td>
<td>Free play on the virtual piano keyboard</td>
</tr>
<tr>
<td>Recognized Gestures</td>
<td>Move a layer in the canvas using panning</td>
<td>Create a note using double-tapping</td>
</tr>
</tbody>
</table>

As soon as the raw stream is received, gestures can be recognized by the same application on the receiver, as illustrated in Fig. 3.5. As a gesture is essentially a sequence of raw touch events, before the last touch event arrives at the receiver and leads to a successfully recognized gesture, all preceding touch events have already been streamed to the receiver. In other words, the recognizer in the receiving application has been progressively receiving “partial” information about this gesture; once the last raw touch event is received, the gesture is immediately recognized. In contrast, if a gesture is streamed when it is fully recognized by the sender, the receiver has to wait for the transmission of an entire gesture, which is typically larger than a touch event, i.e., the delay of recognizing a gesture in the receiving application will be higher.

To packetize and transmit a raw stream of touch objects (Fig. 3.5), we propose to use a simple binary format, due to the fact that there are only four types of events involved (touch-begin, touch-move, touch-end and touch-cancel). With a raw stream, touch objects are continuously generated as a user interacts with her device, and transmitted with a compact form of presen-
User's touches

Touch Objects

Raw Stream

Recognizers

Mobile Application

iOS

GestureFlow

Recognizers

Picture in Picture

Application

raw touches

recognized gestures

Figure 3.5: Streaming a raw stream of touch objects.

Each of these touch events should be accompanied by its sequence number and timestamp. At a receiver, sequence numbers are used to detect out-of-order delivery and losses of the event stream, and timestamps help to replay them with precise time intervals as they are originally generated: any time interval \( \Delta t_i \) can be computed as the difference between the timestamp of the \( i^{th} \) event and that of the \( (i - 1)^{th} \).

3.3 Transporting Gesture Streams

When transporting gestures, we wish to achieve the best possible Quality of Experience, in that gesture recognizing delays are to be as short as possible. It is intuitive to conceive a design where a TCP connection is established between each pair of users, forming a complete graph of overlay. Although TCP guarantees the reliable and in-order delivery of a stream of bytes, the realistic nature of traffic on the Internet dictates that overlay links based on TCP connections exhibit a wide range of delays, and vary significantly over time as well. Further, since TCP uses retransmissions to guarantee reliable delivery, delays may escalate with a slightly more congested link, leading to high delay jitters.
To minimize end-to-end delays of delivering gestures to receivers with guaranteed reliability, we present two approaches by taking advantage of the “all-to-all” broadcast nature of GestureFlow, where every participating node is the source node of a broadcast session to all others, and multiple broadcast sessions exist concurrently in the complete overlay graph connecting all users. First, we propose to use random network coding, which streams coded blocks using UDP flows rather than TCP, and allows possible relaying nodes to relay blocks that they receive after recoding. Second, instead of direct connections in the case of using TCP, multiple paths between the source and each receiver are used to minimize the end-to-end delays of delivering gestures. Since all relaying nodes are receivers themselves, no additional bandwidth is consumed to take advantage of relay paths. The essence of our transport solution is to use network coding as a rateless erasure code for all broadcast sessions to guarantee reliable delivery, tightly coupled with the use of multiple paths to minimize end-to-end delays.

3.3.1 Design Overview

In GestureFlow, we have designed a custom-tailored protocol to utilize random network coding in all of the concurrent broadcast sessions, each streaming gesture events from one of the participating nodes. The reliable delivery of original data blocks is guaranteed with the erasure correction nature of random network coding. Should a particular coded block be lost, subsequent coded blocks received are equally innovative and useful.

Data blocks flow conceptually from each source in a coded form that is mixed with other blocks, via multiple single-hop or two-hop paths to each destination, with each two-hop path using one participating node as a relay. With multiple paths, original data blocks will arrive at a receiver via the path with the lowest delay, yet in a coded form. Fig. 3.6 illustrates an example to show how blocks from Node 1’s session are transmitted in coded forms and following different paths.

When participating nodes are used to relay data blocks, they are also producing their own original blocks. What should each node do with these data blocks belonging to multiple broad-
cast sessions? Since a node is capable of recoding all coded blocks it has received before transmitting them to others, shall we allow for recoding across multiple broadcast sessions? If we do, a participating node would then serve the dual role of being a source node and a relaying node. Referred to as inter-session network coding in the theoretical literature, it has not yet been adopted in any practical systems using network coding.

In the GestureFlow framework, we have made the decision that all nodes are to perform network coding across multiple broadcast sessions. If each node is allowed to mix all incoming blocks with original blocks produced by itself, there is no longer a need to allocate outgoing bandwidth to multiple concurrent sessions, or to schedule outgoing blocks belonging to different sessions competing for outgoing bandwidth. With inter-session network coding, every node only needs to transmit as many coded blocks as the outgoing bandwidth allows, without considering the sessions they belong to.

In the four-node example shown in Fig. 3.6, if we consider all 4 broadcast sessions from 4 users concurrently, the inter-session network coding engine in node 2 is shown in Fig. 3.7. As we can see, node 2 produces coded blocks covering all 4 sessions, each of which carries the necessary information other nodes require to decode, such as the sequence numbers of original blocks, all random coefficients, and the coded payload, so that it is self-contained.
3.3.2 GestureFlow: Protocol Design

Now that we have presented an overview of GestureFlow, we are ready to discuss more details in our protocol design.

**Basics of random network coding.** Let us revisit the basic idea of random network coding, as we have briefly introduced in Chapter 2. Random network coding has been well established in recent research literature [24, 43], and has been shown to maximize throughput in multicast sessions. With random network coding, $k$ original data blocks $b = [b_1, b_2, \ldots, b_k]^T$, each with $s$ bytes, are to be transmitted from the source to multiple receivers in a network topology. A source of a broadcast session generates coded blocks $x_j = \sum_{i=1}^{k} c_{ji} \cdot b_i$ as a linear combination of original data blocks in a finite field (typically $GF(2^8)$), where the set of coding coefficients $c_j = [c_{j1}, c_{j2}, \ldots, c_{jk}]$ is randomly chosen. A relay node is able to perform similar random linear combinations on received coded blocks with random coefficients. Coding coefficients related to original blocks $b_i$ are transmitted together with a coded block.

A receiver is able to decode all $k$ data blocks when it has received $k$ linearly independent coded blocks $x = [x_1, x_2, \ldots, x_k]^T$, either from the source or from a relay. It first forms a $k \times k$ coefficient matrix $C$, in which each row corresponds to the coefficients of one coded block. It then decodes the original blocks $b = [b_1, b_2, \ldots, b_k]^T$ as $b = C^{-1}x$. Such a decoding
process can be performed progressively as coded blocks arrive, using Gauss-Jordan elimination to reduce \( C \) to its reduced row-echelon form (RREF). Fig. 3.8 shows an example of reducing the coefficient matrix to its RREF, after receiving a new coded block and its corresponding coefficients \( (c_3) \).

\[
\begin{align*}
C_{2 \times 5} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 1 \end{bmatrix} \\
c_3 &= \begin{bmatrix} 7 & 2 & 3 & 6 & 3 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\overset{\text{Decoded}}{b_1 | b_2 | b_3 | b_4 | b_5} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix} \\
C_{3 \times 5} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 \end{bmatrix}
\end{align*}
\]

Figure 3.8: Block 2 \((b_2)\) is decoded after receiving the third coded block, using Gauss-Jordan elimination.

Although inter-session network coding is conceptually simple to perform, its real-world design and implementation have brought a number of challenges to the spotlight. In what follows, we illustrate our design choices as we address these challenges.

**The size of an original data block.** When determining what the value of \( s \) — the size of an original data block — should be, we have discovered in our experiments that the size of one gesture event in the stream is very small: less than 512 bytes. In GestureFlow, each original data block contains one touch event if any touch event is produced. When original blocks of different sizes are coded, they are padded with zeros to the size of 512 bytes. Since gesture events do not vary substantially in size and the streaming bit rate is very low, the overhead introduced by such padding is not a concern.

**Cumulative acknowledgments.** How should receivers acknowledge the source node of a broadcast session in which an original data block has been correctly decoded? The first intuitive idea is to selectively acknowledge each of the data blocks as soon as they become decoded, even if they are not consecutive to one another. While it is certainly possible for the source
node to remove any of the original blocks from the coding window when it is acknowledged by all the receivers, it requires a coded block to carry the sequence numbers of all original blocks that are coded. Since sequence numbers of original blocks are not consecutive, it is no longer feasible to carry only the starting sequence number of the earliest original block. Since the additional overhead and complexity may not be justified, we propose to use cumulative acknowledgments, which are much simpler.

With cumulative acknowledgments of decoded data blocks, a receiver uses Gauss-Jordan elimination to reduce the coefficient matrix of all coded blocks it has received so far to its RREF, and finds out which block has just been completely decoded. Instead of acknowledging a newly decoded data block immediately, the receiver sends an acknowledgment for a decoded block only if all earlier blocks with smaller sequence numbers have been decoded. As an example shown in Fig. 3.9, the receiver does not acknowledge $b_3$ even though it has been decoded after receiving two coded blocks $x_1$ and $x_3$. It waits until receiving another coded block $x_4$, which renders all three data blocks, $b_1$, $b_2$, and $b_3$, decoded.

Figure 3.9: A receiver sends a cumulative acknowledgment only when all earlier blocks have been decoded.

**Basics of the coding window at a source node with no relaying.** In the theory of random network coding, it is assumed that $k$ data blocks are to be coded, and if more data blocks are
being transmitted, they are divided into groups of $k$ blocks, and coding is to be performed within each group. If $k$ is fixed, it corresponds to a fixed number of blocks to be coded. However, a fixed group size $k$ may negatively affect the QoE metric in gesture streaming: due to inherently bursty traffic when gestures are streamed, a fixed group size may increase the gesture recognizing delays. As an example, consider the case where 4 blocks are to be coded at a source node, yet only 3 are received or produced, followed by a long idle period. With a fixed group size, the source node would have to wait for all 4 blocks to become available.

To address this challenge, blocks are to be coded within a sliding window in GestureFlow, referred to as the coding window. To explain the basic idea of a coding window, let us first consider the simplified scenario where a node does not relay coded blocks from other broadcast sessions, i.e., only original blocks are coded by the source node of a broadcast session. In this case, as a new original block containing a new gesture event is produced, it is added to the coding window at the source node. A maximum size of the coding window, $W$, is imposed to guarantee successful decoding at receivers, and it corresponds to the maximum number of original blocks that can be coded to produce an outgoing coded block. The source node performs random network coding on original blocks within the coding window, and sends coded blocks to all the receivers as newer blocks are being added to the coding window. The coding window advances itself by removing the earliest data block from the window when the source node has received acknowledgments from all the receivers in the broadcast session.

Fig. 3.10 shows the basic idea of the coding window at a source node. At time $t_1$, the coding window grows to 3 as block 5 enters, and then reaches the maximum coding window size $W$ (4 blocks in this example) at time $t_2$. Note that even though blocks 7 and 8 have already been produced containing new multi-touch events (and buffered), they are not added to the coding window since it has already reached its maximum size. After a few coded blocks are received, the receiver acknowledges that blocks 3 – 5 have been successfully decoded. At time $t_3$, the coding window at the source node advances itself by removing acknowledged blocks, and then blocks 7 and 8 enter the coding window. By adopting the sliding window mechanism, during
bursty periods when touch events are produced back-to-back, the coding window expands to cover new events, so that they can be received and decoded by receivers in time. During idle times when touch events are scarce, the size of the coding window is naturally reduced as acknowledgments are being received.

*Shrinking the maximum size of the coding window on demand.* Since raw touch events are streamed directly and a gesture consists of multiple raw touch events, with a fixed maximum size for the coding window, blocks containing information to recover one gesture may be split into separate coding windows, incurring longer gesture recognizing delays at the receivers. Longer gesture recognizing delays can be reduced if we can send out a coded block *immediately* when the recognizer on the source node recognizes a gesture.

In our design, the source node will always “shrink” the maximum size of the coding window to the current block whenever a gesture is recognized, so that no other blocks can enter the current coding window. By doing so, the source node will not need to hold itself back and wait for new original blocks after a gesture is already recognized; it is also easier for receivers to
receive enough coded blocks to recover the gesture since fewer blocks are contained in a coded block. Fig. 3.11 shows an illustration of our maximum coding window size adjustment mechanism with the previous example in Fig. 3.10. When a gesture is recognized by the recognizer at time $T_4$, the source node will adjust its maximum size of the coding window to 2 blocks, and send out these coded blocks immediately. When the source node receives acknowledgments confirming that blocks belonging to this gesture are successfully decoded by all receivers, the source node advances its sliding window, and the maximum window size is then restored to its original value (4 blocks).

Figure 3.11: The source node adjusts its maximum coding window size $W$ when a gesture is recognized by its gesture recognizer. The number in the circle indicates the size of the coding window; and the dark circle indicates that the maximum coding window size has been reached.

**The coding window with inter-session network coding.** Unfortunately, the design of the coding window in GestureFlow becomes more complicated with inter-session network coding, where a source node of a broadcast session also serves as a relaying node for other sessions. The initial complexity comes from the computation of the coding window size. Even though a source node also serves as a relay and mixes incoming coded blocks from other sessions, these incoming coded blocks should not affect the computation of the coding window size. In other words, the coding window size should still be the number of original blocks that the source node itself has produced, and the source node will not include new original blocks in its coding
window if it has reached its maximum size.

As we mix blocks from multiple broadcast sessions, what is the set of blocks that is to be coded on a node for an outgoing block to be produced? Of course, if an original data block is already decoded at a downstream receiver, it should not be included in the recoding process. As a result, with inter-session network coding, the coding window at each node should selectively remove original data blocks within other broadcast sessions, if they are no longer useful to all the receivers. But how does a node in its role as a relay know which original block is already decoded at receivers?

The short answer to this question is: the relaying node does not know directly, but the source node knows, since receivers send cumulative acknowledgments to the source node of a session, acknowledging the latest original block that has been decoded. Of course, these acknowledgments are sent directly from a receiver to the source node of a session, and are not sent to any of the relaying nodes. Nevertheless, after the source of a session advances its coding window by removing its earliest original block, all relaying nodes will easily detect such an advance, as the sequence number of the earliest original block is embedded within a coded block.

When does a node remove a block from its coding window? As a coded block arrives or as an original block is produced, it adds its coefficient row to the existing coefficient matrix, and reduces the new matrix to its RREF. After eliminating original data blocks from its own broadcast session that are acknowledged by all receivers, it simply recodes all rows in the existing matrix, even if an original data block from another broadcast session is completely decoded when reducing the coefficient matrix to its RREF. The node does not remove the block from its coding window immediately, since doing so introduces the risk that subsequent original blocks may not be decoded. Instead, a node, in its role as a relay, waits until the source node of a broadcast session advances its own coding window. The relaying node removes an original block from its coding window only when its sequence number is smaller than the starting sequence number in a newly received coded block from the source node of a session.
Since the source node only advances its coding window when all receivers have decoded an original block, recoding such a block will no longer benefit any of the receivers.

Figure 3.12: Removing data blocks from the coding window of a node in its role as a relay (node 4).

To illustrate the design of the coding window with inter-session network coding, in the context of our four-node example given in Fig. 3.7, Fig. 3.12 shows the coefficient matrix at node 4. $b_i^{(j)}$ represents an original data block $i$ in the $j^{th}$ broadcast session, $S_j$. In the left-side matrix in RREF, we observe that node 4 has decoded $b_1^{(1)}$ and $b_1^{(2)}$, from $S_1$ and $S_2$ respectively, by receiving the first three coded blocks, and the fourth coefficient row corresponds to $b_1^{(4)}$ from node 4 itself for inter-session network coding. In a newly received coded block (the coefficients of this block are in the last row), any information related to $b_1^{(1)}$ and $b_1^{(2)}$ is no longer included, which indicates that the coding windows at node 1 and node 2 have advanced beyond these original data blocks. In its role as a relay, node 4 now removes coefficient rows that correspond to these two blocks (circled by dashed rectangles) from its coding window, within which it produces outgoing coded blocks in the future.

In summary, Algorithm 1 describes the GestureFlow protocol using inter-session network coding.
3.4 Experiences with GestureFlow

We dedicate this section to investigations of how GestureFlow performs in real-world systems using MusicScore, a collaborative music composition application. Users interact with MusicScore to compose music using only multi-touch gestures. MusicScore takes full advantage of the GestureFlow framework to stream gesture events among multiple participating users, such that composers can enjoy a live collaborative experience. Our GestureFlow framework has been implemented in Objective-C in the Xcode programming environment. Fig. 3.13 shows a scenario of a live MusicScore composition session on the iPad, in which collaboration is achieved using GestureFlow.

Figure 3.13: MusicScore in action: two users are collaboratively composing a musical piece with support from the GestureFlow framework.

To minimize the computational load on the iPad, we have included an optimized implementation of random network coding in the GestureFlow framework. Our implementation of network coding is able to progressively decode incoming coded blocks using Gauss-Jordan elimination, while taking full advantage of SIMD instructions available in the ARM v7 archi-
Chapter 3. Streaming Multi-Touch Gestures

Architecture, used by CPUs powering the iPad (all generations) and the iPhone (including 3GS, 4 and 4S). The GestureFlow implementation itself contains about 8000 lines of code.

3.4.1 Performance Evaluation

As our primary QoE metric, we first present measurement results with respect to the gesture recognizing delays. In each run of our experiments, we measure the gesture recognizing delay in a collaborative music composition session between a pair of iPads running MusicScore, and the corresponding CDF curve is derived from multiple runs of our experiments. Our experiments are performed in both Wi-Fi networks and 3G cellular networks to better capture the performance of GestureFlow. Given a type of Internet connectivity (Wi-Fi or 3G), two iPads are connected to the Internet via two different ISPs, reflecting a more dynamic network condition. We have also implemented a traditional TCP-based streaming protocol, named TCP Relay, as a baseline for our comparisons. For fairness, TCP Relay also transmits data blocks through both direct TCP link and two-hop relay paths to minimize both end-to-end delays and delay jitters.

As shown in Fig. 3.14 and Fig. 3.15, when GestureFlow is used, averages of gesture recognizing delays are 102.6 msec and 253.3 msec for Wi-Fi and 3G users, respectively. In contrast, TCP Relay suffers from much longer gesture recognizing delays: 183.6 msec and 485.1 msec on average, for Wi-Fi and 3G users, respectively. The shorter delays achieved by GestureFlow can be attributed to both the adoption of inter-session network coding and a sliding coding window, specifically designed for gesture streaming. Besides, since devices in cellular networks (3G or EDGE) cannot directly connect with each other via TCP, due to NAT restrictions, they have to connect to the same set of relays with publicly accessible IP/port (e.g., dedicated relay servers for users in cellular networks) for data exchange. This results in longer end-to-end delays between 3G users, as shown in Fig. 3.15. In contrast, UDP-based GestureFlow can easily achieve NAT traversal in cellular networks, and as a result achieve shorter end-to-end delays.

Furthermore, we would like to evaluate the performance of GestureFlow in a real-world
scenario with four users in “all-to-all” broadcast sessions, with one iPad user connecting to the Internet through campus Wi-Fi, one iPad user using the household Wi-Fi access point, and two iPhone 4S users connecting to the Internet through 3G and EDGE, respectively. Similar to the two-user experiments, these four devices are connected to different ISPs and located in different locations in the same city. Table 3.3 summarizes the average gesture recognizing delays in both GestureFlow and TCP Relay between each pair of devices, over 20 runs of experiments. It is clear that GestureFlow achieves better performance: gesture recognizing delays are 23 – 52% shorter compared to those in TCP Relay.

Next, we evaluate an important design choice adopted in GestureFlow: the use of multiple paths between the source and each receiver to minimize end-to-end delays. Fig. 3.16 shows the average percentage of blocks a node receives from relaying nodes in both GestureFlow and TCP Relay in the four-user “all-to-all” streaming scenario, along with the 95% confidence interval. We observe that in GestureFlow, Wi-Fi and 3G users have more than 10% of the received blocks from relaying nodes, and up to 30% of received blocks are from relaying nodes for the EDGE user, due to longer network delays on direct EDGE links. As a result, EDGE
Table 3.3: Comparisons of Gesture Recognizing Delays (msec) using GestureFlow and TCP Relay.

<table>
<thead>
<tr>
<th></th>
<th>GestureFlow/TCP Relay</th>
<th>Wi-Fi 1</th>
<th>Wi-Fi 2</th>
<th>3G</th>
<th>EDGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi 1</td>
<td>−</td>
<td>103/178</td>
<td>192/376</td>
<td>309/517</td>
<td></td>
</tr>
<tr>
<td>Wi-Fi 2</td>
<td>89/184</td>
<td>−</td>
<td>167/257</td>
<td>274/415</td>
<td></td>
</tr>
<tr>
<td>3G</td>
<td>224/391</td>
<td>188/294</td>
<td>−</td>
<td>364/493*</td>
<td></td>
</tr>
<tr>
<td>EDGE</td>
<td>347/487</td>
<td>301/428</td>
<td>398/519*</td>
<td>−</td>
<td></td>
</tr>
</tbody>
</table>

Note: there is no direct TCP connection between cellular devices due to NAT restrictions. Relay paths have been used as a result.

users rely more on relay paths that have shorter delays than those direct ones. However, only a small percentage of blocks are observed from relaying nodes in TCP Relay, especially for the EDGE user, which indicates that it fails to take full advantage of multiple paths as GestureFlow does.

Figure 3.16: The percentage of blocks from relaying nodes over all received blocks in GestureFlow and TCP Relay.

Figure 3.17: The average of gesture recognizing delays with different network sizes using GestureFlow and TCP Relay.

To investigate the scalability of GestureFlow, we further study the correlation between
the gesture recognizing delay and the number of participating users. Note that all participating nodes are Wi-Fi users in this experiment. Shown in Fig. 3.17, as the number of nodes increases, the average of gesture recognizing delays in *GestureFlow* varies mildly around 100 ms. We can even observe a slight decrease in the gesture recognizing delay when the number of nodes is large in *GestureFlow*, e.g., 14 nodes, which is due to an increased number of relay paths that may provide shorter end-to-end delays. In contrast, the average of gesture recognizing delays in *TCP Relay* increases significantly when the system scales up, which is mainly due to congested TCP connections that are overwhelmed by relayed blocks. Such an observation implies that a set of complex relay selection and rate control algorithms are required in TCP-based gesture streaming, as opposed to the simpler design of inter-session network coding in *GestureFlow*.

We also investigate the bandwidth overhead in *GestureFlow* by evaluating the difference between the gesture streaming bit rate, which is computed as the average of four broadcast sessions, and the upload bit rate per user, which is defined as the average upload bit rate each user devotes to every broadcast session. As shown in Fig. 3.18, the gap between these two curves becomes wider as the streaming bit rate becomes higher. The reason is that during bursty periods, every user has to contribute more bandwidth to upload coded blocks containing blocks from other sessions, which introduces more bandwidth overhead. Yet, the bandwidth overhead for each user is less than 5 kbps in general, which is reasonable. It is critical to point out that even with overhead considered, the upload bit rate per user is only about 8 kbps on average, which is fairly low in streaming systems. This verifies our design philosophy that bandwidth is not a major concern in *GestureFlow*.

### 3.4.2 A Subjective Evaluation of the User Experience

Although our experimental results have so far shown that *GestureFlow* is able to provide a better Quality of Experience for gesture streaming in interactive media applications by providing shorter gesture recognizing delays, it may not yet be fully convincing. We are more interested in the actual feedback from real-world users when *GestureFlow* is in use, since it reflects
the Quality of Experience directly. As a result, we have conducted a series of experiments to capture user feedback. We invite users to use MusicScore in a two-user interactive music composition session over Wi-Fi hotspots. They are asked to rate gesture recognizing delays they have experienced in a 5-min interactive collaboration session, selecting from 4 categories: 1) delay is too long, i.e., not usable from a user’s perspective; 2) delay is long, but still tolerable; 3) satisfying, but with a noticeable delay; 4) excellent.

Figure 3.18: Bandwidth overhead per user.

Figure 3.19: User experience ratings of different gesture recognizing delays in MusicScore using GestureFlow.

Figure 3.20: User experience ratings of different gesture recognizing delays in MusicScore using TCP Relay.
In Fig. 3.19 and Fig. 3.20, we show gesture recognizing delays of collaborative sessions and their corresponding user experience ratings, when *GestureFlow* and *TCP Relay* are in use through Wi-Fi connections, respectively. Clearly, *GestureFlow* is able to garner higher user experience ratings with shorter gesture recognizing delays compared to *TCP Relay*. Similar trends are also observed with other connection types. While most Wi-Fi users reported better collaboration experiences when gesture recognizing delays are shorter than 150 msec, users are observed to be more tolerable to longer delays in 3G networks. Our evaluation results show that users tend to give a rating of 4 even though their experienced delays are around 300 msec in 3G networks. This can be explained as users usually expect 3G networks to be slower than Wi-Fi. Validated by both shorter gesture recognizing delays and higher user experience ratings, we believe that *GestureFlow* is able to achieve a satisfactory Quality of Experience.

### 3.4.3 Performance of Network Coding

![Figure 3.21: The average delays along with 95% confidence intervals in different experiment settings.](image)

Since we apply network coding in *GestureFlow*, it is important to justify this design choice. Fig. 3.21 shows the relationship between the maximum coding window size $W$ and the gesture recognizing delay. We observe that the gesture recognizing delay increases when $W$ is getting
either smaller or larger, and reaches its minimum when $W$ equals to 8. The underlying reason is that when $W$ is set to be too small, the source needs acknowledgments for almost every block to advance the coding window. Subsequent blocks have to wait a longer time before they can be coded and transmitted, which increases the delay, especially in bursty periods. On the other hand, if $W$ is too large, the received coded blocks always contain coding coefficients for newly coded blocks, which increases the delay in the decoding process.

![Figure 3.22: A comparison of gesture recognizing delays between GestureFlow with and without the adaptive coding window size adjustment mechanism.](image1)

![Figure 3.23: CDF of the number of original blocks to be coded at a node with different choices of the maximum coding window size ($W$).](image2)

As an adaptive maximum coding window size adjustment mechanism is specifically designed for QoE-aware gesture streaming, we would also like to verify its effectiveness through performance evaluation. In Fig. 3.22, we compare the gesture recognizing delays when the adaptive maximum coding window size adjustment mechanism is on and off among Wi-Fi users. Clearly, benefited from the shrunk maximum coding window when gestures are recognized, the average gesture recognizing delay is reduced from 121.5 msec to 102.6 msec. Similar results are also observed in measurements among users with other Internet connection types.
Having evaluated the coding window size and its adaptive adjustment mechanism, we now proceed to observe the actual number of blocks to be coded at each node. We plot the CDF for the number of original blocks and the number of relayed blocks at each node, which are shown in Fig. 3.23 and Fig. 3.24, respectively. From Fig. 3.23, we can see that 90% of time there is no more than 5 original blocks to be coded at a node. This indicates that most of the time there is very little delay added in both the encoding and decoding processes, as blocks do not have to wait too long to be transmitted or relayed. The underlying reason is that GestureFlow has very bursty traffic. Since users remain idle most of the time, the actual coding window size is naturally reduced. Similarly, Fig. 3.24 shows that 90% of the coding windows have a size of no more than 11 blocks, with an average of around 4 blocks. This indicates that, in general, there is only one block or two from each broadcast session required to be recoded at the relaying node, which justifies the use of inter-session network coding. By mixing a limited number of coded blocks from multiple sessions together, recoded blocks generated by relaying nodes are useful to downstream receivers with high probability.

