ON THE BENEFITS OF NETWORK CODING IN MULTI-PARTY VIDEO CONFERENCING

by

Yiwei Pu

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright 2013 by Yiwei Pu
Abstract

On the Benefits of Network Coding in Multi-party Video Conferencing

Yiwei Pu
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2013

The widespread use of personal multimedia-rich devices and multi-party video conferencing have made face-to-face communication among multiple users a promising feature. This thesis presents a multi-party conferencing solution with network coding naturally embedded. Our fundamental goal is to study if network coding brings benefits to multi-party conferencing. In this thesis, we first review an existing network coded solution for multi-party conferencing. Then, this solution is evaluated in our framework of evaluating a new transmission protocol for multi-party conferencing. Also, an investigation is set up to dive into the bottlenecks of this network coded solution. Next, an improved solution targeting conferencing services is proposed by tackling the bottlenecks of the existing solution. Based on our experiment results, it is found that network coding does bring benefits in the context of multi-party conferencing.
Dedication

To my parents
Acknowledgements

I would like to first express my sincere gratitude to my thesis supervisor, Professor Baochun Li, for the invaluable help he has provided to me throughout my time as a Master’s student of his. His patient guidance and encouragement not only greatly aided me in the completion of this thesis, but also significantly helped me build up my understanding in the area of computer networks and cultivate my skills of research.

I would also like to thank all the members in iQua research group at University of Toronto, with whom I was involved in frequent discussions which eventually benefited my thesis work. Completing this work would have been all the more difficult were it not for the support and friendship provided by other members of iQua research group.

I must express my gratitude to my mother and father for their continued support and encouragement. They accompanied me throughout my Master’s studies at University of Toronto as a student from overseas, and experienced all the ups and downs of my research. Without their support, I would have never been able to focus on my research so much. A big thank you to them!
## Contents

1 Introduction 1  
   1.1 Existing Solutions for Video Conferencing 2  
   1.2 Conferencing with Network Coding 3  
   1.3 Contributions 5  

2 Related Work 6  
   2.1 Multicasting with Network Coding 6  
   2.2 Conferencing in the Cloud 7  
   2.3 Choosing TCP as A Benchmark 8  
   2.4 Bellini: A Platform for Evaluation 8  
   2.5 Summary 9  

3 Conferencing with Network Coding 10  
   3.1 Path Selection 10  
   3.2 Coding Strategy 11  
   3.3 Packet Redirection 15  
   3.4 Rate Control 16  
   3.5 Summary 16  

4 Evaluation Framework 17  
   4.1 Criteria 17  
   4.2 Scenario 18  
   4.3 Benchmarks 19  
   4.4 Summary 20  

5 Investigation 21  
   5.1 Evaluation 21  
   5.2 Investigation 23  
      5.2.1 End-to-end Traffic Decomposition 23  
      5.2.2 End-to-end Delay Decomposition 23  
   5.3 Analysis 25  
   5.4 Summary 26
6 Improvements

6.1 A Naive Time-out-based Approach ........................................... 28
6.2 A Batch-based Acknowledgment Mechanism ................................ 29
6.3 Distinguishing Valid Acknowledgments ...................................... 32
6.4 Evaluation of the ACK/ARQ Mechanism ..................................... 34
6.5 Load Balancing ....................................................................... 35
6.6 Evaluation of the Load Balancing Mechanism ............................... 39
6.7 Summary ............................................................................... 40

7 Concluding Remarks ................................................................... 41

Bibliography ................................................................................. 42
List of Tables

1.1 Priority of requirements in different online services ........................................ 1
4.1 Node configuration ............................................................................................... 19
5.1 Fairness-related metrics in the data center network ............................................ 22
6.1 Topology of the one-to-many experiment ........................................................... 39
6.2 Topology of the many-to-many experiment ....................................................... 39
List of Figures

1.1 Illustration for the benefits of network coding in terms of throughput: (a) Butterfly network (b) Multicast with routing (c) Multicast with network coding ........................................ 4

3.1 Illustration for the procedure of path selection: (a) Given a network with four nodes, where link A-C is blocked and delay over each three-hop connection is intolerable; (b) The set of all paths is split and narrowed down to multiple eligible path sets for each destination and the source; (c) Two multicast trees are built up with eligible paths to cover all destinations. ................................................................. 11

3.2 Illustration for converting eligible paths to multicast trees .......................................................... 12

3.3 Illustration for using network coding in practice ............................................................................ 13

3.4 A comparison between generation-based and sliding-window based network coded solutions: (a) Traditional sliding-window based solution, where source nodes keep sending off coded blocks while waiting for acknowledgments from its destinations. Therefore, redundant coded packets are generated and sent off; (b) Pipeline style generation-based solution, where source nodes moving onto the next batch of original packets, instead of waiting for the completion of previous batches. .................................................. 14

3.5 Three basic operations when multiple conceptual flows passing through an intermediate node: (a) Replication, (b) Split, (c) Mix ................................................................. 15

4.1 The scenario for evaluation is a network composed of four instances in the Amazon Elastic Computing Cloud (EC2). This scenario is set up to mimic real-time traffic among data centers in different regions. ................................................................. 19

5.1 Maximum achievable rate at different session sources ................................................................. 21

5.2 End-to-end delay in the cloud when fixed rates are assigned ...................................................... 22

5.3 Investigation with traffic decomposition in different scenarios .................................................. 24

5.4 Investigation with delay decomposition in different scenarios .................................................. 24

5.5 An illustration for disordered packets at the receiver-side due to the existence of multi-paths 26

5.6 False retransmission is triggered due to the existence of multi-path between the source and the destination ................................................................. 27

6.1 Illustration for the time-out based solution .................................................................................... 29

6.2 Illustration for the basic idea of batch-based solution .................................................................. 30

6.3 Illustration for the batch-based mechanism to ensure communication reliability ........................ 31

6.4 Example of outdated acknowledgment ....................................................................................... 32
6.5 State Transition Diagram of one network coded generation ......................... 33
6.6 Illustration of Scenarios for comparison ................................................. 34
6.7 Comparison on achievable throughput between TCP-based and batch-based solutions ...... 34
6.8 Comparison on the bandwidth redundancy between naive and improved retransmission solutions ................................................................. 34
6.9 Comparison on achievable rates between the direct TCP solution and the improved network coded solution in a one-to-many scenario .................. 40
6.10 Comparison on sustainable source rates of different sessions between the direct TCP solution and the improved network coded solution .................. 40
Chapter 1

Introduction

There is no doubt that multi-party video conferencing is a promising service. It enables people to enjoy natural face-to-face chatting experience with no geographic limitations. With recent advances of consumer-level multimedia devices in the past decades, providing high-quality