Another concern when applying network coding is its CPU load and memory usage, which
are mainly introduced by Gauss-Jordan elimination in the decoding process. We have measured the CPU load and memory usage over time at an iPhone 3GS node, with results shown in Fig. 3.26. As we can see, the average CPU usage is 8.4%, with peaks corresponding to bursty bit rates in Fig. 3.18. The dashed line shows the memory usage over time, which is 2.4% on average. iPad nodes have even lower CPU loads as they enjoy a higher CPU frequency in their Cortex A8 architecture, and the same memory usage as the iPhone 3GS (both have 256 MB of main memory). As such, the CPU load and memory usage of network coding in GestureFlow are acceptable.

It is critical to point out that coded blocks in network coding, either from source nodes or relaying nodes, are considered useful only when they are *linearly independent* with one another, or else they are regarded as redundant blocks. The ratio of linear dependence among coded blocks with different coding window size $W$ is also investigated, when evaluating the performance of network coding. For a specific data block, its linear dependence is computed as the percentage of linearly dependent blocks over the total number of coded blocks involving the data block. By plotting the CDF of linear dependence of all coded blocks in our experiment in Fig. 3.25, we find out that in 90% of them, around 15% of blocks are linearly dependent, which is an alarmingly high percentage. The percentage of linear dependence is even higher when the coding window size is becoming smaller, *e.g.*, it becomes 18% when $W = 8$.  

![Figure 3.26: The CPU load and memory usage of network coding in an iPhone 3GS device.](image)
A high percentage of linear dependence among coded blocks implies a large portion of redundant blocks, which unnecessarily consumes bandwidth. Though we emphasize that gesture streams typically incur very low bit rates, they are highly bursty as well. Shown in Fig. 3.3, the bursty bit rate reaches 10 kbps in a session. The bandwidth waste due to linearly dependent blocks may escalate with concurrent broadcast sessions. More importantly, a high percentage of linear dependence may result in longer gesture recognizing delays as nodes have to wait for more useful blocks to decode a gesture. With QoE awareness, we need to carefully analyze and address the challenge of linear dependence.

### 3.5 The Problem of Linear Dependence

#### 3.5.1 Analyzing the Effects of Linear Dependence on QoE

In this section, we show theoretical insights on how linear dependence among coded blocks negatively affects the Quality of Experience of users by increasing their gesture recognizing delays.

We first describe our system model formally. Assume that \( N \) users participate in an gesture broadcast session. Each of them is not only a source that generates gestures, but also a receiver and a relay of blocks from other sessions. The network is modelled as a directed acyclic graph \( G = (\mathcal{V}, \mathcal{E}) \). \( \mathcal{V} \) is the set of network nodes that represent participating users, \( i.e., |\mathcal{V}| = N \). \( \mathcal{E} \) is the set of network links. The links are characterized by the total link capacity \( e_{ij} \) expressed in blocks per second, and the average packet loss rate \( \pi_{ij} \), where \( (i, j) \in \mathcal{E} \) denotes the link between the nodes \( i \) and \( j \) in \( \mathcal{V} \). We denote the average percentage of linear dependence among coded blocks by \( ld \).

It is obvious that the gesture recognizing delay depends on the average packet loss rate and the percentage of linear dependence among coded blocks. To be exact, the expected delay observed at each node can be computed by estimating the average number of blocks that it receives before it can decode those gestures. Let \( D_i \) be the average delay observed at node \( i \)
for receiving a sufficient number of blocks such that it can decode gestures in coding windows of all other users. $D_i$ has the form of

$$D_i = d_i \sum_{k=(N-1)W}^{\infty} kP_i(k).$$

In this equation, $k$ is the number of blocks that node $i$ receives before it can decode the gestures; $P_i(k)$ denotes the probability of decoding these gestures after receiving exactly $k$ blocks; the constant $d_i$ denotes the average delay for receiving one block and can be approximated as $d_i = \frac{1}{\sum_{j \in V_{-i}} e_{ji}}$, where $V_{-i}$ is the set of nodes in $V$ without node $i$, i.e., $V_{-i} = V \setminus \{i\}$. Note that $k$ includes all coded blocks that can be either linearly dependent or independent from other blocks.

Since at most $W$ original blocks, including all received coded blocks from other $N-1$ broadcast sessions, are allowed to be coded together to produce a new coded block at each node, the minimum number of blocks needed for decoding gestures from all other users equals to $(N-1)W$. That is to say, the probability of decoding with fewer blocks than $(N-1)W$ equals 0. Hence, the probability $P_i(k)$ of decoding gestures from all other users with exactly $k$ blocks corresponds to the probability of forming a full rank system upon receiving the $k^{th}$ block but not before that. Analytically,

$$P_i(k) = \frac{k-1}{k-(N-1)W} p_i^{(N-1)W} (1-p_i)^{(k-(N-1)W)},$$

where $p_i$ represents the probability that a useful block arrives at node $i$.

Since a block is considered useful if it is not lost due to packet erasures and it is linearly independent to coded blocks that a node has already received, the probability $p_i$ can be represented by the link capacity, the packet loss rate on each link, as well as the average percentage of linear dependence among coded blocks. It takes the following form:

$$p_i = \frac{(1-ld) \sum_{j \in V_{-i}} e_{ji}(1-\pi_{ji})}{\sum_{j \in V_{-i}} e_{ji}}.$$

More formally, the probability that a useful block arrives at each node is defined as the fraction of total useful blocks arrived over its incoming bandwidth capacity.
From our analysis, we can see that when the percentage of linear dependence among coded blocks $ld$ increases, the probability $p_i$ of a useful block arriving at each node will decrease. This then results in an increase of probability $P_i(k)$, since $\frac{\partial P_i(k)}{\partial p_i} < 0$. As a consequence, the average gesture recognizing delay $D_i$ observed at each node will increase.

### 3.5.2 Mitigating Linear Dependence

To mitigate the high percentage of linearly dependent blocks that incur longer gesture recognizing delays, we are inspired by systematic Reed-Solomon codes, and propose to generate coding coefficients for the original blocks at each node based on the Vandermonde matrix in GestureFlow.

With coding coefficients generated by the Vandermonde matrix, each node codes original blocks first, rather than codes coded blocks belonging to its own session from the onset. These original blocks can be seen as a special case of coded blocks, with coding coefficients as rows in an identity matrix. After coding all original blocks, a node starts to generate and code coded blocks from its own session. In order to code $k$ original blocks using an $(n, k)$ Vandermonde matrix over a Galois field $F_q$, a node is able to generate up to $n-k$ coded blocks, after original blocks are coded. In GestureFlow, a $(n-k) \times k$ Vandermonde matrix $G$ [65] of the following form is used to generate these coded blocks:

$$ G = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & k \\ 1 & 2^2 & 3^2 & \cdots & k^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2^{n-k-1} & 3^{n-k-1} & \cdots & k^{n-k-1} \end{bmatrix}. $$

Since matrix $G$ is a Vandermonde matrix, it is easy to see that any $k \times k$ submatrix of $G$ has a non-zero determinant and is nonsingular, and as a result every subset of $k$ rows of $G$ is guaranteed to be linearly independent. As such, linear independence among all original blocks
coded from the source is guaranteed with the use of the Vandermonde matrix. These blocks from the source are always innovative once received.

Compared with random linear codes, one difference of using the Vandermonde matrix at the source is that it is not rateless. With the \((n - k) \times k\) Vandermonde matrix \(G\), a maximum of \(n - k\) coded blocks can be coded, in addition to the original blocks. In contrast, with a randomized generation of code vectors, random network coding is able to produce a practically infinite number of coded blocks to ensure successful decoding with any erasure channel.

In practice, though, this is not a serious limitation in GestureFlow. We have shown before that the optimal coding window size, \(W\), is 8, which implies that \(k \leq 8\), and the receiver is able to decode successfully as long as \(k\) linearly independent blocks — original or coded — are received. Since \(W\) is set to be so small, even if a standard size of the Galois field \(q = 256\) is used, and Galois field arithmetic is performed on \(\text{GF}(256)\) during coding, \(n\) can still be chosen to be as large as \(q - 1 = 255\), which means that the code used is a \((255, k)\) code where \(k \leq W\). This is indeed a linear code with a very low rate, and implies that decoding will be successful with high probability. In situations where the packet loss rate is so high that fewer than \(k\) linearly independent blocks are received, the session is considered to be terminated.

Other parts of the transport protocol in GestureFlow remains the same, in a sense that the cumulative acknowledgments, progressive decoding, relay paths, and inter-session network coding are still adopted. Note that each node still uses random linear network coding to generate coding coefficients when recoding received blocks, so that there are no restrictions imposed on the actual number of coded blocks in the coding process.

### 3.5.3 Evaluating the Use of the Vandermonde Matrix

The effectiveness of using the Vandermonde matrix at the source in GestureFlow is evaluated with additional experiments with MusicScore. Fig. 3.27 shows the comparison of CDFs of linear dependence between the original GestureFlow design and the use of Vandermonde matrix to mitigate linear dependence. The maximum coding window size \(W\) is set to be 8. It is
clear that the 90\textsuperscript{th} percentile of linear dependence is significantly reduced by using the new design, from 18\% to 6\%. Since blocks coded using the Vandermonde matrix from the source are guaranteed to be linearly independent with each other, the linear dependence is caused by the recoding process in relaying nodes, which is acceptably low. The ratio of linear dependence with different coding window sizes is explored in Fig. 3.28. We can see that the ratio of linear dependence is decreasing as the coding window size increases, and the ratio with the Vandermonde matrix used at the source is much smaller than the original GestureFlow design.

Since the most critical design objective in GestureFlow is to satisfy a stringent delay requirement, we compare gesture recognizing delays and their standard deviations from the Wi-Fi 1 User to other three users by using the new and original GestureFlow designs, respectively, shown in Table 3.4. We can observe that by using Vandermonde matrix as coding coefficients at the source, average gesture recognizing delays through different kinds of connections have all been evidently reduced. Since the redundancy due to linear dependence is mitigated with the Vandermonde matrix at the source, a received block can be used to decode with a higher probability, which reduces the decoding delay.


Our experiments have evaluated our important design decisions made in GestureFlow, with the objective of reducing gesture recognizing delays. Our results have confirmed that gesture recognizing delays are effectively reduced with our proposed protocol in GestureFlow, and that it scales well when the number of participating nodes increases.

### 3.6 Summary

We are firm believers that gestures represent a new paradigm for users to interact with mobile devices, and that social and collaborative aspects of gesture-intensive applications will usher in an era of streaming gesture events live, so that applications do not need to design and implement custom-tailored solutions. We are intrigued by the very low yet bursty bit rates when streaming gesture events over the Internet, as shown in the MusicScore application. Such low streaming bit rates, coupled with the need for guaranteed reliability, low gesture recognizing delays, and multiple concurrent broadcast sessions when multiple users are involved, have brought us brand new but very practical challenges that need to be addressed with a new transport solution.

As the GestureFlow framework is being designed, we have tried a number of alternative designs, governed by the principles of simplicity and practicality. In this chapter, we presented our design of using inter-session network coding with multiple paths, allowing for recoding across multiple concurrent sessions. We have not only presented how well our design in GestureFlow works, but also why we have chosen such a design. The use of network coding has simplified our design and implementation, making them more practical. We are in the hope that GestureFlow only represents the first step towards a mature framework that facilitates the
broadcasting of gesture streams, so that users interact with one another in a simple and transparent fashion to create or consume multimedia content, wherever they may be around the world.
Algorithm 1 GestureFlow running on the source node of session $S_j$.

**Require:** Received a multi-touch event

1: Encapsulate the new multi-touch event into an original block, $b_i^{(j)}$, with proper zero-padding.

2: if the maximum size of the coding window has not been reached then

3: Include $b_i^{(j)}$ into the coding window

4: Increment the size of the coding window

end if

**Require:** Received an ACK from a receiver in the session $S_j$

6: Compute the smallest sequence number $r$ from all ACKs received so far from receivers.

7: Advance the coding window by removing all original blocks before $b_r^{(j)}$ (inclusive).

8: Include more buffered original blocks into the coding window, if any, until the maximum size of the coding window has been reached.

9: Recompute the size of the coding window based on the number of original blocks included.

**Require:** Received a coded block

10: Add the coded block to the coding window.

11: Reduce the coefficient matrix (corresponding to blocks in the coding window) to its RREF using Gauss-Jordan elimination.

12: if $b_p^{(q)} (q \neq j)$ and earlier blocks can be decoded and $b_{p+1}^{(q)}$ cannot be decoded then

13: Decode blocks till $b_p^{(q)}$ (inclusive)

14: Send ACK containing $p$ to the source node of $S_q$

15: end if

16: if $b_i^{(q)} (i \leq p, q \neq j)$ is not included in the received coded block then

17: Removing blocks associated with $b_i^{(q)}$ from the coding window

18: end if

**Require:** The network is ready for a block to be transmitted

19: Produce and transmit a linear combination of all blocks in the coding window with randomly generated coefficients.
Chapter 4

Airlift: Video Conferencing as a Cloud Service

4.1 Overview

The key advantage of cloud computing is to maximize the sharing of resources with statistical multiplexing, while keeping users of the cloud satisfied. To provide cloud services with a higher quality, it is customary for cloud providers to deploy a number of datacenters across different geographical regions, inter-connected with high-capacity links. Enterprises, such as Netflix, are moving their entire platform to the cloud [25] to take advantage of its abundant resources that are available on demand.

In Chapter 3, we have proposed a new broadcast protocol based on inter-session network coding, specifically designed for gesture streams with bursty and low bit rates, with the objective that gesture recognizing delays can be as low as possible. In this chapter, we wish to investigate whether cloud computing platforms can be utilized to provide a cloud service to mobile applications with demanding requirements for both bandwidth and end-to-end latencies.

From the perspective of both bandwidth demand and end-to-end delay constraints, multi-
party video conferencing may be one of the most demanding mobile media applications. Existing conferencing solutions in the literature have traditionally focused on the use of peer-to-peer (P2P) [21, 22] or client-server architectures (e.g., Microsoft Lync). With abundant bandwidth between datacenters, one would naturally wonder if it is feasible to take full advantage of inter-datacenter networks in the cloud to support higher bit rates in video conferences, yet still maintaining acceptable delays.

In this chapter, we promote the use of inter-datacenter networks in cloud computing platforms to support live multi-party video conferencing as a cloud service. Our proposed protocol and real-world implementation, collectively referred to as Airlift, is designed from the ground up to support multiple live conferences with an inter-datacenter network operated by a cloud provider. As its name suggests, the unique advantage of Airlift is to provide low-latency end-to-end paths among participants in multiple conferences, yet without the “hustle and bustle” of the public Internet. With Airlift, packets in conferences can be routed through a high-capacity inter-datacenter network, as if they are traveling around the world in chartered private flights with minimal congestion, rather than cruise ships with long lines waiting for embarkation.

In order to make a strong case for the design of Airlift, we open this chapter with extensive measurement studies in the Amazon EC2 cloud, based on our real-world implementation of a video conferencing system that supports multi-hop and multi-path transmission of packets in live conferences. Our studies have demonstrated clear and convincing evidence that the inter-datacenter network offers significant performance advantages over the public Internet with respect to sustainable video flow rates, yet with acceptable delays.

A highlight of this chapter is our design of a new application-layer protocol for inter-datacenter networks. Its original features are two-fold: First, to be more scalable, it aggregates user-initiated conferences to a smaller number of multicast sessions among datacenters. Second, it is designed to maximize the total throughput across all the sessions, while maintaining basic fairness across different conferences, and making sure that stringent delay constraints are not violated.
Due to the multicast nature of aggregated sessions, traditional wisdom resorts to Steiner tree packing [22] in order to maximize the video flow rate from a single source to the remaining participants in a video conference. Since the problem is NP-Complete, existing works [22, 52] pack only depth-1 and depth-2 trees. With a large number of conferences served concurrently in an inter-datacenter network, packing Steiner trees for each source and in each conference is computationally prohibitive, even with trees of limited depth. To solve this problem, we use intra-session network coding as an integral part in both our protocol design and our real-world implementation. Thinking from the perspective of conceptual flows [51], the upshot of network coding is its power of resolving conflicts competing for bandwidth resources in bottleneck links. With the help of network coding, we are now able to formulate the problem of maximizing the total throughput across all aggregated sessions as a linear program, easily solvable using a standard LP solver. Its optimal solution serves as the foundation of the Airlift protocol.

Finally, using Airlift as a video conferencing cloud service is simple. In our design, the cloud service can be treated as a full-service broker: a participating user with a video source in a conference only needs to transmit its packets to one of the datacenters in the cloud, and to
process acknowledgments from the cloud service. Fig. 4.1 shows an illustrative example of a 5-party video conference, supported by Airlift.

We evaluate the validity and performance of Airlift as a cloud service with our real-world implementation, with 17,000 lines of code in C++. Our implementation has been developed with performance in mind: to maximize packet processing rates, asynchronous networking has been used; to minimize the computational overhead of network coding, our network coding implementation is accelerated with the Intel/AMD SSE2 instruction set.

Our real-world experimental results over PlanetLab and the Amazon EC2 cloud have shown substantially (3 to 24 times) higher throughput as compared to Celerity [22], a state-of-the-art P2P solution, yet without any disadvantage on end-to-end delays that can be perceptible to end users. To the best of our knowledge, we are the first to present the design and implementation of a cloud-based solution for video conferencing.

4.2 Motivation and Design Objectives

4.2.1 Conferencing via the Cloud: Motivation

The Achilles’ heel of peer-to-peer video conferencing solutions, such as Celerity [22], is the challenge of computing the flow rate on each overlay link between users who participate in the same conference. Such a challenge comes from the fact that overlay links may compete for the same physical link in the layer-3 Internet topology; yet due to the lack of complete knowledge about the underlying layer-3 topology, it would be infeasible to determine how overlay links share common physical links.

Such a challenge is present even in a simple “dumbbell” topology, illustrated in Fig. 4.2. Without the knowledge that a physical bottleneck exists between the two user pairs, the number of overlay links competing for the bottleneck may not be optimal, if incorrect trees are formed to route packets. To address such a challenge, Celerity resorts to a complex mix of algorithms, including decentralized optimization to converge to optimal overlay link rates based on loss
rates and queueing delays, as well as spanning tree packing at each source to compute its overlay trees.

In contrast, if we take advantage of the high-capacity inter-datacenter network in the cloud, the pairs of users on both sides of the dumbbell topology can each transmit to their respective datacenters, as Fig. 4.3 shows, with user 1 as an example. Each datacenter is responsible for aggregating incoming video flows from both users, and forwarding them to the other datacenter. With such aggregation, the number of video flows sharing the bottleneck, which resides between the two datacenters in the cloud, is naturally minimized, without the complexity of Celerity. If the inter-datacenter link has a higher capacity (e.g., due to private peering relationships), the gain on the total throughput in the conference is even more substantial.

![Figure 4.2: With P2P conferencing, more than the minimum number of overlay links may compete for the same physical bottleneck.](image)

![Figure 4.3: Conferencing via the cloud: flow rates over the physical bottleneck are aggregated and minimized.](image)

But are end-to-end delays sacrificed by routing packets through the cloud? We answer this question with results from real-world experiments, using PlanetLab nodes as video sources. Table 4.1 shows the measured throughput values and end-to-end delays between three pairs of conference participants with diverse geographic locations, comparing the performance of a P2P overlay link with that of routing through the Amazon EC2 cloud. In the latter case, each participating user connects to its closest datacenter in the EC2 cloud, forming a three-hop path. For example, users in Beijing and Seoul will connect to the datacenter located in Japan, and users in Cambridge, UK and Moscow will connect to the datacenter in Ireland. As is self-
explanatory in the table, when routed through the cloud, all three pairs have enjoyed higher throughput values (substantially higher in two of the three pairs), yet this is achieved with similar or even shorter end-to-end delays, as compared to the overlay link in a P2P solution.

Table 4.1: Conferencing with P2P overlays or via the cloud? A comparison of throughput and end-to-end delay.

<table>
<thead>
<tr>
<th>Cloud/P2P</th>
<th>Throughput (Mbps)</th>
<th>Delay (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toronto-Beijing</td>
<td>2.202/0.179</td>
<td>171.6/148.8</td>
</tr>
<tr>
<td>Cambridge-Sao Paulo</td>
<td>1.687/1.432</td>
<td>103.4/204.6</td>
</tr>
<tr>
<td>Seoul-Moscow</td>
<td>7.189/1.103</td>
<td>201.7/436.9</td>
</tr>
</tbody>
</table>

We will revisit such a comparison between Airlift and Celerity with a more elaborate set of experiments in the evaluation part of this chapter. Suffice it to say, there exists a clear performance advantage to provide video conferencing as a cloud service.

4.2.2 Airlift: Design Objectives and Choices

Towards the design of a new application-layer protocol in the inter-datacenter network, we target a number of important objectives.

**Performance.** The best possible incentive that can be touted to attract users to establish video conferences using Airlift is its superior performance, with respect to higher video flow rates from each of the participants in a live conference, while still maintaining an acceptable end-to-end delay. Airlift should, first and foremost, be designed with performance in mind. From the cloud service’s point of view, Airlift will need to be designed to maximize the total throughput across all conferences, subject to constraints with respect to end-to-end delays and basic fairness across conferences.

**Simplicity.** As a cloud service, Airlift should be conceptually simple to use, and work as a full-service broker. A participating user in a conference should only need to connect to the “cloud,” and to start transmitting packets from its video source after a connection is established.
The “cloud” should provide informative feedback to the user as packets arrive, so that the user can adequately increase or throttle its video flow rate by varying parameters of its video codec. In this sense, as long as a packet is acknowledged by the “cloud,” the user will have complete “peace of mind” that the packet will be delivered intact and on time to other participants in the conference, subject to a typical end-to-end delay constraint. Such a conferencing cloud service can be thought of as a full-service broker, and is akin to FedEx, as it delivers packages to destinations around the world on time, using its dedicated cargo fleet.

But what is the “cloud” that a participating user should connect to? Our design in Airlift has intentionally left the decision open with respect to which datacenter that a user should connect to, as existing work has already covered this complementary problem quite well. It is typical to select an appropriate datacenter by taking advantage of the customized IP address returned by DNS servers. Alternatively, users can outsource datacenter selection to third parties [78], with customizable mapping policies. Since video conferencing is sensitive to end-to-end delays, the recommended mapping policy is to choose the “closest” datacenter with respect to delay, using any of the existing selection protocols that can be tailored to consider client proximity (e.g., [78]).

**Scalability.** Datacenters operated by a cloud provider are often inter-connected with high-capacity links. As such, each inter-datacenter link may be able to carry thousands of video flows from different sources and in different conferences simultaneously. This brings the challenge of scalability to the spotlight, in that any online algorithm in the Airlift protocol needs to complete its computation in real-time, so that a large number of conferences can be routed through the inter-datacenter network efficiently and without much fanfare.

To be more scalable, we believe that all the video flows from different participants — in their respective conferences — need to be aggregated, provided that these participants connect to the same source datacenter, and are destined to the same subset of destination datacenters, which, in turn, are responsible for delivering them to all other participants in each of the conferences. To put it simply, we wish to aggregate all the video flows routed through the same
source datacenter and transmitted to the same subset of destination datacenters, regardless of which conference they belong to. Each of these aggregated sessions is inherently a multicast session in the inter-datacenter network.

Considering only aggregated sessions, rather than individual conferences that use the cloud as a service, makes Airlift much more scalable. For example, in order to maximize the total throughput of all conferences routed through the inter-datacenter network, we only need to maximize the total throughput across all aggregated sessions in our problem formulation, with a significantly reduced number of variables that need to be determined. To be more precise, in an inter-datacenter network with $N$ datacenters, the maximum number of aggregated sessions is $\sum_{i=2}^{N} i \cdot \left( \begin{array}{c} N \\ i \end{array} \right)$. With 7 datacenters in the Amazon EC2 cloud, the maximum number of simultaneous aggregated sessions is only 441, which may be an order of magnitude smaller than the total number of participants in all the concurrent conferences routed through the Airlift cloud service.

4.3 Maximizing Total Throughput in the Cloud

In a nutshell, a key idea in the design of Airlift is to take full advantage of the available inter-datacenter capacity in the cloud, so that the total throughput across all conferences is maximized, subject to delay and fairness constraints. We precede our protocol design with a theoretical formulation of this problem.

4.3.1 Feasible Paths Satisfying a Delay Bound

Let us consider an inter-datacenter network with multiple datacenters that are geographically distributed around the world, operated by the same cloud provider. These datacenters form a complete directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ indicates the set of datacenters, and $\mathcal{E}$ indicates the set of directed edges inter-connecting them. For each directed edge $e \in \mathcal{E}$, we use a positive real-valued function $C(e)$ to denote its available capacity, which is the maximum available rate
of packet transmission on $e$.

We use $S^i$ to denote the source datacenter in an aggregated session $i$, and $R^i_j$, $j = 1, 2, \ldots, k_i$ to denote the set of $k_i$ destination datacenters in session $i$. If we overlook fairness concerns for a moment, our objective is to maximize the total throughput of all the aggregated sessions in $\mathcal{G}$, as long as the end-to-end delays from $S^i$ to each of $R^i_j$, $j = 1, \ldots, k_i$ are acceptable, i.e., they do not violate a certain delay bound, $D_{\text{max}}$.

Let us now examine such a delay constraint with a microscope. Each directed edge $e$ in $\mathcal{E}$ has a corresponding propagation delay, $d(e)$, which is readily measurable in practice. Assuming that queueing delays on a relaying datacenter are minimal with the use of small buffers, the end-to-end delay from $S^i$ to each $R^i_j$ can be estimated as the sum of all propagation delays, on each of the edges along the acyclic path that packets follow. Considering our objective of maximizing the total throughput of all aggregated sessions as a variant of the maximum flow problem, it is conceivable that packets from $S^i$ to each $R^i_j$ may need to follow multiple acyclic paths, rather than a single one. We need to make sure that the end-to-end delay on any of these acyclic paths does not violate the delay bound that we impose; in other words, we need to exclude paths that violate such a bound, and only consider the set of feasible paths — denoted by $\mathcal{P}^i_j$ — that do not. More formally:

$$\mathcal{P}^i_j = \{ p \mid p \text{ is an acyclic path from } S^i \text{ to } R^i_j \text{ s.t. } \sum_{e \in p} d(e) \leq D_{\text{max}} \}.$$ 

Given the inter-datacenter graph $\mathcal{G}$ and the delay bound $D_{\text{max}}$, one can easily find the set of all feasible paths $\mathcal{P}^i_j$ from $S^i$ to $R^i_j$ using a simple variant of the depth-first search algorithm, where the search only continues if, with the path obtained so far, there are no cycles and the delay bound $D_{\text{max}}$ has not yet been violated. In our subsequent formulation of the problem, we have the convenience of only considering the set of feasible paths $\mathcal{P}^i_j$. 
4.3.2 The Problem of Maximizing Total Throughput

On the surface, it appears that the problem of maximizing the total throughput of all aggregated sessions in $G$ corresponds to the traditional multi-commodity maximum flow problem. Unfortunately, this is not the case, simply because aggregated sessions\(^1\) are aggregated *multicast* sessions by nature, rather than *unicast* sessions between source-destination pairs as in the multi-commodity maximum flow problem. In essence, packet transmission in a multicast session is more efficient than in multiple unicast sessions, due to the ability for a datacenter to replicate and forward packets to its downstream datacenters in a multicast tree.

To maximize the throughput of a multicast session in $G$, traditional wisdom resorts to *Steiner tree packing* [22]. As an NP-Complete problem, Steiner tree packing seeks to find the maximum number of pairwise edge-disjoint Steiner trees, in each of which the datacenters involved in the session remain connected. To reduce its complexity, existing work on P2P video conferencing [22] packs only depth-1 and depth-2 trees. However, packing Steiner trees within each session is still computationally prohibitive, due to the large number of concurrent sessions.

Fortunately, the concept of *network coding* provides us with a way out of the woods. As we have introduced in Chapter 2, *network coding* [10] extends the capabilities of nodes in a network session: from basic forwarding (as in the maximum flow problem) and replication (as in multicast), to coding in Galois fields. For a multicast session in any directed acyclic graph, if a rate $x$ can be achieved from the source to each of the destinations independently, it can also be achieved for the entire multicast session [10]. In other words, network coding has the power of resolving the competition among different source-destination pairs for edge capacities. To take advantage of such power, Li *et al.* [51] introduced the concept of *conceptual flows*, defined as network flows that co-exist in the network without contending for edge capacities if they are destined to different destinations, each of which is from a source to a destination, transmitted

---

\(^1\)When it is clear from the context of our discussions, *aggregated sessions* in the inter-datacenter network is simply referred to as *sessions* from this point onwards in this chapter.
in a coded form.

To our surprise, inspired by [51], the idea of conceptual flows allows us to formulate the problem of maximizing total throughput as the following linear program, which can be solved by a standard LP solver:

\[
\begin{align*}
\text{max} & \quad \mathcal{X} \\
\text{s.t.} & \quad \mathcal{X} \leq \sum_{p \in \mathcal{P}_{ij}} x^i_j(p) / w_i, \forall i, j = 1, \ldots, k_i \\
& \quad \sum_{p \in \mathcal{P}_{ij}(e)} x^i_j(p) \leq x^i(e), \forall i, j = 1, \ldots, k_i \\
& \quad \sum_i x^i(e) \leq C(e), \forall e \in \mathcal{E} \\
& \quad x^i_j(p) \geq 0, x^i(e) \geq 0, \mathcal{X} \geq 0, \\
& \quad \forall p \in \mathcal{P}_{ij}, \forall i, j = 1, \ldots, k_i, \forall e \in \mathcal{E}. \tag{4.4}
\end{align*}
\]

The objective of this linear program is to maximize the total throughput, which is the sum of flow rates in all the multicast sessions, \( \mathcal{X} \). In each session, its flow rate is the minimum of the flow rates that can be independently achieved from the source to each of the destinations in the session. In constraint (4.1), \( w_i \) is used to provide weighted proportional fairness across different sessions, and \( x^i_j(p) \) represents the conceptual flow rate from \( S_i \) to \( R^i_{ij} \), along an acyclic path \( p \) in the set of feasible paths \( \mathcal{P}_{ij} \). Since the flow rate is specified along a particular path \( p \), the flow conservation constraint for a conceptual flow is implicitly satisfied.

Since conceptual flows destined to different destinations within the same session do not compete with one another for edge capacity, the effective flow rate within a session \( i \) on edge \( e \) is \( x^i(e) = \max_j \sum_{p \in \mathcal{P}_{ij}(e)} x^i_j(p) \), where \( \mathcal{P}_{ij}(e) \) represents the set of paths in \( \mathcal{P}_{ij} \) that uses edge \( e \). Since the \( \max \) function is not linear, this constraint is relaxed to constraint (4.2). Finally, constraint (4.3) reflects the fact that the summation of the effective flow rates of different sessions should not exceed the capacity of an edge, as they contend with one another for edge capacities. Important notations so far are presented in Table 4.2.

A feasible solution to our linear program provides the conceptual flow rates \( x^i_j(p) \) along
Table 4.2: Important mathematical notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{V}$</td>
<td>the set of datacenters operated by a single cloud provider</td>
</tr>
<tr>
<td>$\mathcal{E}$</td>
<td>the set of directed edges connecting datacenters in $\mathcal{V}$</td>
</tr>
<tr>
<td>$C(e)$</td>
<td>the available capacity on edge $e$</td>
</tr>
<tr>
<td>$d(e)$</td>
<td>the propagation delay on edge $e$</td>
</tr>
<tr>
<td>$S^i$</td>
<td>the source datacenter in session $i$</td>
</tr>
<tr>
<td>$R^i_j$</td>
<td>destination datacenters in session $i, j = 1, \ldots, k_i$</td>
</tr>
<tr>
<td>$\mathcal{P}^i_j$</td>
<td>the set of feasible paths from source $S^i$ to each destination $R^i_j$</td>
</tr>
<tr>
<td>$\mathcal{X}$</td>
<td>the common weighted throughput across all the sessions</td>
</tr>
<tr>
<td>$x^i_j(p)$</td>
<td>the conceptual flow from source $S^i$ to destination $R^i_j$ along a path $p \in \mathcal{P}^i_j$</td>
</tr>
<tr>
<td>$x^i(e)$</td>
<td>session $i$’s effective flow rate on edge $e$</td>
</tr>
<tr>
<td>$w_i$</td>
<td>the weight of session $i$ ($0 \leq w_i \leq 1$)</td>
</tr>
<tr>
<td>$\gamma(e)$</td>
<td>the effective length assigned on link $e$</td>
</tr>
<tr>
<td>$\beta^i_j(p)$</td>
<td>the virtual length assigned on path $p$ in feasible path set $\mathcal{P}^i_j$</td>
</tr>
<tr>
<td>$\alpha^i_j$</td>
<td>the minimum weighted virtual path length in aggregation session $i$ to destination $R^i_j$</td>
</tr>
</tbody>
</table>

all feasible paths for each destination, within every session. The effective flow routing scheme $x^i(e)$ for each session, as well as the feasible total throughput $\mathcal{X}$, are all guaranteed to be non-negative, with constraint (4.4). Since only feasible paths are considered in our linear program, the delay constraint is naturally satisfied.

As an example using the inter-datacenter network that we have shown previously in Fig. 4.1, Fig. 4.4 shows the optimal solution obtained by solving our linear program using a standard LP solver. To keep such a conceptual example simple, we assume that all the edge capacities are 10 Mbps. The value labeled on each edge $e$ in the figure indicates its propagation delay $d(e)$, which is the same in both directions. Let us now consider two sessions, $S1$ and $S2$. In $S1$, video flows are transmitted from $D1$ to $D4$ and $D5$; and in $S2$, they are transmitted from $D2$ to $D3$ and $D5$. If the delay constraint $D_{max}$ is set to be 100 milliseconds, some of the paths need to be excluded from the set of feasible paths. The conceptual flows along all the feasible
paths in each session are shown in the figure, where their widths indicate the corresponding conceptual flow rates.

As we can see, the feasible path in $S_1$ to $D_4$ is $D_1 \rightarrow D_2 \rightarrow D_4$, with a flow rate of 10 Mbps for the corresponding conceptual flow; the feasible paths in $S_1$ to $D_5$ are $D_1 \rightarrow D_2 \rightarrow D_3 \rightarrow D_5$ and $D_1 \rightarrow D_2 \rightarrow D_5$, each with 3.9 Mbps and 6.1 Mbps respectively. Similarly, feasible paths in $S_2$ to $D_3$ are $D_2 \rightarrow D_3$ and $D_2 \rightarrow D_5 \rightarrow D_3$, with 6.1 Mbps and 3.9 Mbps, respectively; and feasible paths in $S_2$ to $D_5$ are $D_2 \rightarrow D_5$ and $D_2 \rightarrow D_3 \rightarrow D_5$, with 3.9 Mbps and 6.1 Mbps, respectively. In the optimal solution, the total throughput in both sessions is 20 Mbps in this example, and edge capacities along the feasible paths have been saturated.

To better illustrate the concept of conceptual flows, we examine the edge from $D_2$ to $D_5$. Though two conceptual flows in $S_2$ — each with a rate of 3.9 Mbps — pass through this edge, the effective flow rate in $S_2$ on this edge remains to be 3.9 Mbps, since the power of network coding guarantees that conceptual flows destined to different destinations in the same session do not compete for edge capacities. On the other hand, the effective flow rate in $S_1$ on this edge is 6.1 Mbps, competing with the effective flow in $S_2$ for the edge capacity.

With the use of conceptual flows, the optimal solution of our linear program is quite expressive. It may impose that an incoming flow be replicated, be split, or that multiple incoming flows be merged using network coding, and then forwarded along outgoing edges. In our exam-
ple, consider datacenter $D2$. Its incoming flow of 10 Mbps in session $S1$ is not only forwarded to $D4$ directly, but also split and forwarded at the same time to $D3$ and $D5$, with an outgoing flow rate of 3.9 Mbps and 6.1 Mbps, respectively. The flexibility and power of the optimal solution expressing a wide variety of forwarding strategies at each datacenter have provided a solid foundation for Airlift, yet they also pose a challenge to our protocol design, to make sure that the optimal solution can be realized faithfully in practice.

4.4 Airlift: Protocol Design

With the available capacity and propagation delay on each inter-datacenter edge as input, the optimal solution along the set of feasible paths provides the complete plan to start actual packet transmission: In each conceptual flow, the optimal solution computes its flow rate $x_{ji}(p)$, along the path $p$ in a session $i$ from the source $S_i$ to the destination $R_j$ that packets will follow.

The design objective of the Airlift protocol is to faithfully realize the complete plan that the optimal solution provides in a real-world implementation, with as little gap between theory and practice as possible. As we shall soon observe, such a goal is challenging to achieve; and subsequent experimental evaluations of our Airlift implementation will focus on how tradeoffs in our design will contribute to the gap between theory and reality.

4.4.1 Transport with Network Coding

In Chapter 2, we introduced a primer on random network coding. We now show how random network coding can be practically implemented by using the notion of generations $[24, 42]$, in the context of video streams. A generation of live video is divided into $n$ packets (called the generation size) $b = [b_1, b_2, \ldots, b_n]^T$, where each packet has a fixed number of bytes, $k$. To code a new coded packet $x_j$, the source first independently and randomly chooses a set of coding coefficients $[c_{j1}, c_{j2}, \ldots, c_{jn}]$ in $GF(2^k)$, one for each original or coded packet it has buffered. It then produces one coded packet $x_j = \sum_{i=1}^{n} c_{ji} \cdot b_i$. The destination decodes as soon
as it has received \( n \) linearly independent coded packets \( x = [x_1, x_2, \ldots, x_n]^T \). It first forms an \( n \times n \) coefficient matrix \( C \), using the coefficients of each packet \( b_i \), which are embedded in the packet. Each row in \( C \) corresponds to the coefficients of one coded packet. It then recovers the original packets \( b = [b_1, b_2, \ldots, b_n]^T \) as \( b = C^{-1}x \). Gauss-Jordan elimination is used in such a decoding process, performed progressively as coded packets are being received. The inversion of \( C \) is only possible when its rows are linearly independent, i.e., \( C \) is full rank.

*Airlift* uses UDP as its transport protocol on each inter-datacenter edge, the rate of which is controlled by an implementation of TCP-Friendly Rate Control (TFRC) [34] at the application layer. In such a context, network coding has been applied extensively in the *Airlift* protocol design. This is not only because our problem formulation hinges upon the concept of conceptual flows made possible by network coding, but also since random linear codes are rateless erasure codes, and coded packets — each with its own vector of randomly chosen coefficients — can be generated *ad infinitum*. As long as \( n \) linearly independent packets are received, they are sufficient to recover the original generation. This is a perfect match to UDP as a transport protocol: losing some coded packets is no longer a concern, as more from the source will be arriving soon, provided that the source receives some form of feedback.

Unfortunately, one seemingly trivial question — on an implementation detail when network coding is used at the source — puts the very idea of using network coding at risk.

### 4.4.2 Bandwidth Overhead vs. Delay: A Dilemma

It is the question of what an appropriate *generation size*, \( n \), is, which the source datacenter should use when it applies network coding. In other words, shall we use a smaller number of packets in each generation, or a larger number of them?

Let us consider the outcome of using a smaller number of (say, 5) packets. In the example illustrated in Fig. 4.5(a), we can observe that a small generation size will lead to significant bandwidth overhead. At the time when the source finishes sending all 5 packets, the acknowledgement, to be sent when the entire generation is completely received and decoded at the
destination, has not yet arrived at the source. Such an acknowledgement may only be received by the source after a round-trip time since the last coded packet in the generation has been sent. During such a period of time, the source will have no choice but to either stop sending, in which case its instantaneous flow rate is throttled to zero and outgoing bandwidth is idled; or to keep sending more coded packets, in which case these packets are redundant and useless when received by the destination, leading to significant bandwidth overhead. As the number of packets in a generation becomes smaller, the overhead of such redundant packets, as a percentage, will be even more significant.

![Diagram of bandwidth overhead vs. decoding delay](image)

(a) Using a small generation size: more bandwidth overhead
(b) Using a large generation size: longer decoding delay

Figure 4.5: Bandwidth overhead vs. decoding delay: a dilemma over the choice of the generation size.

Using a much larger generation size (say, a few hundred packets) would certainly mitigate such overhead, but as the example in Fig. 4.5(b) illustrates, it leads to a substantially longer decoding delay at the destination. Consider the analogy that the destination holds a “bucket” — the capacity of which is the generation size — waiting for coded packets to arrive and fill the bucket. Due to the nature of random linear codes, the destination will have to wait till the bucket is almost full in order to recover the first original packet in the generation, even if Gauss-Jordan elimination is used. Such a waiting period adds an additional decoding delay, which becomes longer as the generation size becomes larger. With a generation size of 128
packets, a packet size of 1 KB, and a source flow rate of 64 KB/sec, the decoding delay can be as long as 2 seconds, which is excessive considering typical end-to-end delay constraints in a video conference.

An additional piece of bad news is that the sliding window approach — designed by Sundararajan et al. [71] for incorporating network coding into TCP as a transport protocol — does not solve this dilemma. In [71], a TCP source transmits random linear combinations of packets in its (sliding) congestion window, and advances the window as it receives an acknowledgment from the destination. These acknowledgments are in the form of the degree of freedom of the “bucket” that the destination holds; i.e., an original packet is acknowledged as it is received in a coded form, even before decoding is complete. The size of a generation in this approach is the size of the sliding congestion window, which reflects the bandwidth-delay product on the end-to-end path, and can lead to excessively large decoding delays at the destination.

To add insult to injury, the high probability of receiving linearly dependent packets at the destination is a grave concern if a small generation size is used. It has been shown in [33] that the overhead due to linearly dependent packets can be up to 18% if each generation has 8 packets. Fortunately, as proposed by [33], the source can use systematic Reed-Solomon codes rather than random linear codes to mitigate such linear dependence, and such a change does not affect any theoretical benefits of network coding.

4.4.3 The Protocol at Source and Destination Datacenters

Our design of the Airlift protocol demonstrates that a blend of valuable elements in existing protocols can go a long way towards solving the dilemma.

Decoupling the notion of a generation from a sliding window. First, we must choose to use a small generation size to reduce decoding delays, as we are not in a position to compromise on the end-to-end delay constraint in a live video conference. In order to mitigate the bandwidth overhead, our design in Airlift decouples the notion of a generation from a sliding window: coded packets are generated within a generation, while a sliding window represents all the
packets that have been sent but not yet acknowledged by the destination. These two windows do not have to be the same, and the sliding window can contain a large number of generations. Our design makes it possible to use a sliding window size that corresponds to the bandwidth-delay product between the source and the destination, while keeping the generation size small.

With a generation size of $n$, the source datacenter transmits $n \cdot (1 + \delta)$ coded packets from the current generation by using systematic Reed-Solomon codes, where $\delta$ can be adapted on-the-fly to reflect the amount of redundancy we wish to add to combat packet loss. Afterwards, the source moves onward and starts to transmit $n \cdot (1 + \delta)$ coded packets from the next generation in the sliding window. The rate at which the source transmits in session $i$ is $\min_j \sum_{p \in P_j} x^i_j(p)$, which is part of the complete plan stipulated by the optimal solution.

Since packets from multiple generations are in the pipeline, the destination datacenter will need to hold multiple active buckets accordingly, each containing packets received so far within the same generation. Upon receiving each coded packet, it is placed into its corresponding bucket after performing Gauss-Jordan elimination, so that the corresponding coding matrix in each bucket is guaranteed to be in reduced row-echelon form (RREF). As $n$ linearly independent packets arrive in a bucket, all the original packets will be recovered and the bucket will no longer be active.

**Acknowledging the degrees of freedom in all active buckets.** Upon receiving each coded packet, the destination immediately sends an acknowledgment to the source. The acknowledgment contains the degrees of freedom in all active buckets, corresponding to the number of linearly independent coded packets received in each generation that has not yet been fully decoded. As it receives and examines each acknowledgment, the source transmits a sufficient number of additional coded packets from each of the generations contained in the acknowledgment, starting from the oldest generation, but not including the current generation, from which the source is still in the process of transmitting coded packets.

---

2In our real-world implementation, $\delta$ is adapted by the source periodically, based on the packet loss rate measured at the source in its TFRC implementation.
For example, in Fig. 4.6 with a generation size of 5, once the source observes that the destination has received 4 packets from generation 1, 3 packets from generation 2, and 2 packets from generation 3 (the current generation at the source), it will immediately transmit one new coded packet from generation 1 and two new packets from generation 2, before it resumes the process of transmitting packets from the current generation.

Since an aggregated session is a multicast session from a source to multiple destinations, the source will only remove the oldest existing generation and advance its sliding window once all the destinations in the session have indicated that they have successfully decoded the generation (in that it is not included in their acknowledgments).

### 4.4.4 Realizing Conceptual Flows with Source Routing

The optimal solution to our linear program contains a number of conceptual flows in each session, each consisting of a flow rate and a path from the source to one of the destinations. From the perspective of how packets in an actual flow are processed in reality, there are three distinct cases when multiple conceptual flows pass through an intermediate datacenter: (1) Packets in an actual flow are to be replicated and forwarded to multiple outgoing edges; (2) an actual flow is to be split and forwarded, with different portions of its packets destined to different outgoing edges; and (3) packets in multiple actual flows — from the same source
and destined to different destinations — are to be *merged* with random network coding. These cases are illustrated in Fig. 4.7. An intermediate datacenter may be responsible for handling multiple cases concurrently, as in the example of datacenter $D2$ in Fig. 4.4.

![Diagram](image)

**Figure 4.7:** Three cases of realizing conceptual flows passing through an intermediate datacenter.

Since the first two cases do not require network coding on the intermediate datacenter, they are realized in *Airlift* with the use of *source routing*. The source has full knowledge of all its conceptual flows in the optimal solution, and by allocating outgoing packets to each of the conceptual flows based on their flow rates, it is able to compute the complete *tree* that an outgoing packet should follow to reach its destination(s), and include the tree in the packet header\(^3\). A packet then becomes *self-routing*, in that an intermediate datacenter only needs to examine its header, extract its next-hop datacenters in the tree, and then forward it to its next hop, making copies as needed.

To realize the third case, all datacenters in *Airlift* finds path *overlaps* between different conceptual flows if they are destined to different destinations in the same session, by examining the complete plan given by the optimal solution. If such an overlap exists, as shown in Fig. 4.7(3), the corresponding datacenter will produce random linear combinations of packets from actual incoming flows, and transmit a *merged* outgoing flow according to the rate on the outgoing edge, given by the optimal solution.

All such random linear combinations are only performed on packets from the same gener-

\(^3\)It is a *tree* — rather than a *path* — that a packet should follow when being routed, since conceptual flows to different destinations may share the same outgoing edge from the source.
ation at the source. If packets from the same generation are not readily received from all the incoming flows, the merging process will wait for a timeout period with a buffer of recently received packets. After the timeout expires, it simply merges packets that it was able to receive so far. Since such a timeout period adds to the end-to-end delay, it is not feasible to use a long timeout value.

4.4.5 A Full-Service Broker to Conference Participants

With simplicity in mind, Airlift is designed to serve as a full-service broker: a participant in a conference would simply select a datacenter to connect to, and start transmitting to this datacenter using an adaptive video source rate, coupled with an adaptive video codec. The datacenter adds this new video source to one of its aggregated sessions that share the same subset of destination datacenters. As a full-service broker, any datacenter in Airlift maintains full knowledge about all active conferences, including the list of participants and the datacenter each of them connects to. Updates are simply broadcast to all other datacenters in the cloud.

Original video packets from each participant are transmitted to the datacenter over UDP and TFRC, and with the same Airlift protocol. The only revision in the protocol is an additional form of periodic acknowledgment: the datacenter suggests a new video source rate to the participant, who then adapts its sending rate to be the minimum of the suggested rate and what TFRC imposes.

How does Airlift compute the video source rate that a datacenter suggests to each connected participant? Since the source rate of the aggregated session, \( \min_j \sum_{p \in P_j} x_j(p) \), is specified by the optimal solution of our linear program, the datacenter simply allocates such a source rate to all participants in the aggregated session. The allocation is to be max-min fair, in that if a participant transmits at a lower rate than its fair allocation due to its last-mile bottleneck, it will be allocated what it needs, plus a certain margin to allow upward allocation adjustments if the last-mile bottleneck bandwidth improves; otherwise, it will be allocated its fair share. With our design, the source rate of the session may not be fully allocated to current participants — in
the case that the bottleneck for all of them resides in the last-mile link. Such a residual source rate makes it straightforward to accommodate newly arriving participants, without the need to re-optimize globally by solving a revised linear program.

One remaining challenge is to determine the weight, $w_i$, of aggregated session $i$, which is used as input in our linear program. Again, to keep the design simple, $w_i$ will be proportional to the number of participants in session $i$, and $\sum_i w_i = 1$. This guarantees basic fairness across participants in all the conferences, when their video sources share the inter-datacenter capacity.

4.5 Real-World Implementation and Evaluation

Airlift is the name for not only our application-layer protocol, but also its real-world implementation. The basic unit for our Airlift implementation is a broker, which runs in a VM in one of the datacenters. Beyond its role as a full-service broker to conference participants, a broker is also responsible for two core features in the protocol.

*Multi-generation sliding window with network coding.* A broker is responsible for implementing the Airlift protocol at the source and destination datacenters, which involves sending and acknowledging packets in multiple generations within the source sliding window, coded with random linear codes. Since random network coding lies at the core of the Airlift protocol, our implementation of network coding is optimized with Intel SSE2 acceleration, which offers a five-fold performance gain on average, as compared to a vanilla implementation.

*Packet processing and forwarding.* As an intermediate datacenter in a multi-hop path, a broker is capable of replicating, splitting, and merging incoming flows, and of forwarding packets based on source routing information embedded in their headers. Using asynchronous event-driven networking, our implementation is highly efficient, designed with performance in mind. It supports major UNIX variants and Windows (both are typical in cloud VMs), and incurs less memory and CPU overhead compared to the traditional thread pool concurrency model, especially at high packet processing rates.
A centralized optimizer in our implementation receives periodic reports from all the brokers with respect to measured capacities and propagation delays on inter-datacenter edges. It then generates the set of feasible paths, and solves the linear program that maximizes the total throughput across all the aggregated sessions. The optimal solution is then transmitted back to all the brokers for them to route outgoing packets using source routing.

We choose to use the Amazon EC2 inter-datacenter network in our experiments, which is one of the predominant Infrastructure-as-a-Service (IaaS) cloud providers. We have launched 7 standard on-demand small VM instances on all 7 EC2 datacenters, with a broker in each. We used a large VM instance located in the Virginia EC2 datacenter to host our centralized optimizer.

### 4.5.1 Airlift vs. Celerity: A Performance Comparison

With our new implementation of *Airlift*, our first set of experiments is to compare its performance with *Celerity* [22], in a comparable real-world conference. Since *Celerity* is designed to maximize the total throughput within a single conference only, we initiated only one conference with *Airlift* for a fair comparison. The challenging dumbbell topology, shown in Fig. 4.2 and used extensively in [22], was adopted in our comparison studies. As an example, in the Toronto-Beijing topology, we used two PlanetLab nodes in Toronto and two more in Beijing as conference participants, forming 4 *sessions*, each corresponding to a video source at one of the participants. In *Airlift*, users in Toronto were connected to their nearest datacenter in Virginia, and users in Beijing were connected to their nearest datacenter in Tokyo, leading to two aggregated sessions in the EC2 inter-datacenter network.

Our experimental results with both Celerity and *Airlift* have been summarized in Table 4.3, over three different pairs of geographic locations: Toronto – Beijing, Vancouver – Berlin, and Seoul – Rio de Janeiro. As we can easily observe, with respect to the total throughput of all

---

4 Amazon EC2 datacenters are located at Oregon, Northern California, Virginia, Ireland, Tokyo, Singapore, and San Paulo.
the sessions in the conference, *Airlift* was able to offer a substantial performance advantage, on the order of 325% – 2387% (i.e., 24 times better than Celerity). Yet, *Airlift* did not suffer from a noticeable disadvantage with respect to end-to-end delays: it typically incurred a similar end-to-end delay as compared to Celerity, which uses overlay links. The substantial throughput advantage with similar delays have made *Airlift* a clear winner with respect to performance in our experiments.

*Airlift*'s throughput advantage can be explained by abundant inter-datacenter network capacities, and the excellent performance on end-to-end delays can be attributed to design and implementation choices in our protocol, which have collectively minimized queueing delays as packets were forwarded through the two intermediate datacenters. Our measurements, not presented in the table, have shown that the queues on these datacenters have remained *empty* at most times throughout our experiments.

Table 4.3: Airlift vs. Celerity: a comparison with respect to the total throughput and the maximum end-to-end delay across all 4 sessions in the same conference.

<table>
<thead>
<tr>
<th>Airlift/Celerity</th>
<th>Total throughput (Mbps)</th>
<th>End-to-end delay (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toronto-Beijing</td>
<td>14.11/4.04</td>
<td>169.8/142.3</td>
</tr>
<tr>
<td>Vancouver-Berlin</td>
<td>34.32/1.38</td>
<td>137.2/104.9</td>
</tr>
<tr>
<td>Seoul-Rio</td>
<td>20.32/2.32</td>
<td>228.6/203.5</td>
</tr>
</tbody>
</table>

During the startup phase of a conference, another important disadvantage of Celerity is the longer period of time for it to ramp up its throughput in each session to the steady state. Fig. 4.8 and 4.9 have shown the per-session throughput over time for Celerity and *Airlift*, respectively, as a conference started up. we note that it took Celerity three times longer to ramp up to a steady-state throughput of three times lower. Such subpar performance is due to the nature of the Celerity protocol design: its rate control algorithm that governs overlay link rates does not have any knowledge of the underlying physical topology, and would have to take some time to converge to optimality based on online measurements. Such a challenge is nonexistent in
Airlift, as it simply routes packets through the nearest datacenter over the cloud.

![Graph](image)

**Figure 4.8:** Celerity takes 15 seconds to ramp up its per-session throughput to the steady state of 0.9 Mbps on average.

![Graph](image)

**Figure 4.9:** Airlift takes less than 5 seconds to ramp up its per-session throughput to the steady state of 3.5 Mbps on average, a three-fold improvement.

### 4.5.2 Performance with Multiple Aggregated Sessions

We now proceed to evaluate Airlift’s performance with multiple aggregated sessions in the EC2 inter-datacenter network. For this purpose, we launched 10 actual conferences, with participants connecting to their nearest datacenters. In order to simulate the geographic diversity of conference participants, the number of datacenters involved in each conference is generated with a discrete uniform distribution in the range of \([2, 7]\), resulting in a total of 35 sessions.

As measured by the brokers every 3 minutes on their respective datacenters and reported to the centralized optimizer, the edge capacities ranged from 20.9 to 130.8 Mbps, while their propagation delays ranged from 11.3 to 441.7 msec. These capacity and delay values were used as input to our linear program, which the optimizer used to obtain the optimal solution. With 10 conferences and 35 aggregated sessions, we imposed an end-to-end delay constraint, \(D_{\text{max}}\), of 300 milliseconds. As a result, the optimal solution, as computed by the optimizer,
involved a total of 1869 feasible paths, with 54 feasible paths per session on average, and a maximum of 5 hops in a feasible path. With a reasonable $D_{\text{max}}$, the large number of feasible paths is indeed a piece of good news, in that the optimizer would have the freedom of using all of these paths to deliver packets, saturating edge capacities as much as possible to improve the total throughput.

Again, as a conferencing cloud service, we are most concerned with the achievable throughput and end-to-end delays in our aggregated sessions, both to be measured in actual experiments. Fig. 4.10 shows the throughput observed in each of the aggregated sessions, while Fig. 4.11 shows the maximum end-to-end delays observed from the source to the destinations in 10 sessions in EC2. With stable edge capacities in EC2 over the short term, the observed throughput values in Fig. 4.10 were exactly the same as what the optimizer has computed, thanks to Airlift capability of transmitting, coding, forwarding, and decoding packets at the designated flow rates computed by the optimizer. The end-to-end delays are well controlled, again thanks to the ability of our implementation to keep queueing delays to the minimum while packets traverse through intermediate datacenters on their paths.

Figure 4.10: Observed throughput values in 10 of the aggregated sessions.

Figure 4.11: Observed maximum end-to-end delays from the source to the destinations in 10 of the aggregated sessions.
4.5.3 A Packet’s Life: A Microscopic View

Our readers may be left wondering: What had contributed to Airlift’s substantial performance advantage against Celerity, as well as its superior inter-datacenter performance with multiple aggregated sessions? To best answer this question, we decided to delineate the process of sending, processing, and acknowledging packets, showing fine-granularity details within a single aggregated session involving two destination datacenters. Our experiments ran for a period of 3 minutes.

To saturate inter-datacenter edge capacities on the order of 100 Mbps, let us first examine the transport protocol with multi-generation network coding from the source to a destination. In our experiments, we used a generation size of 10 packets and a packet size of 4 KB. Thanks to our accelerated network coding implementation, we could comfortably reach a session throughput of 59.1 Mbps with a CPU load of just 78% at the source, within a small EC2 VM instance. On the other hand, since the size of the sliding window is decoupled from the generation size, we were able to achieve such a throughput with an average of 36 outstanding generations at the source that are not yet acknowledged by both destinations. Thanks to our application-layer TFRC implementation (based on RFC 3448) and the stability of the EC2 cloud, a negligible packet loss event rate of $4.5 \times 10^{-5}$ was observed. As a result, there were no more than 4 active buckets used concurrently for decoding at both destinations. This minimized the bandwidth overhead of acknowledgments from both destinations back to the source: an average of 1.65 Mbps was used for acknowledgments (with 3905 packets per second and 54 bytes per packet on average), reflecting an overhead of only 2.8%.

The negligible packet loss event rates that we observed have also had a positive effect when it comes to minimizing the bandwidth overhead of using network coding. In our experiments, the number of outdated packets — defined as redundant packets who arrived after their generations have already been decoded — remained in the range of [1%, 4%], as a percentage of all coded packets transmitted. With a low percentage of outdated packets transmitted, the number of linearly dependent packets due to a small generation size was also negligibly small,
accounting for only 0.00004% of all the coded packets in the session.

Under our control in the *Airlift* protocol, there are two factors that affect one-way end-to-end delays in our experiments: (1) the *queue lengths* on intermediate datacenters along a path that packets traversed; and (2) the *decoding delays* at a destination. It turns out that, in terms of the number of packets, the queue lengths we observed on each of the outgoing edges involved in the session were zero most of the time, and were no more than 18 packets in the worse case, as shown in Fig. 4.12. Further, Fig. 4.13 shows the maximum decoding delays observed at runtime every 15 seconds, which represented the time from when a new active bucket was created at a destination to when it was completed decoded. As the figure shows, the decoding delays remained in the range of [10, 27] milliseconds. On average, our measurements have indicated that the queueing and decoding delays combined had accounted for only 23% of one-way end-to-end delays, the bulk of which was attributed to propagation delays on inter-datacenter edges in EC2.

![Figure 4.12](image1.png)  
**Figure 4.12:** Observed queue lengths on each hop along a path (Sao Paulo → Ireland → Virginia → Oregon).

![Figure 4.13](image2.png)  
**Figure 4.13:** Observed decoding delays at both destinations (Oregon and Ireland) over time, measured every 15 seconds.

Finally, Fig. 4.14 shows a microscopic view of a packet's lifetime in our experiment with two destination datacenters. The figure has been annotated by propagation, queueing, and decoding delays that we measured, from the moment the packet was sent by the source data-
center (and acknowledged to the conference participant), to the time it was received by all its destination datacenters (and relayed to the rest of the participants).

Figure 4.14: A packet’s life as measured from the source datacenter (Sao Paulo) to both destination datacenters (Oregon and Ireland).

4.6 Summary

In this chapter, we use multi-party video conferencing as a bandwidth-demanding and delay-sensitive example to explore the possibility of supporting mobile applications with a cloud service. With Google+ Hangouts implemented as a cloud-based client-server video conferencing solution [73], we believe that designing a high-performance cloud service to serve enterprise video conferencing needs is an industry trend that should not be overlooked. In this chapter, we part with the traditional wisdom of a peer-to-peer design; rather, we motivate and present Airlift, a new protocol and its real-world implementation that provide video conferencing as a cloud service by routing video flows through the inter-datacenter network in the cloud, with higher capacities than the public Internet. The design of Airlift is driven by our objective of maximizing the total throughput of all conferences, yet without compromising on end-to-end delays. Intra-session network coding lies at the heart of the Airlift protocol, which is designed with attention to detail, such as a revised transport protocol to better support generation-based network coding. With real-world experiments using PlanetLab nodes and the Amazon EC2
cloud, we show that Airlift is able to deliver on its promises: it is capable of supporting substantially higher video bit rates, yet with similar end-to-end delays as compared to peer-to-peer solutions.
Chapter 5

Maximizing Resource Utilization in the Cloud

5.1 Overview

As we have made very clear in the preceding chapters in this dissertation, media-rich applications running on mobile devices are becoming more resource-hungry, and as a result, cloud computing platforms have been established as a key solution to meet such application demand for resources. The paradigm shift to cloud computing is decisively driven by such a strong demand, from both mobile and enterprise applications. As we have shown in Chapter 1, cloud providers use datacenters to provision a shared pool of computation, storage and bandwidth resources, to be used by mobile applications when the need arises. Since resources at datacenters are shared by using virtualization, applications are allowed to statistically multiplex such resources in the form of virtual machines, such that the overall efficiency of using and managing computing resources can be improved.

We are interested in gesture streaming and video conferencing in Chapter 3 and 4, respectively, mostly for their delay-sensitive nature. We now wish to consider a more traditional video-on-demand streaming service, which requires both computation and bandwidth at the
servers, but are not delay-sensitive. Due to the highly varied resource demand that the service needs to satisfy, video streaming is a prime example where migrating the service to datacenters in the cloud makes more economical sense. Take the video streaming service offered by NetFlix Inc. as an example, with widely varying user demand for bandwidth and the fact that Internet Service Providers (ISPs) bill for 95% of the peak bandwidth usage, it would be much more economical to use cloud services rather than deploying privately owned media servers.

We have also observed from our first-hand real-world experiences that demand for video streaming services may peak at different times. Fig. 5.1 shows the normalized population of two videos in three days by analyzing 200 Gigabytes worth of operational traces that we collected at UUSee Inc. [3], one of the leading peer-assisted video streaming providers in China. As we can see, Video 1 (an on-demand stream) and Video 2 (a live stream) have peaked on August 12 and August 13, 2008, respectively. If privately owned media servers are used by provisioning for peak bandwidth usage, bandwidth will remain severely under-utilized during off-peak times. Therefore, it is an economically sound decision to migrate video streaming services to datacenters in the cloud.

![Figure 5.1: The normalized population of two videos from August 12 to 14, 2008.](image)

Once the decision is made to host video streaming services with datacenters in the cloud, the question becomes how datacenter resources can be better utilized. A datacenter consists of tens of thousands of physical servers. Since videos may reach their respective peak demand at different times, there is an opportunity to increase server utilization ratios by letting multiple
videos share physical resources on the same server. With virtualization ubiquitously used in Infrastructure-as-a-Service cloud platforms, several VMs, each of which packaging one video, are able to be placed on the same physical server and benefit from “on-demand” resource provisioning. Specifically, resources at a physical server are shared among all VMs during off-peak seasons to achieve a higher utilization ratio; once one of the videos encounters its peak in demand, VMs packaging this video can use up all available resources at servers where they are placed, in order to satisfy as many requests as possible.

One possible challenge we may encounter is that servers may become overloaded when a video encounters highly bursty requests, or several videos placed on the same server reach their peak period in demand at the same time. Shown in Fig. 5.1, both Video 1 and Video 2 reached their peaks on August 14. A naive solution will be to move VMs away from overloaded servers to under-utilized ones. Thanks to the support of live migration with virtual machines [58], migrating from one server to another with minimal downtime to the streaming service is feasible in datacenters. With live VM migration, the number of requests being handled at the same time may be effectively increased by migrating VMs with additional resource needed from resource-deficient to resource-rich servers.

Nevertheless, we argue that such migration should be planned with care, by fully exploring all possibilities of utilizing currently available resources to handle the increased requests. Since servers in the datacenter provide resources in three main dimensions of storage, bandwidth and CPU in a tightly-coupled manner, utilization of resources may gradually become severely unbalanced across different dimensions, which implies unnecessary idling of available resources. Since datacenters are expensive to build and to operate, it is a waste of both investment and energy when resources are under-utilized [41].

In this chapter, we seek to design an efficient and practical algorithm to maximize resource utilization, with all three dimensions considered, by migrating VMs across servers in a video streaming datacenter. We first formally formulate the challenge of maximizing system-wide resource utilization as a centralized optimization problem, taking into account practical con-
straints of storage, bandwidth, and CPU cycles. The optimal solution dictates the destination server to which VMs should be migrated. We find that the optimization problem is in the form of a Generalized Assignment Problem (GAP) in integer programming, which is NP-hard and even APX-hard to approximate. Obtaining a near-optimal solution to this problem requires to decouple it into several 0-1 knapsack problems and iterate hundreds of times.

In order to address this challenge, we are inspired by the power of markets in arbitrating decisions of both buyers and sellers in a decentralized fashion. Our solution relates the entire datacenter to a bargaining market. VMs are considered as “commodities” in this market. In our solution, every server makes its decision by participating in this market and bargaining for its desired commodities. We model this market as a Nash bargaining game, and prove that the problem of maximizing resource utilization in a datacenter is equivalent to that of maximizing the joint profit in the Nash bargaining solution. The VM migration decisions are governed by the individually made bargaining strategies in each server in a laissez-faire manner, which avoids additional CPU consumption at a centralized decision maker and reduces the bandwidth cost of VM migration.

5.2 The Problem of Maximizing Resource Utilization

We first present an example to show that VM migration may help to fully utilize resources in all three dimensions to handle more requests, and then formulate the problem of maximizing resource utilization in video streaming datacenters.

5.2.1 Benefits of VM Migration

Satisfying one request for streaming a video, at High-Definition (HD) or Standard-Definition (SD) quality levels, requires different amounts of bandwidth. The variety of existing media formats requires on-demand transcoding, which results in distinct requirements on CPU cycles when processing a video streaming request. As the number of requests for each video fluctuates
over time, the required amount of resources for each video in dimensions of storage, bandwidth and CPU cycles may increase in an unbalanced manner. With live migration of virtual machines (VMs), we may migrate VMs across the boundary of servers in a datacenter, in order to fully explore possibilities of utilizing available resources, which leads to more concurrent requests being handled.

The following conceptual and proof-of-concept example illustrates the potential benefits with such migration. Consider a video streaming datacenter with two servers and three videos. Each of the videos is served by a corresponding VM: VM\(_1\), VM\(_2\), and VM\(_3\) respectively. In this example, Video 1 represents a live video streaming service, e.g., with the standard-definition (SD) quality level, and with network coding adopted in transmission, which require more CPU but less bandwidth resources. Video 2 represents an on-demand streaming (VoD) service, e.g., in HD quality without using network coding during transmission, which requires more bandwidth, but with relatively low CPU demand. Video 3, again, represents a live streaming service, but no network coding is involved in its transmission. Resources required to handle one request for each video are summarized in Table 5.1, along with resource capacities at servers. We note that there are no additional storage resources required for handling each request in video streaming services. The size of each video is assumed to be 4 GB is this example.

Table 5.1: Required resources to accommodate one request in each VM, as well as the resource capacity in each server.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Server 1/2</th>
<th>VM(_1)</th>
<th>VM(_2)</th>
<th>VM(_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage Space (GB)</td>
<td>8</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Bandwidth (Mbps)</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>CPU (MIPS)</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Suppose initially Video 1 is popular and Video 2 and 3 are less popular, as there are five requests for Video 1 and one request for Video 2 and 3, respectively. It is not difficult to find out that the optimal resource utilization strategy is to have the popular video provided by two
servers, each of which is concurrently multiplexed by one unpopular video. That is, placing \((VM_1, VM_2)\) in one server and \((VM_1, VM_3)\) in another server, shown in Fig. 5.2. In this fit, a total of seven requests can be handled at the same time.

![Figure 5.2: The initial fit of three videos on two servers.](image)

Due to the variation of requests over time, the popularity of videos changes as time elapses, with, say, three requests for Video 1, one request for Video 2 and four requests for Video 3 at this time. No possible migration plan exists if we use the naive solution of moving VMs away from overloaded to under-utilized servers, which means one of the eight requests will not be satisfied at the same time due to the unbalanced increase of resource demand in bandwidth and CPU cycles, no matter how the three requests for Video 1 is scheduled. Fig. 5.3(a) indicates one possible case, where one request for Video 1 is directed to Server 2. The heavily utilized CPU in Server 2 results in a drop of one request for Video 3, which also requires CPU. As a consequence, one of the requests can not be satisfied immediately, even as resources are still available on the two servers.

However, if VM migration is conducted in the datacenter, we can swap VM_2 in Server 1 and VM_1 in Server 2, which is shown in Fig. 5.3(b). In such a scenario, the VM, which requires

---

**Figure 5.3: Improved video fit after VM migration.**
more resource in one dimension, is fit into the server with another VM which requires less resource in that dimension in a complementary manner. The eight requests for three videos can be satisfied by the two servers at the same time, which leads to a higher level of resource utilization.

From this example, we can see that by migrating VMs across the boundary of servers, a datacenter is able to better utilize available resources and handle more requests at the same time.

5.2.2 Maximizing Resource Utilization in a Generalized Form

Our primary objective is to find out how VM migration strategies should be designed so that resource utilization in datacenters is maximized. We first present the context of our discussion and a model of a datacenter. Important notations used throughout this chapter is listed in Table 5.2. We first present the context of our discussion and a model of a datacenter.

Instead of restricting ourselves to video streaming datacenters, we discuss this problem in a generalized form so that the designed algorithm is also applicable to datacenters holding general application instances. We consider a datacenter constituting a set of heterogeneous servers, denoted by $\mathcal{N}$. For every server $i \in \mathcal{N}$, the amount of storage space capacity is $C_i$, in Gigabytes; the amount of bandwidth capacity is $U_i$, in Mbps; and the amount of CPU computing capability is $P_i$, in MIPS.

Let the set of application instances hosted by the datacenter be denoted by $\mathcal{M}$. For any $k \in \mathcal{M}$, VM$_k$ represents the corresponding virtual machine serving that application. Operational datacenters typically provide customers the flexibility to choose from a number of different VM instances equipped with different amounts of resources in each dimension. For example, Amazon Elastic Compute Cloud (EC2) provides high-memory, micro and high-CPU instances [1], in order to serve applications with different resource demands. To better indicate the amount of dedicated resources used by each VM, let $s_k$ be the amount of storage space, $b_k$ be the amount of bandwidth, and $cl_k$ be the amount of computing capability devoted by VM$_k$. 
Table 5.2: Notations and Definitions

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_i )</td>
<td>the storage space capacity of server ( i )</td>
</tr>
<tr>
<td>( U_i )</td>
<td>the bandwidth resource capacity of server ( i )</td>
</tr>
<tr>
<td>( P_i )</td>
<td>the computing capability of server ( i )</td>
</tr>
<tr>
<td>( s_k )</td>
<td>the required storage space of VM( _k ) to handle one request</td>
</tr>
<tr>
<td>( b_k )</td>
<td>the required bandwidth resource of VM( _k ) to handle one request</td>
</tr>
<tr>
<td>( c_l_k )</td>
<td>the required computing capability of VM( _k ) to handle one request</td>
</tr>
<tr>
<td>( I^k_i(t) )</td>
<td>the binary variable indicating whether server ( i ) is holding VM( _k ) at ( t )</td>
</tr>
<tr>
<td>( D^k_i(t) )</td>
<td>the number of requests for VM( _k ) server ( i ) handles at ( t )</td>
</tr>
<tr>
<td>( \omega^s_i(t) )</td>
<td>the weight given to storage space by server ( i ) at ( t )</td>
</tr>
<tr>
<td>( \omega^b_i(t) )</td>
<td>the weight given to bandwidth by server ( i ) at ( t )</td>
</tr>
<tr>
<td>( \omega^c_i(t) )</td>
<td>the weight given to computing capability by server ( i ) at ( t )</td>
</tr>
<tr>
<td>( A^k_i(t) )</td>
<td>the anticipation of server ( i ) to VM( _k ) at ( t )</td>
</tr>
<tr>
<td>( \mathcal{F}_i(t) )</td>
<td>the utility function of server ( i ) at ( t )</td>
</tr>
<tr>
<td>( d^k_i(t) )</td>
<td>the relative distance of commodity ( k ) to player ( i ) at ( t )</td>
</tr>
<tr>
<td>( \phi^k_i(t) )</td>
<td>the utility-distance product of player ( i ) to commodity ( k ) at ( t )</td>
</tr>
</tbody>
</table>

to handle one request when serving its corresponding application.

We are aware that it is possible that some applications may span over a set of VMs, such as Map-Reduce computational jobs and multi-tier Web services. These applications may place further restrictions on the locality of their own set of VMs, since they may require a large amount of inter-VM bandwidth. To serve these applications well, a set of VMs serving the same application should be located in the same server. In this case, our model considers the set of VMs serving one application as a special VM, in a sense that they are considered as an single entity. To be exact, for an application \( k' \in \mathcal{M} \), VM\( _k' \) represents the special VM if \( k' \) spans over a set of VMs, and each VM\( _k' \) requires storage space \( s_{k'} \), bandwidth \( b_{k'} \), and computing capability \( c_{l_{k'}} \) to handle one request. Since descriptions of special VMs are in essence the same as regular VMs in our model, we do not distinguish \( k \) and \( k' \) in subsequent analysis.
Let $I^k_i(t)$ be the binary variable that indicates whether VM$_k$ is stored in server $i$ at time $t$.

$$I^k_i(t) = \begin{cases} 
1 & \text{VM}_k \text{ is stored in server } i \text{ at time } t \\
0 & \text{otherwise}
\end{cases}$$

Let $D^k_i(t)$ denote the number of requests for VM$_k$ to which server $i$ handles at time $t$.

Under the assumption that all servers in the datacenter possess global knowledge of which set of VMs other servers have and how many requests each server handles with respect to each of its VMs, at time $t$, the centralized resource utilization maximization problem in a datacenter can be formulated as a binary integer programming problem:

$$\max_{\hat{I}^k_i(t)} \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \hat{R}_i(t)$$ (5.1)

s.t.  
$$\sum_{k \in \mathcal{M}} \hat{I}^k_i(t)s_k \hat{D}^k_i(t) \leq C_i, \ \forall i$$ (5.2)

$$\sum_{k \in \mathcal{M}} \hat{I}^k_i(t)b_k \hat{D}^k_i(t) \leq U_i, \ \forall i$$ (5.3)

$$\sum_{k \in \mathcal{M}} \hat{I}^k_i(t)c_lk \hat{D}^k_i(t) \leq P_i, \ \forall i$$ (5.4)

$$\sum_{i \in \mathcal{N}} \hat{I}^k_i(t) = 1, \ \forall k,$$ (5.5)

where $\hat{R}_i(t)$ is the estimated resource utilization ratio at server $i$ after time $t$, based on the current number of requests and information about resource capacities, and $\hat{D}^k_i(t)$ is the estimated number of requests for VM$_k$ to which server $i$ handles after time $t$, which has the following form:

$$\hat{D}^k_i(t) = \sum_{i \in \mathcal{N}} D^k_i(t)I^k_i(t).$$

Note that $\forall i$ at time $t$, there is only one $I^k_i(t) = 1$, since each VM$_k$ can be possessed by only one server at one time.

To capture three integrated dimensions of resource usage at each server, we define the estimated resource utilization ratio at each server to be the weighted sum of estimated resource
utilization ratios in dimensions of storage, bandwidth and CPU:

\[
\hat{R}_i(t) = \omega_s^i(t)\hat{r}_i^s(t) + \omega_b^i(t)\hat{r}_i^b(t) + \omega_c^i(t)\hat{r}_i^c(t)
\]

\[
= \omega_s^i(t)\sum_k \hat{I}_k^i(t)\frac{s_k\hat{D}_k^i(t)}{C_i} + \omega_b^i(t)\sum_k \hat{I}_k^i(t)b_k\hat{D}_k^i(t) + \omega_c^i(t)\sum_k \hat{I}_k^i(t)cl_k\hat{D}_k^i(t)
\]

where \(\omega_s^i(t), \omega_b^i(t)\) and \(\omega_c^i(t)\) are the weights given to resources in different dimensions according to server \(i\)'s current resource usage states, constrained by \(\omega_s^i(t) + \omega_b^i(t) + \omega_c^i(t) = 1\).

In this formulation, \(\hat{I}_k^i(t)\) is the optimization variable, which is the binary indicator to denote the placement of each VM after time \(t\) based on the information at present. Inequality (5.2) stands for the storage space constraint; Inequality (5.3) represents the bandwidth capacity constraint; and Inequality (5.4) denotes the CPU computing capability constraint. The rationale behind this is that resources consumed by every server can not exceed the resource capacity in each dimension. Eq. (5.5) ensures that each VM can only be possessed by one server at a time.

In a centralized manner, the current VM placement indicator \(I_k^i(t)\), the number of requests handled \(D_k^i(t)\), and resource capacities of each server \(C_i, U_i, P_i\) are supposed to be known at time \(t\).

In our subsequent analysis, we focus on decisions at a specific time. Therefore, the time indices \(t\) in the expressions are dropped, as we obtain the following optimization problem equivalent to the original one (5.1).

\[
\max_{I_k^i} \sum_i \sum_k \left( \frac{\omega_s^i s_k\hat{D}_k^i}{C_i} + \frac{\omega_b^i b_k\hat{D}_k^i}{U_i} + \frac{\omega_c^i cl_k\hat{D}_k^i}{P_i} \right) \hat{I}_k^i \quad (5.6)
\]

We have formally described the problem of maximizing resource utilization in a datacenter as an optimization problem (5.6) in a centralized fashion. Based on existing knowledge, we wish to find out what is the best placement strategy of each VM under the constraint of resource capacities in dimensions of storage, bandwidth and CPU computing capability, so that the resource utilization is maximized.
5.2.3 Lagrangian Heuristic vs. the Nash Bargaining Solution

Our preceding formulation is a comprehensive integer optimization problem, which is in the form of a multi-dimensional Generalized Assignment Problem (GAP). The GAP is NP-hard, and even APX-hard to approximation [45]. Though we may approach the optimal solution through the Lagrangian relaxation heuristic, by decoupling it into several 0-1 knapsack problems, it incurs high computational complexity. Since our objective is to design a practical VM migration algorithm that can be implemented in real-world settings, we need to design alternative heuristics that are more feasible to be used in practice.

5.3 VM Migration Algorithm based on the Nash Bargaining Solution

We propose to use the Nash bargaining solution to solve the utilization maximization problem. The Nash bargaining game discusses the situation in which two or more players reach an agreement regarding how commodities are to be distributed among them, so that the social utility gains are maximized and commodities owned by each player do not exceed its capacity. This is exactly the same as the GAP, which aims to assign a set of objects to agents so that the total profit of the assignment is maximized and all agents do not exceed their budget. With the same objective, we believe that the mechanism of the Nash bargaining solution is a suitable alternative.

In this section, we first prove that the Nash bargaining solution in the bargaining game can be used to solve the resource utilization maximization problem in virtualized datacenters, and then present our VM migration algorithm based on the Nash bargaining solution in more detail.
5.3.1 The Nash Bargaining Solution

As we have introduced in Chapter 2, bargaining problems are known as non-zero-sum games that participating players try to achieve a win-win situation. In the Nash bargaining game, there is always a solution for the optimal strategy at each player, which guarantees that their average payoff is maximized under the assumption that opposing players also use the optimal strategy.

In Nash bargaining games, each player has a different anticipation to each commodity, which represents a state of expectation that may involve the certainty of some contingencies and various probabilities of other contingencies [57]. For example, if Bill prefers apple to banana, then he may have a higher anticipation of apple than that of banana. The utility of each player is a function of his anticipations to commodities he has. The Nash bargaining solution is a Pareto efficient solution to a Nash bargaining game so that the joint profit, which is the product of utility gains of all players, is maximized.

Let \( F_i \) be the utility function for player \( i \). Rational players will seek to maximize the Nash product \( \prod G_i \), where \( G_i = |F_i(x) - F_i(d)| \). \( F_i(d) \) is the status quo utilities (i.e., the utility obtained if one decides not to bargain with other players). Nash has shown that obtaining the maximum of the Nash product will attain the Pareto-optimal solution for the bargaining situation.

**Theorem 1** The problem of maximizing resource utilization in a virtualized datacenter is equivalent to the joint profit maximization problem in the Nash bargaining game.

**Proof** Envision a market, in which servers are treated as players and VMs are considered as commodities. The Nash bargaining game here is to exchange VMs among the set of servers so that their joint profit is maximized. Let \( A^k_i \) be player \( i \)'s anticipation to commodity \( k \). Define the utility function of player \( i \) to be \( F_i \), which is represented as follows:

\[
F_i(x) = F_i(d) + \exp\left(\sum_{k \in M} A^k_i (\hat{I}_i^k - I^k_i)\right).
\]

The definition can be explained as the utility of player \( i \) after bargaining equals its status quo utility plus a function of added anticipations during bargaining. That is to say, the utility
gain of player $i$ can be represented as:

$$G_i = |\mathcal{F}_i(x) - \mathcal{F}_i(d)| = \exp\left(\sum_{k \in M} A_i^k (\hat{i}_i^k - I_i^k)\right). \quad (5.7)$$

In practice, the Nash bargaining solution aims to maximize the Nash Product $\prod G_i$, which can be interpreted as follows:

$$\max \prod_{i \in N} G_i \quad (5.8)$$

subject to

$$\sum_{k \in M} \hat{i}_i^k s_k \hat{D}_i^k \leq C_i, \forall i \quad (5.9)$$
$$\sum_{k \in M} \hat{i}_i^k b_k \hat{D}_i^k \leq U_i, \forall i \quad (5.10)$$
$$\sum_{k \in M} \hat{i}_i^k c_k \hat{D}_i^k \leq P_i, \forall i \quad (5.11)$$
$$\sum_{i \in N} \hat{i}_i^k = 1, \forall k. \quad (5.12)$$

Constraints (5.9), (5.10) and (5.11) represent the fact that commodities each player owns cannot exceed its capacity. Eq. (5.12) confirms that each commodity can only be possessed by one player at a time.

Substitute (5.7) into (5.8), the maximization problem in the Nash bargaining game is equivalent to the following ones.

$$\max \prod_{i \in N} G_i \iff \max \exp \log \prod_{i \in N} G_i$$
$$\iff \max \exp \sum_{i \in N} \log G_i$$
$$\iff \max \sum_{i \in N} \sum_{k \in M} A_i^k \hat{i}_i^k.$$

If we define player $i$’s anticipation to commodity $k$, i.e., $A_i^k$ in the following form:

$$A_i^k = \frac{\omega_i^s s_k \hat{D}_i^k}{C_i} + \frac{\omega_i^b b_k \hat{D}_i^k}{U_i} + \frac{\omega_i^c c_k \hat{D}_i^k}{P_i},$$

the optimization problem (5.8) in the Nash bargaining game can be rewritten as:

$$\max \sum_{i \in N} \sum_{k \in M} \left(\frac{\omega_i^s s_k \hat{D}_i^k}{C_i} + \frac{\omega_i^b b_k \hat{D}_i^k}{U_i} + \frac{\omega_i^c c_k \hat{D}_i^k}{P_i}\right) \hat{i}_i^k.$$
which is equivalent to the resource utilization maximization problem (5.6) in virtualized datacenters. It then follows that the joint profit maximization problem in the Nash bargaining solution is equivalent to the resource utilization maximization problem in virtualized datacenters.

Based on the established connection between the two problems, the optimization variable \( \hat{I}_k^i \) can be viewed as an indicator of the strategy adopted by each player in the bargaining game. The estimated resource utilization ratio of server \( i \), \( \hat{R}_i \), can be treated as the profit gains of player \( i \) by adopting strategy \( \hat{I}_k^i \). This implies that the resource utilization maximization problem can be solved using the mechanism of Nash bargaining solution.

5.3.2 VM Migration in the Bargaining Game

**VM-based market organization.** We envision the existence of an online market place, where all servers in a datacenter behave as players; application VMs are treated as commodities. Every VM is associated with an anticipation \( A_k^i \) from each player, which is the evaluation of VM \( k \) from server \( i \)'s perspective based on its own information. According to the proof of Theorem 1, \( A_k^i \) should be of the following form:

\[
A_k^i = \omega^s_i \frac{s_k D_i^k}{C_i} + \omega^b_i \frac{b_k D_i^k}{U_i} + \omega^c_i \frac{c_l D_i^k}{P_i}. \tag{5.13}
\]

In our bargaining market, the anticipation of player \( i \) to each commodity is defined as the consumed fraction of resources in the corresponding application VM in server \( i \), with respect to dimensions of storage, bandwidth and CPU computing resources. Since our objective is to fully utilize resources in every server, application VMs requiring more resources will be more valuable. However, fractions in different dimensions should be treated differently, which are weighted by \( \omega^s_i \), \( \omega^b_i \) and \( \omega^c_i \) according to Player \( i \)'s current resource usage states. The rationale is that resources with a high utilization ratio will become the “bottleneck” towards fully utilizing resources in other dimensions, hence should be less desirable by that server. For simplicity, we
define the weights to be inversely proportional to the current resource utilization ratio in the corresponding dimension. That is,

\[
\omega^a_i = \frac{\frac{1}{r^a_i}}{\sum_{i} \frac{1}{r^a_i}}, \quad \omega^b_i = \frac{\frac{1}{r^b_i}}{\sum_{i} \frac{1}{r^b_i}}, \quad \omega^c_i = \frac{\frac{1}{r^c_i}}{\sum_{i} \frac{1}{r^c_i}},
\]

where \(\sum \frac{1}{r} = \frac{1}{r^a_i} + \frac{1}{r^b_i} + \frac{1}{r^c_i}\). Recall that \(r^a_i, r^b_i\) and \(r^c_i\) are defined as the current resource utilization ratio of server \(i\) in the dimension of storage, bandwidth and CPU computing, respectively, \(i.e.\),

\[
\begin{align*}
    r^a_i &= \frac{\sum_{k \in \mathcal{M}} I^{k} s_k D^k_i}{C_i} \\
    r^b_i &= \frac{\sum_{k \in \mathcal{M}} I^{k} b_k D^k_i}{U_i} \\
    r^c_i &= \frac{\sum_{k \in \mathcal{M}} I^{k} c l_k D^k_i}{P_i}.
\end{align*}
\]

**The bargaining strategy based on spacial representation.** It is proved that the problem of determining whether there exists a Nash equilibrium in which each player has a specific minimum payoff is NP-complete as a function of the number of players [17], so that the Nash bargaining solution in a bargaining game is usually approximated by numerical methods such as the Newton’s method or the bisection method [19], which require numerous iterations. Since our objective is a practical VM migration algorithm that can be implemented in real-world datacenters and executed in a lightweight fashion, we propose to adopt a bargaining strategy based on the spacial representation of Nash bargaining games [79].

This bargaining strategy based on the spacial representation fits our design objective in virtualized datacenters, since it reduces the computational overhead significantly by eliminating the requirement of generating the utility gain for every single possible set of strategy execution. By introducing the **utility-distance product** \(\phi^{k}_i\), a function of the anticipation \(A^{k}_i\), it is proved that the utility-distance product of a commodity is analogous to the moment of force by weights based on a lever system. By suitably locating a pivot location such that the distribution of the utility-distance product is uniformly positioned about a pivot, equilibrium can be achieved.

From a spacial perspective, outcomes of games have been assumed to lie in some low-dimensional Euclidean space, such that anticipations to the players are defined in terms of
distances from them [59]. In such spacial games, anticipation of a player to a commodity is assumed to be an inverse function of the distance that commodity lies from the player [17], such that commodities of higher anticipation values have a closer spatial proximity (i.e., a shorter spatial distance).

![Spacial representation of a 2-player game.](image)

Figure 5.4: Spacial representation of a 2-player game.

For example, Fig. 5.4 shows the 2-dimensional spatial representation of a 2-player game with player-to-player distances defined to be constants. Anticipations of the two players to commodities 1 and 2 are clearly reflected by spacial distances $d^1_1$ and $d^2_2$, where $i \in \{1, 2\}$. In the example, commodities are represented as points based on their spatial proximities, which lie within the boundary enclosed by all participating players. The relative distances of a commodity $k$ to player $i$ is defined as $d^k_i$, which is in the form of:

$$d^k_i = f(A^k_i) = \sum_i \frac{1/A^k_i}{\sum_i (1/A^k_i)} \forall i \in \mathcal{N}, k \in \mathcal{M}. \quad (5.14)$$

The distances of each commodity to all players are normalized so that they add up to a unitary value, i.e., $\sum_i d^k_i = 1, \forall k$. In our example, player 1 has higher anticipation to commodity 1 and player 2 prefers to commodity 2.

In this bargaining strategy, each player possesses commodities sorted by their relative distance $d^k_i$, such that commodities with higher anticipations will be given higher priority. The utility-distance product of a player to a commodity is defined as $\phi^k_i = d^k_i \cdot A^k_i$. Fig. 5.5 shows how commodities are sorted by player $i$ and the associated utility-distance product $\phi^k_i$ of each commodity.

From a mechanical perspective, weights on a lever are aligned along the same direction such that weights on the left hand side generate a collective moment that opposes the moment caused by weights on the right. Similarly, to maintain an equilibrium in a bargaining game,
the sum of utility-distance products of all commodities should be equally divided among all participating players, as shown in Fig. 5.6. Players 1 and 2 are considered to be lying at the end points of the lever, where forces can be applied. The utility-distance products of commodities are regarded as weights.

It is then not difficult to find out that the pivot point of the lever in our example should be lying between commodities 2 and 8, where the collective moment generated by weights $\phi_1^3 + \phi_1^4 + \phi_1^5$ on Player 1’s side equals to that of $\phi_2^1 + \phi_2^5 + \phi_2^8$ on Player 2’s side. To be precise, the pivot point in a lever system is determined by balancing weights between two end points,
which is:

$$\mu = \frac{1}{2} \sum_{k \in M} \phi_i^k.$$ 

After determining the pivot point, it is natural that the bargaining solution is to assign commodities lying on the left hand side of the pivot point to player 1, and commodities on the right hand side to player 2.

This bargaining strategy can be easily generalized to multi-player bargaining games. For the ideal condition whereby all commodities lie in a space between vertices representing all players, the determination of a pivot location can be based on balancing the utility-distance product with respect to all players, which is given by:

$$\mu_i = \frac{1}{|N|} \sum_{k \in M} \phi_i^k, \forall i \in N.$$  \hspace{1cm} (5.15)

Fig. 5.7 shows an example of a 3-player game. The vertical arrows on each commodity represents the cumulative utility-distance product $p_i^k$, which is defined as the sum of utility-distance products of commodities whose relative distances are not greater than that of $k$, i.e.,

$$p_i^k = \sum_{\forall j \in M, d_i^j \leq d_i^k} \phi_i^j.$$ 

By finding the pivot point where $p_1^{k_1} = p_2^{k_2} = p_3^{k_3}$, the bargaining solution to the game is found, which is, in our example, $C_1 = \{1, 3, 5, 7\}$, $C_2 = \{2, 8, 9, 12\}$, and $C_3 = \{4, 6, 10, 11\}$.

After each player determines its own pivot point $\mu_i$, commodities inside the market will be assigned to each player accordingly. To be precise, each player will be assigned a set of commodities $C_i$, so that the sum of utility-distance products of $C_i$ is maximized but not larger than $\mu_i$. If one commodity $k$ is assigned to more than one players at one time, the player who has the smallest relative distance $d_i^k$ will obtain this commodity. To conclude, our VM migration algorithm based on the Nash bargaining solution is summarized in Algorithm 2.

Practicality. Though migrating VMs have potential benefits to improve the resource utilization ratio, it does not come without substantial upfront costs of bandwidth. An example orchestration of live VM and storage migration on the testbed through HARMONY shows that the transaction throughput is reduced by 12% during VM migration [70]. Since the application
Figure 5.7: Finding the pivot point in a 3-player game according to the utility-distance products.

As a consequence, our VM migration algorithm is triggered in a *laissez-faire* manner. Whenever the resources provided by one server can not sustain requests for applications placed on that server, the migration algorithm is triggered. The idea is, if requests can be satisfied under the current provision, we’ll maintain the same even if the resource utilization is not the optimal, *e.g.*, when the number of requests for one application decreases dramatically at some time. Note that it might be the case that even after VM migration, the application fit in the datacenter still can not handle all requests at the same time, *i.e.*, the total available resources is less than the sum of required resources for all requests. In this scenario, the only solution for the datacenter is to add new servers. To avoid triggering the VM migration algorithm constantly in this scenario, we restrict the minimum interval between two trigger points to be $T$, where $T = 20$ min in our simulation.
Algorithm 2 The VM Migration Algorithm.

1: Each player $i$ computes its anticipation to each commodity $k$ in the market, i.e., $A_i^k$ given by (5.13). Then, it derives the relative distance $d_i^k$ using (5.14), and the corresponding utility-distance product $\phi_i^k$, given by $\phi_i^k = d_i^k \cdot A_i^k$.

2: Each player $i$ computes the cumulative utility-distance product $p_i^k$ for all commodities, according to their relative distances $d_i^k$.

3: Each player $i$ determines his pivot point $\mu_i$ according to (5.15).

4: Each player $i$ finds a subset of commodities $C_i$, whose cumulative utility-distance products are not larger than $\mu_i$, i.e., $C_i = \{k : p_i^k \leq \mu_i, \forall k \in \mathcal{M}\}$.

5: if Commodity $k$ belongs to more than one $C_i, \forall i \in \mathcal{N}$ then

6: The player $i' (k \in C_{i'})$ who has the smallest $d_i^k$ to commodity $k$ will obtain this commodity.

7: else

8: The player $i' (k \in C_{i'})$ will obtain this commodity.

9: end if

10: All players migrate their commodities according to the obtained assignment results.

5.4 Experimental Evaluation

In this section, we investigate how the proposed VM migration algorithm performs in practical scenarios. Our real-world trace-driven experimental results validate that our VM migration algorithm increases resource utilization ratios with fluctuating requests in video streaming datacenters effectively, yet in a lightweight fashion.

Our evaluation of the proposed VM migration algorithm is based on a C++ implementation in an event-driven simulator, which is driven by real-world streaming requests captured by our traces. We are using 200 Gigabytes worth of operational traces, which we have collected throughout the 17-day Summer Olympic Games in August 2008, with UUSee as one of the official online broadcasting partner in China. Both VoD and live streaming videos were involved,
with each of them represented by a VM in our simulation. More detailed information of our trace used in this simulation is summarized in Fig. 5.3.

Table 5.3: Detailed Information About the Trace

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of videos</td>
<td>1625</td>
</tr>
<tr>
<td>Number of live streaming videos</td>
<td>216</td>
</tr>
<tr>
<td>Number of VoD videos</td>
<td>1409</td>
</tr>
<tr>
<td>Number of videos with network coding</td>
<td>1472</td>
</tr>
<tr>
<td>Peak number of concurrent requests (for all videos)</td>
<td>24757</td>
</tr>
<tr>
<td>Highest video bitrate</td>
<td>879 kbps</td>
</tr>
<tr>
<td>Lowest video bitrate</td>
<td>264 kbps</td>
</tr>
</tbody>
</table>

As we can see from the table, videos in the trace vary in terms of their bitrates, which results in different bandwidth resource demands in each request. In our simulation, we assume each user asks for 10 seconds of video with every request, \( i.e. \), the bandwidth resource required per request ranges from 264 KB to 879 KB. Besides, videos also differ from whether network coding is applied during transmission, which results in deviated CPU cycles required in each request. For videos without using network coding, the required CPU cycles per request is assumed to follow a normal distribution of \( N(2, 0.25) \) MIPS; for those with network coding, the required CPU cycles per request is assumed to be represented by \( N(2, 0.25) + \text{bitrate} / 100 \times N(1, 0.25) \), since it is measured to be proportional to the normalized bitrate of each video. We simulate a system with 25 servers, each of which is assumed to have the same amount of resources: 1000 GB storage space, 1000 Mbps bandwidth and 1000 MIPS CPU cycles.

Our objective is to increase resource utilization ratios in video streaming datacenters, so that they can satisfy as many requests as possible with their available resources. We compare the bargaining-based VM migration algorithm with the naive VM migration algorithm: only the VM that causes overload is migrated; and its destination server is greedily selected from all under-utilized servers, \( i.e. \), the one with the most available resources. Main performance metrics in this simulation are, therefore, the improvement on resource utilization ratios and
the number of requests the datacenter can handle successfully. In addition, we also show the bargaining overhead from an implementation point of view. We run our simulation 20 times, each lasting 100 time intervals.

We first present the improvement on resource utilization ratios by using the bargaining-based VM migration algorithm, as a percentage of the ratios using the naive algorithm. Fig. 5.8 shows the demand variation in 200 hours. Fig. 5.9 shows improvements on average resource utilization ratios at all servers over time and their 95% confidence intervals under the demand pattern shown in Fig. 5.8. As we can observe, the average improvement on resource utilization ratios varies with the same trend as demand patterns, with a maximum improvement of almost 20% than that of the naive VM migration algorithm.

The reason that the improvement on resource utilization is less evident and the number of requests is low is that most servers are under-utilized in this scenario. The optimal solution that maximizes the overall resource utilization by the bargaining-based VM migration algorithm is more likely to conform with the greedy results. However, when the number of requests increases, improvements on resource utilization with the bargaining-based algorithm become more evident. Fig. 5.10 shows the average of improvements on resource utilization ratios vs. the number of requests. It is clear that the improvement is significant when the number of
requests becomes large.

Figure 5.11: Improvement on the number of requests successfully handled by the datacenter.

Figure 5.12: Improvement on the number of request successfully handled vs. the total number of requests.

In addition to resource utilization ratios, another important performance metric is the number of requests that the datacenter is able to handle, given current available resources. As shown in Fig. 5.11, it is clear that the number of requests the datacenter can respond to has increased accordingly by applying the bargaining-based VM migration algorithm, with an improvement of up to 9% compared with the naive algorithm. With more than 20000 concurrent requests at the peak time, this implies an increase of more than 1800 requests being handled by the datacenter. This result confirms what we observed in the improvement on resource utilization ratios, since the more efficient available resources are being utilized, the more requests the datacenter can handle. Fig. 5.12 shows the improvement on the number of requests successfully handled vs. the total number of requests. We can observe a similar trend as Fig. 5.10, that the improvement becomes more evident as the number of requests increases.

To further investigate the effectiveness of our algorithm, in Fig. 5.13, we show reductions in the standard deviation of resource utilization ratios at each server, as a percentage across 500 bargaining samples. A reduced standard deviation of resource utilization ratios reflects a more balanced resource utilization. Fig. 5.14 shows the CDF of reductions on standard deviations.
We can observe that the standard deviation of resource utilization ratios is successfully reduced, the 90th percentile of the ratio of such reduction is 0.45%, which shows that our VM migration algorithm is able to mitigate resource under-utilization due to imbalanced resource usage across different dimensions.

Finally, it is important to evaluate the overhead incurred by VM migration with our bargaining-based solution. We plot the number of trades conducted after running the bargaining algorithm to balance the resource utilization across all servers in 1000 bargaining samples in Fig. 5.15,
and the CDF of all samples in Fig. 5.16. As we can see, in 90% of the cases, the algorithm incurs fewer than 5 trades. It reveals that the VM migration overhead is reasonably low.

5.5 Summary

From the cloud provider’s point of view, we first realize that by providing their resources in the form of virtual machines, it is certainly the cloud providers’ wish to fully utilize their available resources. Our focus in this chapter is to fully utilize resources in dimensions of storage, bandwidth and CPU computing cycles in video streaming datacenters, by migrating VMs live among servers when they are overloaded. We argue that such migration should be conducted with careful planning, in order to fully explore possibilities of utilizing current available resources, in terms of storage, bandwidth and CPU. From time to time, conducted in a laissez-faire manner, we believe that VM migration is helpful to improve overall resource utilization, and ultimately to deliver better performance as more video requests are being handled. As a practical way to govern these VM migration decisions, we have designed an algorithm based on the Nash bargaining solution. With event-driven simulations based on real-world video streaming traces from UUSee Inc., we show that the bargaining algorithm is able to improve resource utilization over time, with a small amount of VM migration overhead.
Chapter 6

Jetway: Minimizing Costs on Inter-Datacenter Traffic

6.1 Overview

As we have already pointed out in Chapter 5, due to abundant resource availability and reduced management costs in the cloud, it is an emerging trend for video streaming services to migrate to cloud services, such as Amazon Web Services (AWS). As an example, Netflix is using the Amazon Simple Storage Service (S3) for storing all of its video masters, which are further transcoded to a number of video formats, and are then distributed to Content Distribution Networks (CDNs), ready to be served to end users [7, 25].

On the other hand, Amazon Web Services, as cloud providers, have recently introduced CloudFront, a CDN service with edge servers around the world, designed to meet the needs of video streaming services. CloudFront seamlessly integrates with Amazon S3 and the Amazon Elastic Compute Cloud (EC2), so that videos hosted in S3 or EC2 can be streamed using the Real Time Messaging Protocol (RTMP) to end users, from one of the edge servers with geographical proximity and low network latencies [12]. In addition, in order to accommodate high-definition videos, Amazon S3 has substantially raised its limit on object sizes (from 5 GB...
to 5 TB) in December 2010 [11]. Given such a win-win situation as video streaming providers are going “100% cloud” [25], it will be a near-term certainty that large volumes of video traffic will flow from cloud datacenters (e.g., S3), which host video masters, to CDN edge servers (e.g., CloudFront), which serve end users.

The tenet of providing cloud services is to maximize the sharing of resources, while keeping tenants (e.g., Netflix) satisfied. To provide cloud services with better availability and scalability, it is customary for cloud providers to deploy a number of datacenters across different geographical regions. These datacenters are typically inter-connected with high-capacity links leased from ISPs. With substantial upfront investments to construct these datacenters, it is certainly to the advantage of cloud providers to minimize operational costs.

Greenberg et al. [41] have revealed that traffic costs amount to around 15% of operational costs incurred to a cloud provider. In particular, Chen et al. [23] have pointed out that inter-datacenter traffic accounts for up to 45% of the total traffic going through datacenter egress routers. As large quantities of high-definition videos are being hosted in these datacenters, a substantial amount of inter-datacenter traffic will be incurred by replicating these videos and by serving these videos to CDN edge servers, in order to provide a highly available and scalable streaming service. Such inter-datacenter video traffic constitutes a large portion of a cloud provider’s inter-datacenter traffic. Since most cloud providers today rely on multiple Internet Service Providers (ISPs) to connect their geographically dispersed datacenters [82], operational costs can be effectively reduced, if costs charged by these ISPs on inter-datacenter video traffic can be minimized.

Given dominant percentile-based charging models currently in use by most ISPs [66], e.g., the 95-th percentile charging model, it is feasible to reduce or even minimize cloud providers’ costs by designing optimal routing and flow assignment algorithms for inter-datacenter video traffic. In other words, video flows across inter-datacenter links can be — and should be — split and transmitted along multiple multi-hop paths, each of which can be optimally and dynamically computed over time. The rationale is that, the cost of transmitting the same amount
of videos varies significantly across different inter-datacenter links, due to regional pricing and peering relationships among ISPs [74]. For example, domestic video flows are substantially cheaper than flows to global destinations, and video flows within the backbone network built by cloud providers themselves incur very low costs.

Further, spatial and temporal characteristics of inter-datacenter video traffic also motivates the design of new routing and flow assignment algorithms. Temporally, a portion of the video flows across datacenters may be more delay-tolerant than others, if they represent video replication and backups. To reduce costs, we may re-route these delay-tolerant video flows by using intermediate datacenters as relays, flowing over multi-hop paths and splitting into multiple paths [49]. Spatially, datacenters located in different time zones experience peak video traffic at different times, providing more opportunities of resource multiplexing.

In this chapter, we present Jetway, a new set of algorithms designed to minimize operational costs on inter-datacenter video traffic in an efficient and simple way. To guide the design of our algorithms in Jetway, we present a methodical and in-depth analytical study on how inter-datacenter video traffic costs are to be minimized by routing video flows via multiple multi-hop paths in an optimized fashion. With Jetway, we take advantage of different traffic costs on inter-datacenter links, usually charged by a multitude of ISPs with the percentile-based charging model, taking into account practical constraints of limited link capacities, as well as different desired transmission rates of videos, representing their delay tolerance. Our study leads to new combinatorial algorithms that are simple yet efficient enough for cloud providers to implement in practice: video flows are split and routed in an on-the-fly fashion by solving the classic minimum-cost multicommodity flow and maximum concurrent flow problems. An illustrative example of routing inter-datacenter video flows is shown in Fig. 6.1, in which the width of each flow denotes the flow rate on a particular link.

We evaluate the performance of Jetway in minimizing costs on inter-datacenter video traffic with our real-world implementation in the Amazon EC2 cloud, as well as extensive simulations. Our Jetway implementation has been developed based on a flexible and reusable video stream-
CHAPTER 6. JETWAY: MINIMIZING COSTS ON INTER-DATACENTER TRAFFIC

Figure 6.1: Selecting the best paths for inter-datacenter video traffic in a cloud with 5 datacenters. Based on differing traffic costs on inter-datacenter links and varying transmission rates of videos, the video from Datacenter 2 (D2) to Datacenter 5 (D5) is best routed along path \( \{D_2 \rightarrow D_1 \rightarrow D_5\} \), \( \{D_2 \rightarrow D_3 \rightarrow D_5\} \), and \( \{D_2 \rightarrow D_4 \rightarrow D_5\} \), represented by red (light gray) flows; the video from Datacenter 3 (D3) to Datacenter 1 (D1) is best routed along path \( \{D_3 \rightarrow D_2 \rightarrow D_1\} \) and \( \{D_3 \rightarrow D_1\} \), represented by blue (dark gray) flows.

Before we formulate our problem in a more rigorous fashion, we first present the rationale and challenges that motivate the design of Jetway, with an objective of minimizing operational costs on inter-datacenter video traffic.
6.2 Rationale and Challenges

A cloud provider is usually charged by ISPs for its inter-datacenter traffic. The operational costs incurred are typically based on the amount of traffic the cloud provider generates. The percentile-based charging model, which is also called the \( q \)-th percentile charging model, is predominantly used by ISPs today. With this charging model, an ISP records the traffic volume a cloud provider generates during each 5-minute interval and sort them in a descending order. At the end of a complete charging period, the \( q \)-th percentile of all 5-minute traffic volumes is considered as the charging volume \( x \), which will be used to derive the cost by a piece-wise linear non-decreasing function \( c(x) \) [40]. For example, if the 95-th percentile charging model is in use and the charging period is one year, then the charging volume \( x \) of the cost function corresponds to the traffic volume sent during the 99864-th sorted interval \( (95\% \times 365 \times 24 \times 60/5 = 99864) \).

Besides the fact that traffic costs on each inter-datacenter link differ from one another, in that a relay path might incur much lower costs than a direct path, the percentile-based charging model provides further opportunities to reduce operational costs. With a percentile-based charging model, if some video traffic is already generated on one link, idling or transmitting less video traffic in subsequent time intervals within the same charging period will be a waste of capital investment, as these time intervals will be charged based on the already generated traffic volume anyway. As such, a feasible way to reduce costs is to carefully design routing paths and flow assignments for each pair of inter-datacenter video flow — a key idea in the design of Jetway — such that the idling and under-utilized time intervals are eliminated as much as possible, and the \( q \)-th percentile of video volumes over all time intervals is minimized.

The following example intuitively explains the rationale of optimal routing with the percentile-based charging model. Shown in Fig. 6.2, 3 videos are to be transferred in an inter-datacenter network with 4 datacenters. Assume that Video 1 and 2 are to be streamed from datacenter \( D_2 \) to \( D_4 \) with rate 8 and from \( D_3 \) to \( D_1 \) with rate 6 within the first time interval, respectively; and Video 3 is to be streamed from \( D_3 \) to \( D_4 \) with rate 5 in the second time interval. For the sake
of simplicity, we assume that the 100-th percentile charging model with a linear cost function is in use in this example, which implies that the cost incurred between each datacenter pair is the maximum video volume sent during these time intervals, multiplied by a flat cost per traffic unit shown on the link. The link capacity are assumed to be 5 for all the links.

![Fig. 6.2: How traffic costs can be reduced with optimal routing: a motivating example.](image)

If the difference of costs per unit of traffic on each inter-datacenter link is considered, cheaper paths are preferred by video flows to reduce traffic costs. Shown in Fig. 6.2 (a), the optimal routing and flow assignment in this scenario is: Video 1 will take paths D\(_2\) → D\(_1\) → D\(_4\) and D\(_2\) → D\(_3\) → D\(_4\), each with a flow of 4, and Video 2 will take paths D\(_3\) → D\(_2\) → D\(_1\) and D\(_3\) → D\(_1\) with flow 1 and 5, respectively in the first time interval; and Video 3 will take the direct path at the second time interval, leading to a total cost of 104 per time interval.

If we further incorporate the consideration of the 100-percentile charging model in this example, we can see that a part of Video 3 can be routed along the more expensive path D\(_3\) → D\(_1\) → D\(_4\), taking advantage of the already generated traffic volume on this path in past time intervals, when carrying the flows of Video 1 and 2. Shown in Fig. 6.2 (b), the optimal routing and flow assignment in this scenario is to route Video 3 through path D\(_3\) → D\(_1\) → D\(_4\) and D\(_3\) → D\(_4\) in the second time interval, with rates 1 and 4, respectively. By doing so, costs on inter-datacenter video traffic per time interval can be reduced to 96, as Video 3 is carried for free with the percentile-based charging model.
Unfortunately, applying the basic concept of multi-hop routing in each video flow presents formidable challenges when it comes to more general cases, involving multiple video flows with different source-destination datacenters. Due to the consideration of the percentile-based charging model, the dimension of time has to be taken into account when computing incurred costs on a link, which increases the complexity of the problem significantly. The cost of inter-datacenter video traffic in one time interval is affected by the traffic volume in time intervals before and after that time interval within the same charging period. If we wish to optimize the cost globally, we will need to estimate future traffic demand within the entire charging period (say, a month or a year), yet inter-datacenter traffic may not be accurately predictable beyond much finer time scales (such as a few seconds) [18]. In order to design algorithms to minimize costs incurred by inter-datacenter video traffic, it is our objective to formulate the problem such that it is practically solvable, yet sufficiently efficient.

### 6.3 System Model

Before formulating the problem of minimizing operational costs on inter-datacenter traffic formally, we need to first introduce our network model. Important notations used throughout this chapter are listed in Table 6.1.

We consider a cloud with multiple geographically distributed datacenters operated by a single cloud provider. Every datacenter in the cloud is connected to all other datacenters. We use a complete directed graph \( G = (V, E) \) to represent the inter-datacenter network, where \( V \) indicates the set of datacenters, and \( E \) indicates the set of directed links inter-connecting datacenters. For each link \( \{i, j\} \in E \), we use a non-negative real-value function \( a_{ij} \) to denote the cost per traffic unit from datacenter \( i \) to \( j \); a positive function \( c_{ij}(t) \) to denote the available link capacity at time \( t \), which is the maximum available rate of transmission from datacenter \( i \) to \( j \).

Let \( K(t) \) be the set of videos to be transmitted at time \( t \), all of which are represented by
source-destination video flows. During its transmission to the destination datacenter, each flow can be split and relayed by other datacenters, e.g., it can be routed over multiple paths and multiple hops within a path. For each video flow $k \in K(t)$, we use a specification $(s_k, d_k, r_k)$ to describe it. Here, vertices $s_k$ and $d_k$ indicate the source and the destination datacenter from and to which flow $k$ is being routed, and $r_k$ is the desired transmission rate for video flow $k$. In the interest of minimizing traffic costs incurred on inter-datacenter links, the desired transmission rate for each video flow that is being replicated to other datacenters is obtained by its size divided by its corresponding maximum tolerable transfer time; and is the minimum rate for an enjoyable video playback for transit server-to-customer video flows.

Since the time dimension has to be considered in the percentile-based charging model, we use a time-slotted model to incorporate multiple time intervals in a charging period. Let $I$
be the number of time intervals in a charging period, with \( t \) as indices. The duration of one time interval is assumed to be 5 minutes, which is denoted by \( \bar{t} \). To focus on the essence of the problem, we assume that the 100-th percentile charging model is in use, \( i.e. \), a cloud provider is charged based on the maximum of traffic volumes generated over all time intervals in a charging period. Our results can be extended to a cloud environment with any percentile-based charging model, which will be a topic of discussion forthcoming in this chapter.

For simplicity, we assume that the cost function \( c(x) \) is a linear function \( c(x) = a \cdot x \), where \( x \) is the traffic volume to be charged. To be exact, if we use \( f_{ij}(t) \) to denote the aggregate flow rate on link \( \{i,j\} \) in the time interval \( t \), cost on inter-datacenter traffic in one charging period is the dot product:

\[
\text{cost} = \sum_{\{i,j\} \in E} a_{ij} (\max_t f_{ij}) \bar{t} I,
\]

where \( \max_t f_{ij} \) is the maximum aggregate flow rate on link \( \{i,j\} \) of all \( f_{ij}(t) \), from \( 1 \leq t \leq I \). Note that the aggregate flow rate \( f_{ij}(t) \) may also contain other inter-datacenter traffic such as backups and propagation of large updates. If we use \( d_{ij}(t) \) to denote the rate of this part of flow on link \( \{i,j\} \) in the time interval \( t \), then the aggregate flow rate on link \( \{i,j\} \) can be represented by

\[
f_{ij}(t) = d_{ij}(t) + \sum_{k \in K(t)} f_{ij}^{k}(t),
\]

where \( f_{ij}^{k}(t) \) indicates the amount of flow assigned on link \( \{i,j\} \) for video flow \( k \) in the time interval \( t \).

To capture the fact that each video flow \( k \) may be initiated at any time in a charging period, and will also be terminated after a period of time, we let \( r_k(t) \), the desired transmission rate of video flow \( k \), depend on the time index \( t \), indicating whether or not flow \( k \) is in transmission. More precisely:

\[
r_k(t) = \begin{cases} 
  r_k & t \in \text{time intervals video flow } k \text{ is in transmission,} \\
  0 & \text{otherwise.}
\end{cases}
\]

To simplify the model, we assume that video flows are always initiated at the beginning of a
time interval, and terminated at the end of a time interval.

### 6.4 Problem Formulation

In the design of Jetway, the problem we are trying to solve is: what is the optimal routing and flow assignment strategy we can apply to inter-datacenter video flows initiated in the current time interval to minimize operational costs, with the assumption that past (historical) information is known? In other words, we seek to find optimal routing paths and flow assignments for each video flow initiated in the current time interval \( t \), so that the operational costs to the cloud provider are minimized till the end of time interval \( t \), given all routing paths and flow assignments for video flows initiated from time interval 1 to \( t - 1 \).

If we use \( \text{cost}_{ij}(t) \) to denote the operational costs on link \( \{i, j\} \) \textit{up to} time interval \( t \), the optimal routing and flow assignment problem in Jetway can be formulated as the following optimization problem, with the assumption that all datacenters in the cloud possess information about all video flows:

\[
\begin{align*}
\min_{f^k_{ij}(t)} & \sum_{\{i,j\} \in E} \text{cost}_{ij}(t) \\
\text{s.t.} & \sum_{k \in K(t)} f^k_{ij}(t) + d_{ij}(t) \leq c_{ij}(t), \forall \{i, j\} \in E \\
& \sum_{j \in V} f^k_{ij}(t) = \sum_{j \in V} f^k_{ji}(t), \forall k \in K(t), \forall i \in V / \{s_k, d_k\} \\
& \sum_{j \in V} f^k_{s_kj}(t) - \sum_{j \in V} f^k_{js_k}(t) = r_k(t), \forall k \in K(t) \\
& f^k_{ij}(t) \geq 0, \forall k \in K(t), \forall \{i, j\} \in E,
\end{align*}
\]  

(6.1)

where \( \text{cost}_{ij}(t) \) equals the product of the maximum aggregate flow rate on each link over \( t \) time intervals and the duration of a time interval with the 100-th percentile charging model, \textit{i.e.},

\[
\text{cost}_{ij}(t) = a_{ij}(\max_{t} f_{ij})\bar{t}I.
\]

(6.5)

The optimization variable of problem (6.1) is \( f^k_{ij}(t) \), which indicates the flow assigned on link \( \{i, j\} \) for video flow \( k \) in the time interval \( t \). The assumption is that flows assigned on ev-
every link for each video flow in the past time intervals, and current flow rates incurred by other applications are known a priori, i.e., \( f_{ij}^k(t) \) up to \( f_{ij}^k(t-1) \) and \( d_{ij}(t) \) are known. Inequality (6.2) stands for the link capacity constraint, which ensures that the total flow assigned on one link will not exceed its current capacity. Inequalities (6.3) represent the flow conservation constraint. For each video flow, the flows going into any intermediate datacenter should equal to flows going out of that datacenter; while the flows coming from the source datacenter should be exactly the same as its desired transmission rate. Inequality (6.4) ensures that all flows are non-negative.

With the 100-th percentile charging model, the cost function (6.5) can be rewritten as:

\[
\text{cost}_{ij}(t) = \begin{cases} 
\text{cost}_{ij}(t-1) & f_{ij}(t) \leq \max_{t-1} f_{ij} \\
a_{ij} f_{ij}(t) \bar{t}I & \text{otherwise}
\end{cases}
\]

Based on historical information, \( \max_{t-1} f_{ij} \bar{t} \), the charging volume up to time interval \( t-1 \), is known. The cost function on link \( \{i, j\} \) up to time interval \( t \) is equivalent to:

\[
\text{cost}_{ij}(t) = \text{cost}_{ij}(t-1) + a_{ij}(t) \left( f_{ij}(t) - \max_{t-1} f_{ij} \right) \bar{t}I, \tag{6.6}
\]

where \( a_{ij}(t) \) is a step-function of \( f_{ij}(t) \) that has the form of:

\[
a_{ij}(t) = \begin{cases} 
0 & f_{ij}(t) \leq \max_{t-1} f_{ij} \\
a_{ij} & \text{otherwise}
\end{cases}
\]

By substituting Eqn. (6.6) into the objective function, we get the following optimization problem that is equivalent to problem (6.1):

\[
\min_{f_{ij}^k(t)} \sum_{\{i, j\} \in \mathcal{E}} a_{ij}(t) f_{ij}(t) \bar{t}I. \tag{6.7}
\]

If we focus on decisions in the time interval \( t \), and drop the time indices in the problem expressions, we can see that the optimization problem (6.7) is in the form of the classic minimum-cost multicommodity flow problem.

However, none of the algorithms solving the minimum-cost multicommodity flow problem can be applied to solve this problem. The reason is that \( a_{ij}(t) \) is now a function of \( f_{ij}(t) \), which
results in a non-linear cost function. Since our objective is to find a simple yet efficient algorithm that can be readily implemented, we would like to study alternative and more tractable formulations of this problem.

### 6.5 Splitting and Routing Flows Optimally to Minimize Costs

From Eqn. (6.6), we find that the cost on a link up to time interval $t$ equals the sum of the cost on that link up to time interval $t - 1$, and the additional cost incurred by the possible overflow in the time interval $t$. This implies that, if the traffic volume on a link during one time interval is less than the charging volume on that link up to the previous time interval, which is the maximum aggregate flow rate on that link over $t - 1$ time intervals times the duration of one time interval, no additional costs will be incurred. In other words, the traffic volume in interval $t$ is carried on the link for free. However, if the traffic on that link exceeds the previous charging volume, the cloud provider will be charged extra for the overflow.

![Figure 6.3: An illustration of the operational costs on link \( \{i, j\} \).](image)

Fig. 6.3 illustrates the operational costs on link \( \{i, j\} \). Having the charging volume up to time interval $t - 1$, the same amount of traffic is already paid for in the time interval $t$, no matter if it is used up or not. The blue net area in the figure represents the already paid portion of the traffic volume, and the red diagonal area indicates the potential traffic volume that will
incur additional costs. This implies that, if the video flows on each link are assigned in a fashion that, the already paid portion of the traffic volume is utilized as much as possible, and the traffic volume with additional costs is minimized, the operational costs on inter-datacenter traffic up to time interval $t$ will be minimized as a result. Based on this idea, the design of Jetway is based on decoupling problem (6.1) into two sequential optimization problems.

### 6.5.1 Fully Utilizing the Already Paid Portion of Traffic Volume

Having $\max_{t-1} f_{ij} \bar{t}$ as the charging traffic volume on link $\{i, j\}$ up to time interval $t - 1$, the amount of already paid traffic flow in the time interval $t$ can be obtained as $\sum_{\{i, j\} \in E} \min(\max_{t-1} f_{ij}, c_{ij}(t) - d_{ij}(t)) \bar{t}$, in which the flow on each link is determined by its charging traffic volume up to time interval $t - 1$ and its available link capacity in the time interval $t$. Choosing from video flows initiated in the time interval $t$, we can obtain a subset $K_f(t)$, in which the sum of the desired transmission rates are the maximum possible amount that is no larger than the already paid traffic volume divided by the duration of a time interval in interval $t$. $K_f(t)$ is the already paid set of video flows to be carried during time interval $t$. In the ideal case, all video flows in $K_f(t)$ should be carried without incurring additional costs.

As a consequence, for video flows in $K_f(t)$, our objective is to find their feasible flow assignments, under the joint link capacity constraint, the flow conservation constraint, and the non-negative flow assignment constraint. The optimization presentation of this problem is to find the maximum fraction $z$, such that up to $z$ fraction of each flow’s desired transmission rate is assigned on links in the time interval $t$ [26]. Referred to as the maximum concurrent flow problem, it has the following form:

$$\max f_{ij}^k(t) \quad z \quad (6.8)$$

s.t. $\sum_{k \in K_f(t)} f_{ij}^k(t) + d_{ij}(t) \leq \min(\max_{t-1} f_{ij}, c_{ij}(t)), \forall \{i, j\}$

$\sum_{j \in V} f_{ij}^k(t) = \sum_{j \in V} f_{ji}^k(t), \forall k \in K_f(t), \forall i \in V / \{s_k, d_k\}$
\[
\sum_{j \in \mathcal{V}} f_{skj}^k(t) - \sum_{j \in \mathcal{V}} f_{jsk}^k(t) = zr_k(t), \forall k \in \mathcal{K}_f(t)
\]
\[
f_{ij}^k(t) \geq 0, \forall k \in \mathcal{K}_f(t), \forall \{i,j\} \in \mathcal{E}
\]
\[
0 \leq z \leq 1.
\]

Note that compared to the general maximum concurrent flow problem, optimization problem (6.8) has its additional constraints resulted from our objective to minimize costs on inter-datacenter video traffic. Instead of being restricted by the available link capacity only, the capacity constraint in problem (6.8) is further restricted by the already paid portion of video traffic on each link, which ensures that no potential cost is incurred.

There exists a fast combinatorial algorithm to approach the \(\epsilon\)-optimal solution of this problem. A flow assignment is said to be \(\epsilon\)-optimal if it overflows the link capacities by at most \(1 + \epsilon\) factor and has a cost that is within \(1 + \epsilon\) of the optimum. Since \(\epsilon\) can be defined as small as possible, the \(\epsilon\)-optimal solution can approach the optimum value as close as possible. It has been proved that the \(\epsilon\)-optimal flow can be computed deterministically in \(O(\epsilon^{-2}knm \log K \log^3 n)\) time, where \(K\) is the number of commodities (video flows), \(m\) is the number of links, and \(n\) is the number of datacenters in the cloud [63].

The general idea of this algorithm is to find the minimum dual variable \(\lambda\) that indicates the congestion of flows. The procedure is as follows. Routing each video flow separately, an initial flow assignment \(f^k\) that satisfies the desired transmission rate and obeys the link capacity constraint to the extent of \(\lambda_k\) is obtained for all \(k \in \mathcal{K}_f(t)\). Define a cost function \(b_k\) to each flow assignment \(f^k\) with respect to a non-zero, non-negative length function. Repeatedly examine all flows in \(\mathcal{K}_f(t)\) in a round-robin fashion. If a flow is found with a “bad” flow assignment by solving its corresponding minimum-cost flow problem, its flow assignment is updated. Note that the minimum-cost flow problem at each iteration can be solved by successive approximation, which has a running time of \(O(\min(nm \log n, n^{5/3}m^{2/3}, n^3) \log (nC))\) [39].

The combinatorial maximum concurrent flow algorithm is described in Algorithm 3, where a potential function \(\phi\) is used to guide the algorithm [63].
Algorithm 3 The combinatorial maximum concurrent flow algorithm.

1: Obtain the initial solution \((f_1, f_2, ..., f_{|K_f(t)|})\).
2: Set \(\alpha = 3(1 + \epsilon)\lambda^{-1} \ln m \epsilon^{-1}; \sigma = (2\epsilon)/(\alpha \lambda^{-1})\).
3: while \(\lambda_f > (1 - \frac{\epsilon}{2})\lambda \& \Delta \phi > \epsilon^2 \phi^{(k)}\) do
   4: for Each \(k \in K_f(t)\) do
      5: Find the minimum cost flow assignment \(f_k^\ast\) over \(P_k\).
      6: if \(b_k - b_k^\ast \geq \epsilon b_k\) then
         7: Update \(f_k = f_k + \sigma(f_k^\ast - f_k)\).
      8: end if
   9: end for
10: end while
11: Return \((f_1, f_2, ..., f_{|K_f(t)|})\).

6.5.2 Minimizing the Additional Cost

The optimal fraction \(z^\ast\) obtained by solving problem (6.8) indicates that at most \(z^\ast\) fraction of flows from \(K_f(t)\) can be carried using the already paid traffic volume. If \(z^\ast = 1\), all flows in \(K_f(t)\) can be carried without additional cost. If \(z^\ast < 1\), only \(z^\ast F_k\) of each video flow \(k\) incurs no extra cost, and the transmission of the remaining part of each flow does incur an additional cost. In this case, we use the same indicator \(k\) to denote the leftover part of the original video flow \(k\), which has a transmission rate of \((1 - z^\ast)r_k\).

Let \(K_c(t)\) denote the set of video flows that incur additional costs in the time interval \(t\), including both flows in \(K(t) - K_f(t)\) and the remaining partial flows in \(K(t)\) after solving optimization problem (6.8). Since the traffic of carrying video flows in \(K_c(t)\) is bound to incur additional costs, the cost function remains a linear function with a flat per-unit cost. As a result, the optimal flow assignment for these flows can be found by solving a minimum-cost multicommodity flow problem with a linear cost function in the time interval \(t\), which is exactly in the form of minimum-cost multicommodity flow problem.
The conventional way of solving this problem is to express it as a linear program, and then to solve it with a polynomial-time linear program solver [26]. Similar to the maximum concurrent flow problem, there exists a fast combinatorial algorithm to approach the $\epsilon$-optimal solution to this problem, and the running time of this combinatorial algorithm is proved to be $\tilde{O}(\epsilon^{-3}Kmn)$ [44].

The algorithm solves the problem as follows. Represent the flow assignment as $(f^1, f^2, ..., f^{|K_c(t)|}) \in (P^1, P^2, ..., P^{|K_c(t)|})$, where $f^k$ is the $|E| \times 1$ flow assignment vector for flow $k$, and $f^k \in P^k$. The polytope $P^k$ corresponds to the feasible flow assignments of flow $k$, i.e., flow assignments that obey the flow conservation constraint and the individual link capacity constraint, disregarding the rest of the video flows. At each iteration, a flow $k$ is randomly chosen for any $k \in K_c(t)$. Compute the minimum-cost flow assignment $f^k*$ over $P^k$, and update its flow assignment $f^k$ to $(1-\sigma)f^k + \sigma f^k*$ if it causes a decrease in the potential function $\phi$, which is used to guide the algorithm [44]. Similarly, the minimum-cost flow problem at each iteration can be solved by successive approximation [39]. The combinatorial minimum-cost multicommodity flow algorithm is described in Algorithm 4.

**Algorithm 4 The combinatorial minimum-cost multicommodity flow algorithm.**

1: Obtain the initial solution $(f^1, f^2, ..., f^{|K_c(t)|})$.
2: Set $\alpha = (1/\epsilon) \ln(3|E|)$; $\sigma = \Theta(1/\alpha^3)$.
3: while $\phi(f^1, f^2, ..., f^{|K_c(t)|}) > 3|E|$ do
4: Randomly choose $k \in K_c(t)$.
5: Find the minimum cost flow assignment $f^k*$ over $P^k$.
6: if $\phi(f^1, ..., f^k^*, ..., f^{|K_c(t)|}) < \phi(f^1, ..., f^k, ..., f^{|K_c(t)|})$ then
7: Update $f^k = (1-\sigma)f^k + \sigma f^k*$.
8: end if
9: end while
10: Return $(f^1, f^2, ..., f^{|K_c(t)|})$.

Our complete routing and flow assignment strategy in Jetway in the time interval $t$ is pre-
sent in Algorithm 5. As indicated by our problem formulation, the routing and flow assignment for video flows in Jetway is updated in a periodic fashion, which can be adjusted flexibly by cloud providers.

Algorithm 5 Routing and flow assignment strategy in the time interval $t$.

1: Compute the already paid traffic volume in the time interval $t$.

2: Obtain the set of video flows $K_f(t)$. 

3: For all flows $k \in K_f(t)$, find the optimal flow assignment $(f^{1}, f^{2}, ..., f^{|K_f(t)|})^*_f$ by solving the maximum concurrent flow problem using Algorithm 3.

4: Obtain the set of video flows $K_c(t)$. 

5: For all flows $k \in K_c(t)$, find the optimal flow assignment $(f^{1}, f^{2}, ..., f^{|K_c(t)|})^*_c$ by solving the minimum-cost multicommodity flow problem using Algorithm 4.

6: for Each flow $k \in K(t)$ do

7: if $k \in K_f(t) \& k \in K_c(t)$ then

8: $f^k = f^k_f + f^k_c$. 

9: else

10: if $k \in K_f(t)$ then

11: $f^k = f^k_f$. 

12: else

13: $f^k = f^k_c$. 

14: end if

15: end if

16: end for

17: Return $(f^{1}, f^{2}, ..., f^{|K(t)|})^*$. 


6.5.3 Implementation Issues

A key assumption in Algorithm 5 is that every datacenter is required to possess information about all video flows. If we need a distributed heuristic, there are both good and bad news. The good news is that there exist distributed algorithms to approach the $\epsilon$-optimal solution of the maximum concurrent flow problem [15], which we use to maximize the utilization of the already paid portion of the traffic volume. The bad news is that the design of efficient distributed solutions to the minimum-cost multicommodity flow problem — which we use to minimize additional traffic costs — remains an open and elusive problem in theoretical computer science [16]. Fortunately, since all datacenters are operated by the same cloud provider, we believe that, partly due to the small number of datacenters in use, it is technically straightforward to devise a centralized controller to obtain and access information about all the video flows, which makes the algorithm design in Jetway much less dependent on the availability of distributed algorithms.

Although the 100-percentile charging model is used as an example in this chapter, our algorithm is not only limited to this specific charging model. Instead, it can be applied to any percentile-based charging model as a feasible approach to reduce traffic cost when no workload prediction is considered. With a general $q$-th percentile charging model, the charging volume will be the $q \times I$-th largest traffic volume over all time intervals in a charging period, which implies that traffic volumes from the $q \times I + 1$-th to the largest time intervals can be as large as possible since they will not generate additional costs. Again, we seek to find out the optimal routing and flow assignment strategy we can apply at the current time interval, with the objective of minimizing operational costs with a $q$-th percentile charging model.

To be specific, if we sort the aggregate flow rate on link $\{i, j\}$ from time interval 1 to the last time interval $I$ in a decreasing order, and use $qt(f_{ij}, q)_I$ to denote the $q \times I$-th value in this time series of the aggregate flow rates, the already paid traffic volume in the time interval $t$ can be represented by a general form of $\sum_{\{i,j\} \in E} \min(qt(f_{ij}, q)_I, c_{ij}(t) - d_{ij}(t)) t$. Then Algorithm 5 can be applied to make the routing and flow assignment strategy with any given $q$-th percentile
charging model.

## 6.6 Performance Evaluation

We believe that the best way to evaluate Jetway is to implement it, and in this section, we evaluate how Jetway performs in the Amazon EC2 inter-datacenter network. Our real-world experimental results have validated that, by optimally finding routing paths and assigning flows for videos flows at each time interval, Jetway reduces costs on inter-datacenter video traffic by a substantial margin.

### 6.6.1 Implementation

Our Jetway implementation contains more than 9000 lines of C++ code, and is developed from scratch. To evaluate our proposed algorithms in the Amazon EC2 inter-datacenter network, we need to send actual video traffic between source-destination pairs, over either single-hop or multi-hop paths, with possibilities of splitting flows. For this reason, we have implemented a daemon process that is able to send, receive, and relay video traffic, referred to as a Jet. By using asynchronous event-driven networking provided by the Boost asio C++ framework, it is designed with performance and scalability in mind. Since it is supposed to be running in cloud VMs, the Jet supports major UNIX variants and Windows. As compared to the traditional thread pool concurrency model, our implementation incurs less memory and CPU overhead, even at high packet processing rates.

Fig. 6.4 has shown the architecture overview of a Jet, comprising of a number of key components.

**Video Flow Manager.** This component manages all ongoing video flows within each Jet, where all video flows are included in the Video Flow List. When the source-destination pair corresponding to a new video flow is initiated, the Video Flow Manager will add the corresponding flow in the video flow list, and create an I/O proxy for transmission between the
storing video and the Jet at the source datacenter. The Jet at the source is responsible for determining the routing paths and their corresponding transmission rates for each video flow, based on our Jetway algorithms. All data packets are then passed on to the Transmitter.

Transmitter. For data packets within a video flow, the Transmitter retrieves its route from the Video Flow List, and send them to the corresponding next-hop Jet, which is either the destination Jet over a direct path, or a relay Jet in a multi-hop path with routing information embedded. Further, the flow rate on each path is carefully controlled by the rate control algorithm, so that they conform to the decisions made by the Jetway algorithms.

Receiver. As an incoming data packet arrives, the Receiver identifies which destination datacenter that packet is addressed to. If it is addressed to the datacenter this Jet is located at, the packet will be saved via the I/O proxy. Otherwise, if it belongs to a multi-hop relay path, the packet will be forwarded by the Transmitter to the next-hop jet according to the routing path carried in the header.

Port. This component maintains active connections to all Jets for video flows, each of which is located in a different datacenter. The Port is also able to probe link capacities and detect connection failures, such that the routing and flow assignment for each flow can be adjusted.
adaptively as the network status evolves. The Jet is designed to support a large number of concurrent video transmissions. Both higher-level Transmitter and Receiver and the lower-level network Port support multiplexed inter-datacenter connections to reduce additional overhead consumed by each video flow.

### 6.6.2 Experiments in the Amazon EC2 Cloud

We have conducted our Jetway experiments in the Amazon EC2 Cloud, one of the dominant Infrastructure as a Service (IaaS) cloud providers. Fig. 6.5 shows the inter-datacenter network topology in the Amazon EC2 cloud that we have used in our experiments. We have launched 7 standard on-demand medium instances with 2 compute unit and 1.7 GB memory in each of the datacenters, installed with our Jetway implementation. We log all statistics on every link for each video flow every 30 seconds, including the actual transmission rate, receiving rate, and the end-to-end delay.

![Amazon EC2 Cloud Network Topology](image)

**Figure 6.5:** The network topology used in our experiments.

We have first obtained the link capacities (Mbps) in the inter-datacenter network through saturating the outgoing link of each Jet. The results are listed in Table 6.2, showing the average link capacity between each datacenter pair that we observe in 3 minutes. Since costs charged by ISPs on inter-datacenter links in the Amazon cloud are not revealed, we assume a certain cost
per Mbps per time interval (5 minutes) on each link, reflecting different costs on transmitting the same amount of video. Costs we used in our experiments are also listed in Table 6.2.

Table 6.2: Link Capacities and Costs per Traffic Unit in the Amazon EC2 Inter-Datacenter Network

<table>
<thead>
<tr>
<th>Capacity/Cost</th>
<th>North Calif.</th>
<th>Oregon</th>
<th>Virginia</th>
<th>Sao Paulo</th>
<th>Ireland</th>
<th>Singapore</th>
<th>Tokyo</th>
</tr>
</thead>
<tbody>
<tr>
<td>North Calif.</td>
<td>—</td>
<td>520.40/1</td>
<td>252.67/2</td>
<td>116.75/15</td>
<td>98.67/20</td>
<td>103.69/27</td>
<td>173.06/30</td>
</tr>
<tr>
<td>Oregon</td>
<td>545.06/1</td>
<td>—</td>
<td>215.84/3</td>
<td>81.18/17</td>
<td>104.22/15</td>
<td>81.99/25</td>
<td>152.75/27</td>
</tr>
<tr>
<td>Virginia</td>
<td>240.78/2</td>
<td>210.64/3</td>
<td>—</td>
<td>139.41/10</td>
<td>221.55/10</td>
<td>81.44/15</td>
<td>110.10/17</td>
</tr>
<tr>
<td>Sao Paulo</td>
<td>40.98/15</td>
<td>60.84/17</td>
<td>11.85/10</td>
<td>—</td>
<td>22.41/25</td>
<td>9.59/18</td>
<td>62.42/22</td>
</tr>
<tr>
<td>Ireland</td>
<td>106.45/20</td>
<td>135.02/15</td>
<td>215.85/10</td>
<td>90.12/25</td>
<td>—</td>
<td>77.89/23</td>
<td>76.10/20</td>
</tr>
<tr>
<td>Singapore</td>
<td>124.40/27</td>
<td>110.95/25</td>
<td>84.54/15</td>
<td>57.31/18</td>
<td>80.92/23</td>
<td>—</td>
<td>242.80/5</td>
</tr>
<tr>
<td>Tokyo</td>
<td>178.36/30</td>
<td>143.44/27</td>
<td>99.40/17</td>
<td>61.99/22</td>
<td>43.61/20</td>
<td>116.33/5</td>
<td>—</td>
</tr>
</tbody>
</table>

We conduct our experiments for one hour, which has 12 time intervals in total. We consider the scenario that 10 videos are to be replicated to the datacenter located at Singapore at the beginning of our experiments, with 3 of them being Standard Definition (SD) videos with sizes uniformly random between [500, 800] MB, and 7 of them being High Definition (HD) videos with sizes uniformly random between [2, 4] GB. Source datacenters of these videos are randomly selected from the remaining 6 datacenters in the EC2 cloud. If we assume all video replications have to be finished within 30 minutes, we get the desired transmission rate of each video, ranging from [2.22, 17.78] Mbps. We further assume that there are 3 inter-datacenter video flows satisfying requests from CDN edge servers at the beginning of each time interval, with the source and destination datacenters randomly selected from all 7 datacenters in the cloud. The desired transmission rate for these videos are assumed to be uniformly random between [2.5, 8] Mbps, which are standard rate requirements for today’s video streaming services.

For fair comparisons, we have also implemented an alternative routing solution that performs better than the straightforward approach in terms of reducing traffic costs. Referred to
as the *time-slotted minimum cost algorithm* (TSMC), it considers different costs per traffic unit on each link only, without considering important aspects of the time dimension. In particular, TSMC solves the multi-commodity minimum cost problem using **Algorithm 4** at each time interval, and obtains the routing paths and flow assignments for the set of videos to be transmitted, based on current available link capacities. To study the effects of a more realistic percentile-based charging model, we have implemented Jetway and TSMC, with both the 100-th percentile charging model and the 95-th percentile charging model, and compared their performance with different charging models.

We first present the main performance metric, the total cost per traffic unit over all links in the inter-datacenter network over time. Fig. 6.6 shows the total cost per traffic unit in the Amazon EC2 inter-datacenter network over one hour by using Jetway and TSMC, respectively, with the 100-th percentile charging model. As we can observe, starting from minute 15, Jetway is substantially more cost-effective than TSMC. The benefits of Jetway is becoming increasingly visible as time elapses, and achieves a cost reduction of 19% after one hour with the 100-th percentile charging model.

![Figure 6.6: Cost per traffic unit with the 100-th percentile charging model.](image1)

![Figure 6.7: Cost per traffic unit with the 95-th percentile charging model.](image2)

Note that both algorithms incurred almost the same cost over the first few time intervals
in our experiment. The rationale is that, due to the limited number of video flows over a large number of inter-datacenter links, the already-paid traffic volume can hardly be utilized to the benefit of cost reduction. For each video flow, both algorithms lead to similar optimal routing and flow assignments, by solving the same multi-commodity minimum cost problem. Possibilities of utilizing the already-paid traffic volume increase as time elapses, which results in better performance with Jetway. Take the transit server-to-customer video flow (Video 20) from the Ireland datacenter (5) to the Oregon datacenter (2) as an example. Fig. 6.8 shows the routing and flow assignment for this video flow in Jetway. We can see that, initiated at minute 20, the flows of this video takes both cheaper paths 5 → 3 → 2 and 5 → 3 → 1 → 2, and expensive paths 5 → 6 → 3 → 2 and 5 → 6 → 3 → 1 → 2, taking advantage of links 5 → 6 and 6 → 3 that are already used for video replication at the first time interval.

```
{   "Protocol": "TCP",   "PacketSize": 4096,   "AverageRate": 3,   "NodeID": 5,   "RoutingList": [     { "PathList": ["3,1,2"], "Weight": 0.334967 },     { "PathList": ["3,2"], "Weight": 0.334967 },     { "PathList": ["6,3,1,2"], "Weight": 1.8986165 },     { "PathList": ["6,3,2"], "Weight": 0.4314495 }   ],   "VideoID": 20,   "StartTime": 1200
}
```

Figure 6.8: Routing and flow assignment for Video 20 in Jetway.

We also present the total cost per traffic unit using both Jetway and TSMC with the 95-th percentile charging model in Fig. 6.7, where we can observe a similar trend. With smaller absolute values in costs, the reduction on the total cost per traffic unit with Jetway is up to 8% in this case. Our results have confirmed that Jetway can reduce costs on inter-datacenter video traffic substantially, even with the (more realistic) 95-th percentile charging model.

To investigate how our Jetway algorithms affect the performance of video streaming services, we record the receiving rates for each video flow every 30 seconds in our experiment.
We define the normalized receiving rate to be the actual receiving rate observed at the destination datacenter divided by the desired transmission rate of this video flow. Fig. 6.9 shows the normalized receiving rates for 5 video flows from the 2nd time interval (minute 5) to the 7th time interval (minute 30). Each of them starts its transmission at the beginning of every time interval, respectively. As we can see, the normalized receiving rates for 3 video flows are 1 most of the time, with minor fluctuations; and the normalized receiving rates for Video 15 and Video 17 exhibit obvious fluctuations, which is resulted by the unstable network status on their common link 3 → 6.

We also plot the CDF of the normalized receiving rates for all video flows using Jetway in our experiment. Shown in Fig. 6.10, over 91.5% video flows using Jetway exhibit receiving rates that are more than 90% of the desired transmission rates at their destination datacenters. The mean value of the normalized receiving rate is 0.97 by using Jetway, which shows that the performance of video streaming regarding the receiving rates is satisfactory by using Jetway.

Another important performance metric in video streaming service is the end-to-end delay experienced by each video flow. Since Jetway allows multi-path multi-hop transmission, we are interested in investigating whether it will affect the end-to-end delay when transmitting
videos. Fig. 6.11 shows the CDF of the end-to-end delays experienced by all video flows using Jetway in our experiment. We can observe that 90\% of the video flows experienced end-to-end delays that are smaller than 587.3 ms, and the mean value of end-to-end delays for all videos is 349.09 ms. We believe that these values are reasonable, considering the nature of long-distance intercontinental inter-datacenter video transmissions.

To further investigate the effect of Jetway on end-to-end delays for video streaming, we take the Singapore datacenter (6) as an example, and compute the average end-to-end delays experienced by video flows from every other datacenter to this one, together with their 95\% confidence intervals. For comparisons, we also compute the average end-to-end delays in each direct link, which are obtained by the measured round trip time (RTT) on each link divided by 2, with their corresponding 95\% confidence intervals as well. The results are shown in Fig. 6.12. As we can see, video flows in Jetway have experienced almost the same end-to-end delays as compared to those by sending through direct paths. Our results prove that the multi-path multi-hop routing and flow assignment algorithms in Jetway will not affect the performance of video streaming services significantly, yet the deployment of Jetway leads to substantially reduced operational costs on inter-datacenter video traffic.

Figure 6.11: CDF of the end-to-end delays for all video flows using Jetway.

Figure 6.12: Average end-to-end delays to the Singapore datacenter (6).
6.6.3 Simulation Results

To investigate the scalability and stability of Jetway, we further evaluated it in a time-slotted simulator. The simulated inter-datacenter network has 20 datacenters, forming a complete graph. The capacity on each link is assumed to be 1000 units, and the cost per traffic unit on each link is set to be uniformly random within $[1, 100]$. In each time interval, the number of video flows to be transmitted is uniformly random between $[1, 200]$, each with a desired transmission rate uniformly random between $[1, 10]$ units. The source and destination datacenters of each video flow are also chosen uniformly random from the datacenter set. We conduct our simulations 20 times, each lasts for 100 time slots.

![Graph 1](image1.png)

Figure 6.13: Average percentage of cost reduction with the 100-th percentile charging model.

![Graph 2](image2.png)

Figure 6.14: Average percentage of cost reduction with the 95-th percentile charging model.

Again, the main performance metric we are interested in is the reduction in the total cost per traffic unit over time by using Jetway, as a percentage of the normalized cost reduction ratios compared to TSMC and their 95% confidence intervals. Fig. 6.13 and Fig. 6.14 show the average percentage of reduction in the total cost per traffic unit on video traffic with both the 100-th percentile charging model and the 95-th percentile charging model, respectively. Consistent with our experimental results shown in Fig. 6.6 and Fig. 6.7, the normalized cost
reduction ratio by applying Jetway is increasing as time elapses, and reaches up to an average of 13% after 100 time intervals with the 100-th percentile charging scheme. The similar trend is observed with the 95-th percentile charging model, with even more cost reduction of up to 20% in the long term. Our results further confirmed that Jetway can successfully reduce costs on inter-datacenter video traffic, even in inter-datacenter networks of larger scales and many video flows.

Figure 6.15: Average percentage of flows carried by utilizing the already paid portion of traffic volume with the 100-th percentile charging model.

Figure 6.16: Average percentage of flows carried by utilizing the already paid portion of traffic volume with the 95-th percentile charging model.

To further study the effects of the 95-th percentile charging model on Jetway, we show the portion of video flows carried by utilizing the already paid portion of traffic volume in Jetway with both the 100-th percentile and the 95-th percentile charging models. Shown in Fig. 6.15 and Fig. 6.16, around 75% of video flows are carried by utilizing the already paid portion of traffic volume with both charging models. These results have provided further solid evidence that Jetway works effectively when it comes to utilizing the “free of charge” bandwidth that is available during later time intervals in the same charging period, made possible by any percentile-based charging models typically used by ISPs.
Chapter 6. Jetway: Minimizing Costs on Inter-Datacenter Traffic

6.7 Summary

To stream videos to end users, it is now the norm for video streaming service providers, such as Netflix, to use services of cloud providers, such as Amazon Web Services. To provide a better level of performance, the cloud provider typically deploys multiple datacenters across different geographical regions, and connect them with an inter-datacenter network. Recent research reveals that traffic costs amount to a large portion of operational costs incurred to a cloud provider, and among those, inter-datacenter traffic occupies around half of the traffic going through a datacenter’s egress router. For a cloud provider to reduce its operational costs efficiently, it is desired that the cost incurred by the inter-datacenter traffic be minimized. Inter-datacenter traffic is typically charged by ISPs based on a percentile-based charging model, with which cloud providers pay based on the $q$-th percentile of traffic volumes measured in a short time interval, over a number of such intervals in a charging period. Such a model implies that, if traffic has already been generated during one time interval, up to the same volume of traffic may be carried free of charge in subsequent time intervals.

In this chapter, we have presented Jetway, designed to minimize costs on inter-datacenter video traffic by splitting and routing video flows over multiple multi-hop paths. Jetway takes full advantage of our key observation that some of the traffic volumes can be transferred free of charge, while the desired transmission rates of video flows remain satisfied. In order to design Jetway, we have formulated the problem of minimizing costs with the intent of maintaining its tractability and the practicality of our solutions. We have evaluated the performance of Jetway using our real-world implementation, with actual traffic flowing across seven Amazon EC2 datacenters around the world. We have shown that Jetway is capable of reducing traffic costs, while maintaining satisfactory performance with respect to both throughput and end-to-end delay.
Chapter 7

Postcard: Minimizing Costs with Store-and-Forward

7.1 Overview

In Chapter 6, we have motivated the design of new algorithms for cloud providers to minimize their operational costs on inter-datacenter traffic, while keeping their customers satisfied. We reiterate our view that, with diverging cost structures used by various ISPs and unique characteristics of inter-datacenter traffic, it is feasible to minimize traffic costs due to two reasons. Spatially, the cost of transmitting the same amount of traffic varies significantly across different inter-datacenter overlay links. For example, domestic traffic is substantially cheaper than traffic to global destinations [74], and traffic within the backbone network built by cloud providers themselves incurs much lower operational costs. Temporally, a large portion of inter-datacenter traffic is delay-tolerant, including backups, propagation of large updates, and migration of customer data. To reduce costs, we may re-route such traffic with intermediate datacenters as relay nodes, splitting traffic into smaller fractions, and transmitting them along multiple routing paths [49].

However, as we formulate the problem that motivates the design of Jetway in Chapter 6,
we have not yet considered an important observation: intermediate datacenters are able to temporarily store the data to be relayed, and forward them at a later time to its downstream relay node or to the destination. Such an ability for intermediate nodes to store-and-forward does not reduce the amount of traffic to be relayed; yet the delayed forwarding may reduce peak traffic demand over an overlay link, and as such reduce operational costs over a longer charging period, if the ISP’s charging scheme hinges upon such peak traffic demand.

At first glance, we only need to formally formulate the problem of minimizing operational costs, in the general case that data, from multiple sources to their destinations, can be temporarily stored at intermediate datacenters for a period of time, and be fractionally split to multiple paths. Unfortunately, as we will show in this chapter, the cost minimization problem in the general case — and in fact even a much simplified problem based on data flows without temporal storage on intermediate nodes — is challenging to solve.

In this chapter, we present **Postcard**, an online optimization problem carefully formulated to minimize operational costs on inter-datacenter traffic, with intermediate nodes being able to store incoming data and forward them at a later time to reduce peak traffic demand. **Postcard** is formulated as a tractable convex optimization problem only with linear constraints (e.g., on link capacity and traffic conservation), to be solved with a standard solver (e.g., with interior-point methods). In order to achieve this goal, **Postcard** is formulated with minimal simplification, much like a time-slotted model, where a data file starts its transmission at the beginning of a time interval, and finishes at the end of it. The key idea towards making the problem tractable to solve is to construct a time-expanded graph over multiple time intervals. With extensive simulations, we compare results from solving **Postcard** to those from solving a flow-based problem, **Jetway** as an example, and present the advantages and drawbacks of store-and-forward when it comes to minimizing costs on inter-datacenter traffic. To our knowledge, **Postcard** represents the first attempt to systematically study and formulate the problem of minimizing operational costs on inter-datacenter traffic, with the ability of intermediate nodes to store incoming data and forward them at a later time.
We first present the rationale and challenges of minimizing operational costs on inter-datacenter traffic, followed by a brief discussion on a flow-based simplification to address some of the challenges, yet still capturing the essence of the problem.

7.2 Rationale and Challenges

As we have stated in Chapter 6, cloud providers deploying a number of geographically distributed datacenters today, usually lease bandwidth from multiple ISPs for their inter-datacenter traffic. They are charged based on the predominant percentile-based charging scheme, i.e., the $q$-th percentile charging scheme, in which operational costs are determined by the amount of traffic each cloud provider generates. To be more precise, an ISP records the traffic volume a cloud provider generates during each 5-minute interval. At the end of a complete charging period, all 5-minute traffic volumes are sorted in an ascending order, and the $q$-th percentile is used as the charging volume $x$ to derive the cost by a piece-wise linear non-decreasing function $c(x)$ [40].

Since cloud providers usually over-provision bandwidth resources to guarantee their peak-hour performance, percentile-based charging schemes may cause a substantial waste of capital investment, especially when a strong diurnal pattern is observed in inter-datacenter traffic [23]. When a certain amount of traffic is already generated between two datacenters during their peak hours, the same costs are incurred in subsequent time intervals even if the link between these two datacenters is idle or under-utilized (e.g., during off-peak hours), as the cloud provider will have to pay for subsequent time intervals based on the already generated traffic volume nonetheless.

The geographically distributed locations of datacenters have provided a possible solution to reduce such costs on inter-datacenter traffic, as different ISPs charge traffic based on different prices per unit of traffic, and traffic demand differs at different datacenters at any given time (perhaps due to distinct time zones they reside). If traffic from one datacenter is allowed
to be fractionally split into sub-flows, even to be stored temporarily by intermediate datacen-
ters, cloud providers may benefit from “cheaper” routing alternatives, and time intervals along
links that are already “paid” may be utilized more efficiently. In other words, if routing paths,
flow assignments and scheduling strategies for each source-destination traffic pair are care-
fully designed, operational costs on inter-datacenter traffic can be efficiently reduced, or even
minimized.

The following example illustrates the benefits by considering routing and scheduling strate-
gies for inter-datacenter traffic. Shown in Fig. 7.1, datacenter D_2 needs to send a file of size
6 MB within 15 minutes to datacenter D_3, and the same cloud provider also operates another
datacenter D_1, which is inter-connected with the other two datacenters. For simplicity, we as-
sume that the 100-percentile charging scheme is in use, i.e., the maximum traffic volume sent
during time intervals will be the traffic volume to be charged; the cost function between each
datacenter pair is the volume to be charged multiplied by a flat price (per MB) shown on the
link; and the link capacity is sufficiently large, e.g., 1 Gbps for conventional optical links, so
that it is not a constraint in this example.

(a) Without routing and scheduling

(b) With routing and scheduling

Figure 7.1: How traffic costs can be reduced with routing and scheduling strategies: a motivat-
ing example.

As we can see, the flat price between each datacenter pair differs from one another, due
to possible reasons of going through several paid peering ISPs, domestic regional pricing,
and pricing discounts due to ISP backplane peering relationships [74]. Without routing or
scheduling considerations, as shown in Fig. 7.1 (a), the file will be sent directly to D_3 within
three 5-minute intervals, and the cost per time interval is \(10 \times 2 = 20\). However, if routing and scheduling strategies are considered, the file can be split evenly into two blocks, and transmitted sequentially along the path \(\{D_2 \rightarrow D_1 \rightarrow D_3\}\) through three 5-minute intervals. As Fig. 7.1 (b) indicates, the cost per time interval can be reduced to \(1 \times 3 + 3 \times 3 = 12\), which is much lower than that in the former case.

Unfortunately, the design of such routing and scheduling strategies in datacenters with multiple source-destination traffic pairs and different transfer time requirements involves formidable challenges. By allowing possible storage at intermediate datacenters, the complexity of scheduling strategies is increased significantly. Upon receiving data blocks, each intermediate datacenter has to make decisions on not only to relay them immediately or to store them for a while, but also how much time should each data block be stored and at what rate should it be transmitted to the next “hop” along the path. Since both waiting times at intermediate datacenters and the transmission rate on every link along the routing path will affect the transfer completion time of each traffic pair, those decisions are hard to make, especially if an optimization objective is involved. The problem is further complicated by the lack of synchronization among all inter-datacenter traffic pairs: traffic can be generated at any time in a charging period; each data block can be stored at intermediate datacenters for an arbitrary period of time; and its transmission can be completed at any time, as long as the total transfer completion time does not exceed its maximum tolerable transfer time.

### 7.3 Jetway Review: A Flow-Based Approach

Due to these difficulties involved when designing optimal routing and scheduling strategies with the objective of minimizing traffic costs, simplifications are necessary to make the problem tractable. One possible way is to completely eliminate the possibility of temporal storage at intermediate datacenters, so that each source-destination traffic pair can be represented by a *flow* with its desired transmission rate. The resulted routing and flow assignment problem
for all inter-datacenter flows can be formulated by a flow-based model, which is practically solvable with efficient combinatorial algorithms, yet the essence of which is still captured.

In this approach, the desired transmission rate of each traffic pair can be obtained by the quotient of the traffic size divided by its maximum tolerable transfer time, which equals the minimum tolerable transmission rate in the interest of minimizing traffic costs. For example, the desired transmission rate of the file in Fig. 7.1 is \( \frac{6 \text{ MB}}{15 \text{ min}} = 54.6 \text{ kbps} \). If each inter-datacenter flow can still be transmitted to its destination datacenter via multiple paths, each with multiple hops, the original problem becomes a simpler problem of how routing paths are determined and how flows are assigned to these paths, such that the incurred traffic costs are minimized.

As we have discussed in Chapter 6, even the solution to such a simpler problem, based on the flow-based model, is still quite elusive. Although the problem shares some similarity with the combinatorial minimum-cost multicommodity flow problem [44], the non-linear cost function caused by the percentile-based charging scheme makes it impossible to directly apply existing algorithms, such as [39], to solve this kind of problems. We believe that this problem can be decoupled into two sub-problems. The first problem solves the routing and flow assignment problem for a subset of traffic pairs, with the objective of fully utilizing the already paid traffic volume in the inter-datacenter network; and the second one finds the optimal routing paths and flow assignments for the rest of traffic pairs, aiming at minimizing additionally incurred traffic costs. Since these two sub-problems fall in the form of the maximum concurrent flow problem and the minimum-cost multicommodity flow problem, respectively, the original cost minimization problem with the flow-based model can be solved by solving these two sub-problems sequentially.
### Table 7.1: Notations and Definitions

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G = (V, E)$</td>
<td>the inter-datacenter network</td>
</tr>
<tr>
<td>$K$</td>
<td>the set of files to be transferred across datacenters</td>
</tr>
<tr>
<td>$c_{ij}$</td>
<td>the available capacity on link ${i, j}$</td>
</tr>
<tr>
<td>$a_{ij}$</td>
<td>the cost per traffic unit on link ${i, j}$</td>
</tr>
<tr>
<td>$(s_k, d_k)$</td>
<td>the source and destination datacenter of file $k$</td>
</tr>
<tr>
<td>$F_k$</td>
<td>the size of file $k$</td>
</tr>
<tr>
<td>$T_k$</td>
<td>the maximum tolerable transfer time of file $k$</td>
</tr>
<tr>
<td>$t_k$</td>
<td>the transfer starting time of file $k$</td>
</tr>
<tr>
<td>$\bar{t}$</td>
<td>the duration of one time interval</td>
</tr>
<tr>
<td>$I$</td>
<td>the number of time intervals in a charging period</td>
</tr>
<tr>
<td>$M_{ij}^k(t)$</td>
<td>the size of a fraction of file $k$ to be transferred on link ${i, j}$ at time interval $t$</td>
</tr>
<tr>
<td>$\max_I M_{ij}$</td>
<td>the maximum aggregate traffic on link ${i, j}$ of all time intervals in a charging period with $I$ time intervals</td>
</tr>
<tr>
<td>$G(I)$</td>
<td>the time-expanded graph for inter-datacenter network $G$</td>
</tr>
<tr>
<td>$M_{ij}^{i,t}$</td>
<td>the size of a fraction of file $k$ to be transferred from node $i_t$ to $j_{t+1}$ in $G(I)$</td>
</tr>
</tbody>
</table>

### 7.4 System Model

In *Postcard*, we are interested in making optimal decisions to minimize the cost on inter-datacenter traffic to a cloud provider in an online fashion. Since inter-datacenter traffic cannot be accurately predicted beyond a few seconds [18], we seek to answer the following question: at a certain time $t$, when multiple source-destination traffic pairs are to be transferred among datacenters, what is the optimal routing and scheduling strategy we can apply to minimize the cost incurred by this traffic, under the assumption that all routing paths and flow assignments for previous traffic pairs are already known? Before formulating this problem formally, we first introduce our system model in this chapter. Important notations used throughout this chapter are listed in Table 7.1.

In our model, *files* are used to represent all inter-datacenter traffic. The term is used in a
very generic fashion, in that it means a block of data to be transmitted across the boundary of
datacenters with its own size and maximum tolerable transfer time, and does not necessarily
have to be related to file systems. For example, a file in our context can be a set of intermediate
results in MapReduce tasks. Let $\mathcal{K}(t)$ be the set of files to be transmitted at time $t$. For each
file $k \in \mathcal{K}(t)$, we use a four-tuple specification $(s_k, d_k, F_k, T_k)$ to describe it. Here, vertices
$s_k$ and $d_k$ indicate the source and the destination datacenter from and to which file $k$ is being
transmitted, $F_k$ is the size of file $k$, and $T_k$ describes the maximum tolerable transfer time of
file $k$. The general case that a file can have multiple destinations can be handled by introducing
a separate file to each source-destination pair, with the same source datacenter, file size, and
maximum tolerable transfer time.

We consider an inter-datacenter network that consists of multiple geographically distributed
datacenters, operated by a single cloud provider. Such an inter-datacenter network can be
denoted by a complete directed graph $G = (V, E)$, where $V$ indicates the set of datacenters, and
$E$ indicates the set of overlay links inter-connecting datacenters. Each datacenter is connected
to all other datacenters through several ISPs. For each link $\{i, j\} \in E$, we use a positive
function $c_{ij}(t)$ to denote the available link capacity at time $t$, which is the maximum residual
link capacity left after some of the capacity is used for the transmission of files before $t$. Since
each file’s ongoing transfer will remain in the network for some time, the transmission of a
previously generated file will affect the available link capacity at the current time. We also
use a nonnegative real-value function $a_{ij}$ to denote the cost per traffic unit transferred from
datacenter $i$ to $j$.

Since traffic volume is computed periodically in every 5-minute time interval in percentile-
based charging schemes, we slot the time dimension into multiple time intervals with the same
duration, denoted by $\bar{t}$. If the number of time intervals in a charging period is $I$, the time di-
mension in this charging period can be represented by $\{t|0 \leq t \leq I\}$. To make the problem
tractable, we assume that files in $\mathcal{K}(t)$ is sufficiently small, such that each file is guaranteed
to be completely received over an overlay link by the downstream datacenter within one time
interval. Such an assumption is valid in general, since most data to be transferred across data-
centers are within a few hundred Gigabytes, and overlay links that interconnect datacenters are
usually designed with high capacities, such as OC-192 with thousands of miles of connected
fiber in SevenL Networks [6], one of the datacenter infrastructures in North America. OC-192
is a network link with transmission rates of up to 1.2 GB/second, allowing the complete trans-
fer of 360 GB of data within one time interval of 5 minutes. For even larger files, they can
be divided into smaller pieces, each of which can be considered as a new file with the same
four-tuple specification.

We further assume that the inter-datacenter transmission of a file starts at the beginning
of a time interval, and finishes at the end of that interval. It is obviously feasible for a file
to be completely transferred to the downstream datacenter with a higher rate, so that it is
received in less time than the duration of a time interval. Yet, for the sake of minimizing
traffic costs incurred on inter-datacenter links, and with the assumption that shorter transfer
times do not lead to any higher utility to the cloud provider due to the delay-tolerant nature of
inter-datacenter traffic, it is desirable to finish the transfer with exactly one time interval. More
formally, if a fraction of file $k$ with size $M_{ij}^k(t)$ to be transferred from datacenter $i$ to datacenter
$j$ at time interval $t$, the corresponding flow rate of file $k$ on link $\{i, j\}$ at time interval $t$ equals
$M_{ij}^k(t)/t$. During its transmission, a file can be stored temporarily in intermediate datacenters
before being relayed to the destination datacenter, and can be transferred to the destination
datacenter over multiple paths, as well as over multiple hops within a path.

\section{Formulating the Problem: the First Try}

Based on our time-slotted model, the problem of minimizing costs on inter-datacenter traffic
in Postcard can be formally stated as: with all routing paths and flow assignments of files
transferred before $t$ known, for every file to be transferred at time interval $t$, we seek to make
optimal decisions with respect to a few dimensions of the design space. At the source datacen-
ter, we will decide the proper number of fractions a file should be divided into, and compute forwarding paths via intermediate datacenters for each fraction to follow. At each intermediate datacenter, upon receiving a fraction of a file, we need to decide whether it needs to be forwarded immediately to a downstream datacenter along its path, or held temporarily for later forwarding, following the philosophy of store-and-forward.

Without loss of generality, we assume the cost function to be linear, i.e., \( c(x) = ax \), where \( x \) is the traffic volume to be charged. We also assume that the 100-th percentile charging scheme is being used for simplicity. To be specific, if we use \( M_{ij}^k(n) \) to denote the size of a fraction of file \( k \) to be transferred along link \( \{i, j\} \) at time interval \( n \), the inter-datacenter traffic volume to be charged on link \( \{i, j\} \) after transmitting files generated up to time \( t \) equals

\[
X_{ij}(t) = \max\{X_{ij}(t-1), \max_{\max_k T_k} \sum_{k \in K(t)} M_{ij}^k(n)\},
\]

where \( \max_{\max_k T_k} \sum_{k \in K(t)} M_{ij}^k(n) \) is the maximum aggregate traffic volume on link \( \{i, j\} \) of time intervals \( \{n|t \leq n \leq t + \max_k T_k\} \). Illustrated in Fig. 7.2, the shadowed area represents time intervals to be optimized. For simplicity, we assume all files can finish their transmission within a charging period, i.e., \( t + \max_k T_k \leq I, \forall k \). As a special case, \( M_{ii}(n) \) represents the volume of data that is stored temporarily from time interval \( n \) to \( n+1 \) — but not forwarded — at datacenter \( i \), referred to as the holdover. More formally, the problem of minimizing costs on
inter-datacenter traffic in Postcard can be formulated as the following optimization problem:

\[
\min_{M_{ij}^k(n)} \sum_{\{i,j\} \in E} a_{ij} X_{ij}(t) I \quad (7.1)
\]

\[
s.t. \quad \sum_{k \in K(t)} M_{ij}^k(n) \leq c_{ij}(n) \bar{t}, \forall \{i,j\} \in E \quad (7.2)
\]

\[
\sum_{j \in V} \sum_{n=t}^{t+\max_k T_k} (M_{skj}^k(n) - M_{jsk}^k(n-1)) = F_k,
\]

\[
\sum_{j \in V} \sum_{n=t}^{t+\max_k T_k} (M_{djk}^k(n) - M_{jdtk}^k(n-1)) = -F_k,
\]

\[
\sum_{j \in V} M_{ij}^k(n) - \sum_{j \in V} M_{ji}^k(n-1) = 0, \quad \forall k \in K(t), \forall i \in V/\{s_k,d_k\} \quad (7.3)
\]

\[
M_{ij}^k(n) \geq 0, \forall k \in K(t), \forall \{i,j\} \in E \quad (7.4)
\]

\[
T_k' \leq T_k, \forall k \in K(t). \quad (7.5)
\]

In this formulation, the objective is to find optimal traffic allocation and scheduling strategies for to-be-transferred files generated at time interval \( t \), such that the costs on inter-datacenter traffic after transmitting all files generated up to \( t \) are minimized. The optimization variables are \( \{M_{ij}^k(n)\}\forall \{i,j\} \in E, \forall k \in K(t), t \leq n \leq t + \max_k T_k \}, each of which indicates the size of a fraction of file \( k \) to be transferred along link \( \{i,j\} \) from time interval \( n \) to \( n + 1 \).

At time \( t \), we can observe that \( M_{ij}^k(t) \) represents the size of fractions that files generated at datacenter \( i \) should be divided into. At time \( n \ (t \leq n \leq t + \max_k T_k) \), \( M_{ij}^k(n) \) reflects the scheduling strategy of whether or not a fraction of a file \( k \) should be temporarily held at an intermediate datacenter \( i \). As a result, \( M_{ij}^k(n) \) over all datacenters at time intervals \([t, t + \max_k T_k]\) naturally describes both routing and scheduling strategies for file \( k \in K(t) \). Due to the assumption that flows assigned on every link for files transferred in past time intervals are known \textit{a priori}, the traffic volume to be charged on each link after transmitting files generated up to time \( t - 1 \) is known, i.e., \( X_{ij}(t - 1) \) is known at time \( t \).

Inequality (7.2) represents the link capacity constraint, which ensures that the total data volume transmitted on a link during a time interval does not exceed the link capacity. Equa-
tion (7.3) represents the traffic conservation constraint, which ensures the traffic volume going into an intermediate datacenter at each time interval equals to that going out of it at the next time interval, and the traffic volume coming from the source datacenter and going to the destination datacenter over the entire charging period is exactly the same as the file size. Inequality (7.4) ensures that fractions of any file are of non-negative sizes. Inequality (7.5) restricts that the actual transfer time of each file $T'_k$ — including both the transmission time and the waiting time at intermediate datacenters — has to be within its maximum tolerable transfer time.

However, representing $T'_k$ analytically appears to be very difficult, if not impossible. Since the actual transfer time of each file is determined by both its waiting time at intermediate datacenters and lengths of its routing paths, $T'_k$ is a function of the optimization variable $M_{ij}^k(n)$. To be exact, the actual transfer time of file $k$ equals the maximum number of hops among all paths in use for the transmission of all file $k$’s fractions from its source datacenter to its destination one, plus the number of time intervals each fraction is stored in intermediate datacenters, i.e., time intervals with $\{M_{ii}^k(n) > 0 | \sum_j M_{ji}^k(n') > 0, \forall n' < n\}$ over all datacenters, times the duration of one time interval. The result is a non-linear transfer time constraint (7.5), which increases the complexity of the problem substantially.

Since the optimization problem (7.1) resembles the traditional dynamic flow problem that is first proposed by Ford and Fulkerson in 1958 [48], their solutions shed some light on how our problem may be tackled. The dynamic flow problem is to answer the following question: what is the maximal amount of traffic that can be transferred from a source to a destination in any given number of time intervals, in a network where each link is associated with a flow capacity and a traversal time?

The model used in the dynamic flow problem is much simpler than ours. We have traffic-dependent transfer times, in that the transfer time of each file is determined by the number of hops along the path rather than traversal times on each link. We also have different transfer time constraints for each file, whereas the dynamic flow problem assumes the same constraint for all flows. Despite these discrepancies, a variant of the dynamic flow problem — taking link
costs into consideration — has a similar formulation as our optimization problem (7.1).

Ford and Fulkerson proposed to solve the dynamic flow problem by representing the time dimension through time expansion. By introducing a “virtual” copy of all nodes at each time interval, they transformed a dynamic problem over time into an equivalent static problem in a time-expanded graph [48]. Inspired by their gadget of time expansion, we seek to construct a well defined time-expanded graph for the optimization problem (7.1) as well, with the hope that the transfer time constraint for each file (7.5) can be represented analytically in the constructed time-expanded graph. If successful, our dynamic problem of minimizing costs on inter-datacenter traffic may also be solved by a corresponding static problem in the time-expanded graph. We now begin to explore such a possibility.

7.6 Problem Formulation on a Time-Expanded Graph

The key idea in our approach to solve the traffic cost minimization problem in a store-and-forward inter-datacenter network is to construct a time-expanded graph for all datacenters in the network over the considered time period. We use $G(t) = (V(t), E(t))$ to represent the time-expanded graph for inter-datacenter network $G = (V, E)$ over time intervals $[t, t + \max_k T_k]$. $G(t)$ contains one virtual copy of each node $i, \forall i \in V$ at the beginning of each time interval, which builds a time layer; it also contains a copy for all links $\{i, j\} \in E$ between each pair of time layers, with the same bandwidth capacity and cost per traffic unit.

To be rigorous, let $\{i^n|\forall i \in V, t \leq n \leq t + \max_k T_k\}$ be the nodes in $G(t)$, and define directed links $\{i^n, j^{n+1}|\forall \{i, j\} \in E, t \leq n \leq t + \max_k T_k - 1\}$ to be links in $G(t)$. For each link $\{i^n, j^{n+1}\}$ with $i \neq j$, it is associated with the available bandwidth capacity $c_{ij^n}$ and the cost per traffic unit $a_{ij}$, corresponding to link $\{i, j\}$ in the original inter-datacenter network $G$; for each link $\{i^n, i^{n+1}\}$, it is associated with bandwidth capacity $\infty$ and zero cost per traffic unit, reflecting the ability to store data at intermediate datacenters. Since the source and destination datacenters of file $k \in \mathcal{K}(t)$ in the time-expanded graph will be $s_k^t$ and $d_k^{t+T_k}$, file $k$ can be
Figure 7.3: An example inter-datacenter network and its corresponding time-expanded graph.

An illustration of constructing the time-expanded graph for an inter-datacenter network is shown in Fig. 7.3. The example network on top of the figure has 4 datacenters and 2 files to be transferred. \( a \) on each link denotes the cost per traffic unit, the link capacity \( c \) is 5 for all links, and the duration of each time interval is 1. At the beginning of time interval \( t = 3 \), File 1 is to be transferred from datacenter 2 to datacenter 4, with a size of 8 and a maximum tolerable transfer time of 4 time intervals; File 2 needs to be transferred from datacenter 1 to datacenter 4, with a size of 10 and a maximum tolerable transfer time of 2 time intervals. The figure on below shows the time-expanded graph for the example network from \( t = 3 \) to \( t + \max_k T_k = 7 \). Nodes with colours and patterns indicate the source and destination datacenters, implicitly showing the corresponding tolerable transfer times of two files.
In the constructed time-expanded graph, if we use $M^k_{ij,n}$ to represent the size of a fraction of file $k$ to be transferred from node $i^n$ to $j^{n+1}$, it corresponds to the optimal variable $M^k_{ij}(n)$ in the original dynamic optimization problem (7.1). The transfer time constraint can be reflected by the restriction that for each file $k$, $M^k_{ij,n} = 0$ for all nodes after the time layer $n = t + T_k$, after file $k$ completes its transmission. That is to say, we only consider the traffic allocation and scheduling strategy for each file in a subgraph of the time-expanded graph. The dashed square in Fig. 7.3 shows the subgraph corresponding to File 2. As a result, the dynamic traffic allocation and scheduling problem in an inter-datacenter network can be solved by a static traffic allocation problem in the constructed time-expanded graph, which has the form of:

$$\min_{M^k_{ij,n}} \sum_{\{i^n,j^{n+1}\} \in E(t)} a_{i^n,j^{n+1}} X_{ijt} I$$

s.t. \hspace{1cm} \sum_{k \in K(t)} M^k_{ij,n} \leq c_{ij,n} I, \forall \{i^n,j^{n+1}\} \in E(t) \hspace{1cm} (7.7)

$$\sum_{j^n \in V(t)} \left( M^k_{s^k_j,ij(t)} - M^k_{s^k_j,ij(t-1)} \right) = F_k,$$

$$\sum_{j^n \in V(t)} \left( M^k_{d^k_j,ij(t+T_k)} - M^k_{d^k_j,ij(t+T_k-1)} \right) = -F_k,$$

$$\sum_{j^n \in V(t)} M^k_{ij,n} - \sum_{j^n \in V(t)} M^k_{ij,(n-1)} = 0,$$

$$\forall k \in K(t), \forall i^n \in V(t) \setminus \{s^k_i, d^k_i + T_k\} \hspace{1cm} (7.8)$$

$$M^k_{ij,n} \geq 0, \forall k \in K(t), \forall \{i^n,j^{n+1}\} \in E(t) \hspace{1cm} (7.9)$$

$$M^k_{ij,n} = 0, \forall k \in K(t), \forall \{i^n,j^{n+1} | n > t + T_k\} \hspace{1cm} (7.10)$$

where $X_{ijt} = \max\{X_{ij(t-1)}, \max_{max_k T_k} \sum_{k \in K(t)} M^k_{ij,n}\}$ represents the traffic volume to be charged on link $\{i,j\}$ after transmitting files generated up to time $t$.

The formulated optimization problem is equivalent to the original problem (7.1), with its presentation in the time-expanded graph. Inequality (7.7), which shows the link capacity constraint, equations (7.8), which state the traffic conservation constraints, and inequality (7.9), which represents the non-negative traffic allocation constraint, correspond to constraints (7.2) (7.3) and (7.4) in the original optimization problem. The transfer time constraint
for each file is now represented by equation (7.10), which guarantees that the transmissions of all files are done within their corresponding maximum tolerable transfer times. It is not difficult to find out that problem (7.6) is a convex optimization problem, since both the operations of non-negative weighted sums and pairwise maximum preserve convexity [19]. With linear constraints only, classic algorithms such as subgradient projection methods and interior-point methods can be applied to solve this problem.

For the example shown in Fig. 7.3, optimal results obtained by solving the formulated static traffic allocation problem are shown in the time-expanded graph. As we can see, Instead of sending File 1 immediately when it is generated, part of it can be stored at the source datacenter 2 and the intermediate datacenter 1 for later transmission, taking advantage of the already paid link \( \{1, 4\} \) when transmitting File 2 during the first two time intervals. By doing so, the cost per time interval can be reduced to 32.67, compared to 52 if no routing and scheduling strategy is considered. If we solve the same problem by using the flow-based approach, the desired transmission rates of both files are \( r_1 = 2 \), and \( r_2 = 5 \). The optimal flow assignments for both files are shown in the left hand side of Fig. 7.3. Since File 2 will be transferred via the cheapest path \( \{D_1 \rightarrow D_4\} \), File 1 will not be able to take the cheaper path \( \{D_2 \rightarrow D_1 \rightarrow D_4\} \) due to the link capacity constraint. As a result, it will have to take the cheapest available path \( \{D_2 \rightarrow D_3 \rightarrow D_4\} \), which results in a cost per interval of 50.

As illustrated in the example, allowing temporal storage at intermediate datacenters may increase the possibility of fully utilizing links that have already been paid for. With percentile-based charging schemes, if some traffic has already been transmitted along a link during a time interval, the same cost will be incurred even if the link is idled or used for transmitting less traffic in later time intervals. With the store-and-forward approach, traffic is “time-shifted” to later time intervals as much as possible, with the objective of minimizing costs by using links that have already been paid for, perhaps due to the transmission of other files.
7.7 The Time Expansion Approach

In the Postcard problem formulation, the construction of a time-expanded graph is used to solve the traffic cost minimization problem in a store-and-forward inter-datacenter network. The time expansion approach, however, can be applied to ease the formulation of a class of similar problems.

For example, an interesting problem is to utilize leftover bandwidth during non-peak hours for the benefit of bulk “background” traffic, such as backups and data migration, first proposed by Laoutairs et al. [49]. Since the cloud provider only use the leftover bandwidth, which is assumed to be covered by the already paid traffic volume generated during peak hours, there is no longer a concern on traffic costs. The problem is to find the optimal traffic allocation and scheduling strategy, such that as many bulk files as possible can be transferred within a maximum transfer time constraint for each of them.

Using the time expansion approach, with the same optimization variables \( \{M_{ijn}^k | j^n j^{n+1} \in E(t), k \in K(t), t \leq n \leq t + \max_k T_k \} \) indicating the size of a fraction of file \( k \) to be transferred from node \( i^n \) to \( j^{n+1} \), the corresponding optimization problem can be formulated by replacing the objective function in problem (7.6) with the following one:

\[
\max_{M_{ijn}^k} \sum_{n=1}^{t + \max_k T_k} \sum_{\{i^n j^{n+1}\} \in E(t), k \in K(t)} M_{ijn}^k, \quad (7.11)
\]

which indicates the maximum of the transmitted traffic volume up to time interval \( t \). With all constraints remaining the same, problem (7.11) is a linear optimization problem that can be efficiently solved by linear programming solvers. Laoutairs et al. [49] has also proposed a very similar time expansion approach to solve the problem in a store-and-forward inter-datacenter network. However, they only consider the case of making an optimal decision for a single file, while we consider the case of making optimal decisions for transmitting multiple files with different transfer time constraints, which is more generic and much more challenging.

Another possible problem faced by a cloud provider might be: given a certain budget on costs incurred by inter-datacenter traffic, what is the maximum number of files that a cloud
provider can transfer among geographically distributed datacenters up to a certain time interval $t$? This problem makes more sense during the peak traffic hours, when more files are waiting to be transferred, compared to the limited capacity restricted by a cloud provider’s budget. The cloud provider would wish to satisfy as many inter-datacenter file transfer requests as possible to provide a more competitive service, as long as the incurred traffic costs are within its budget.

Such problem can also be easily formulated by using our time expansion approach. With the same objective function (7.11), this problem takes the form by adding an additional constraint on traffic costs, $\sum_{(i^n,j^{n+1}) \in \mathcal{E}(t)} a_{i^n,j^{n+1}} X_{ijt} I \leq B$, where $B$ is the budget on traffic costs. The resulted problem is also a convex optimization problem with a linear objective function and a convex inequality constraint [19].

7.8 Performance Evaluation

We dedicate this section to investigate how Postcard performs in reducing costs on inter-datacenter traffic. Specifically, we seek to investigate its advantages and drawbacks compared to the flow-based approach.

The evaluation of Postcard is based on our implementation in a time-slotted simulator, using the fmincon function provided by MATLAB. We simulate in a system with 20 datacenters, forming a complete graph. The cost per traffic unit on each link is set to be uniformly random within $[1, 10]$. In each time interval, the number of files to be transferred is uniformly random between $[1, 20]$, each of which has a uniformly random size of $[10, 100]$ GB. The source and destination datacenters of each file are chosen from the datacenter set uniformly. We conduct our simulations 10 times, each lasts for 100 time slots. For comparisons, we also implement the flow-based approach introduced in Chapter 6 with the same evaluation setup. Still, we assume that the 100-th percentile charging scheme is in use in our simulation.

We consider four different simulation settings. The first two settings are with sufficient link capacities, i.e. $c_{ij} = 100$ GB/\(t\) for all $\{i,j\} \in \mathcal{E}$, one of which has more urgent files
(\text{max}_k T_k = 3), whereas the other has more delay tolerant files (\text{max}_k T_k = 8). The last two settings are with limited link capacities, \textit{i.e.} \( c_{ij} = 30 \text{ GB/\( t \) for all \( \{i, j\} \in \mathcal{E} \), and again, one has more urgent files and the other has more delay tolerant files. Average costs per time interval on inter-datacenter traffic and their 95\% confidence intervals with both \textit{Postcard} and the flow-based approach in each setting are shown in Fig. 7.4, Fig. 7.5, Fig. 7.6, and Fig. 7.7, respectively. The results reveal that the flow-based approach outperforms \textit{Postcard} significantly when there are sufficient link capacities, while \textit{Postcard} demonstrates superior performance when link capacities are throttled. We also discover that \textit{Postcard} leads to lower costs when there are more delay tolerant files in the system, with either sufficient or limited link capacities.

The reason is that, store-and-forward incurs bursty traffic on relay paths compared to the flow-based approach. Take the network in Fig. 7.3 as an example. If we wish to transfer a file with a size of 10 from datacenter D_2 to D_4 in two time intervals via the path \{D_2 \rightarrow D_1 \rightarrow D_4\}, which is the cheapest path between these two datacenters, the maximum traffic volume per time interval with the flow-based approach is 5, while that with \textit{Postcard} will be 10. Instead of forwarding the file immediately after receiving the first byte in the flow-based approach, datacenter D_1 has to wait until it has fully received the entire file in \textit{Postcard}. This results in bursty traffic on both links, and hence, higher costs with percentile-based charging schemes.
However, when link capacities are limited, cheaper links may be occupied by urgent traffic for some time intervals. As a result, they are unavailable with the flow-based approach, even if they will idle for most subsequent time intervals, e.g., link \{1, 4\} in the example shown in Fig. 7.3. On the contrary, store-and-forward provides possibilities to fully utilize those cheaper links by taking advantage of files with longer tolerable transfer times. Shown in our simulations, Postcard exhibits better performance when link capacities are limited. Since the more delay tolerant files in the network, the more opportunities exist for the “time-shifting” of inter-datacenter traffic, costs are reduced with Postcard when there are more delay tolerant files in the system, with either sufficient or limited link capacities.

7.9 Summary

In this chapter, we presented Postcard, an online optimization problem that is formulated to minimize operational costs on inter-datacenter traffic, with the ability for intermediate nodes to store data and forward them at a later time. Different from Jetway, the highlight of Postcard is our proposed way to simplify the cost optimization problem with store-and-forward in the general setting: by restricting data transmission to a time-slotted model, it becomes feasible to
formulate the problem by modelling the inter-datacenter network with a time-expanded graph. By solving the optimization problem with convex optimization solvers, Postcard allows us to compare the cost of allowing store-and-forward on intermediate datacenters with that of Jetway in our simulations. We have observed that Postcard exhibits better performance when link capacities are limited, or when the data to be transferred among datacenters are more delay tolerant.
Chapter 8

Concluding Remarks

8.1 Conclusions

The overarching research objective in this dissertation is to identify open challenges and design practical solutions for mobile applications and the cloud to work harmoniously together, so that users will enjoy the best possible experience. In this dissertation, we have designed new algorithms and protocols to be used on both mobile devices and in the cloud, such that application performance can be improved to provide better user experiences, yet within the budgetary constraints of available resources and operational costs.

We started from an examination of the nature of mobile applications. We discovered that it is customary for mobile users to use multi-touch gestures to create and consume content in mobile applications, both interactively and collaboratively with other users. To support such collaboration among multiple users in real time, we proposed that gestures, rather than application-specific states, should be streamed from one user to all participating users in a broadcast session. Once received, gestures can be recognized and rendered in real time by a live instance of the same application at each of the participating users.

Different from traditional media streams, gesture steams typically incur low yet bursty bit rates, but packet losses are not tolerable since each lost packet will severely affect the accuracy
of the gesture recognizer at the destination. To address these challenges and realize the vision of streaming gestures, we presented our design and implementation of GestureFlow in Chapter 3. GestureFlow is our new gesture streaming protocol that uses inter-session network coding to achieve the objective of minimizing gesture recognizing delays. It used random network coding with multiple paths, allowing for recoding across multiple concurrent sessions. Our real world experiments with MusicScore, a gesture-intensive music composition application, have validated the effectiveness of GestureFlow in supporting collaborative interaction among multiple users with satisfactory gesture recognizing delays.

Although gesture streams are delay-sensitive, they typically involve low bit rates. On the other hand, there exists another category of mobile applications that demands both high bandwidth and low latencies: multi-party video conferencing. Video conferencing is well known to be demanding: a large amount of bandwidth is needed to support high-definition video, yet stringent delay constraints need to be imposed to support interactive conversations. Given that inter-datacenter networks in the cloud may be able to offer higher capacities, we begin to explore the possibility of supporting multi-party conferencing by using a cloud service. We use video conferencing as an example to show how cloud can be utilized to improve the performance of mobile applications.

In Chapter 4, we have studied the feasibility of supporting video conferencing with a cloud service. The basic idea is to route video conferencing traffic over the inter-datacenter network in the cloud, enjoying its high-capacity links. We presented Airlift, a new cloud service that delivers packets in video conferences to their respective destination datacenters, with the objective of maximizing the total throughput across all conferences, yet without violating end-to-end delay constraints. In order to simplify its protocol design, we proposed to use intra-session network coding and the concept of conceptual flows, so that the optimization problem can be conveniently formulated as a linear program. To bridge theory with practice, Airlift has been implemented and deployed over the Amazon EC2 cloud, where it has been shown to deliver a multi-fold performance advantage over the best possible peer-to-peer solution in the literature.
We then take the cloud provider’s point of view, and try to see what challenges may be present if the cloud is used to improve the performance of mobile applications. We first realize that by providing their resources in the form of virtual machines, it is certainly the cloud providers’ wish to fully utilize their available resources. With virtualization, datacenters in the cloud have provided a shared pool of computation, storage, and bandwidth resources, to be used by cloud applications when the need arises. Due to varied demand from cloud applications, cloud servers may become overloaded when highly bursty requests are encountered, since several virtual machines (VMs) placed on the same server may reach their peak period in demand at the same time. An intuitive solution to such a problem would be to migrate VMs away from overloaded servers to under-utilized ones. Yet, such migration should be planned with care, since servers supply resources in multiple dimensions of storage, bandwidth, and CPU cycles, and resource utilization may become severely unbalanced across different dimensions.

Our focus in Chapter 5 is to fully utilize resources in dimensions of storage, bandwidth and CPU computing cycles in video streaming datacenters, by migrating VMs live among servers when they are overloaded. Inspired by the power of markets in arbitrating decisions of both buyers and sellers in a decentralized fashion, we proposed a new VM migration algorithm based on Nash bargaining games. Our solution relates the entire datacenter to a bargaining market. VMs are considered as commodities in such a market, and by maximizing the joint profit in the Nash bargaining solution, resource utilization is maximized as a result.

Beyond optimizing resource utilization in a single datacenter, we are also interested in minimizing the costs of inter-datacenter traffic. In order to stream videos to end users, it is now the norm for video streaming service providers to use the services offered by cloud providers. It is typical for cloud providers to deploy a number of datacenters inter-connected by high-capacity links, spanning different geographical regions. Recent research reveals that traffic costs amount to a large portion of operational costs incurred to a cloud provider, and among those, inter-datacenter traffic occupies close to half of the traffic going through a datacenter’s
egress router. For a cloud provider to reduce its operational costs efficiently, it is desirable that the cost incurred by the inter-datacenter traffic be minimized. Inter-datacenter traffic is typically charged by ISPs based on a percentile-based charging model, with which cloud providers pay based on the $q$-th percentile of traffic volumes measured in a short time interval, over a number of such intervals in a charging period. Such a model implies that, if traffic has already been generated during one time interval, up to the same volume of traffic may be carried free of charge in subsequent time intervals.

In Chapter 6, we have presented Jetway, designed to minimize costs on inter-datacenter video traffic by splitting and routing video flows over multiple multi-hop paths. Jetway takes full advantage of our key observation that some of the traffic volumes can be transferred free of charge, while the desired transmission rates of video flows remain satisfied. Our work is motivated by the research question: can such inter-datacenter traffic costs be minimized, by carefully choosing paths and assigning flow rates along each path? In the design of Jetway, we have designed a new set of combinatorial algorithms to minimize such costs, by splitting and routing flows optimally in the inter-datacenter network. To evaluate its performance, we have implemented Jetway in the Amazon EC2 cloud, and our experimental results validate that Jetway is able to reduce costs on inter-datacenter traffic effectively.

In Chapter 7, we have also designed an online optimization problem, called Postcard, to minimize operational costs on inter-datacenter traffic with the ability for intermediate nodes to store data and forward them at a later time. Different from Jetway, the highlight of Postcard is our proposed way to simplify the cost optimization problem with store-and-forward in the general setting: by restricting data transmission to a time-slotted model, it becomes feasible to formulate the problem by modelling the inter-datacenter network with a time-expanded graph. By solving the optimization problem with convex optimization solvers, Postcard allows us to compare the cost of allowing store-and-forward on intermediate datacenters with that of Jetway in our simulations. We have observed that Postcard exhibits better performance when link capacities are limited, or when the data to be transferred among datacenters are more delay
tolerant.

8.2 Future Work

The emerging use of Siri on iOS devices and Google Voice Search on the latest Android devices has clearly indicated an industry trend to seamlessly integrate cloud services with mobile applications: in order to provide the best possible user experience, a small amount of data can be transmitted to the cloud to take advantage of its abundant data and computational power. A large number of research questions remains open in the area of mobile cloud computing, as we optimize the use of cloud computing resources to scale up to millions of mobile users.

The substantial performance advantage in our experimental experiences with Airlift, shown in Chapter 4 of this dissertation, has spoken volumes on the achievable performance over inter-datacenter networks in the cloud. Yet, the potential of inter-datacenter networks has largely remained untapped. We believe that abundant capacities in these inter-datacenter networks should be better utilized, by fairly allocating and “slicing” network bandwidth using network virtualization, and by pricing such virtualized network “slices” accordingly to accommodate resource-demanding applications that are willing to pay for premium quality. Such virtualized networks should naturally support not only unicast communication, but also multicast and broadcast, since network virtualization is achieved at the application layer, without the need to change traditional transport protocols that are largely best effort. With well-designed resource scheduling disciplines, a range of fairness policies and statistical performance guarantees may be supported. If application-layer network virtualization can be implemented in reality, “slices” of the network can be coalesced, split, and even traded as commodities among participating applications in economic markets.
Bibliography


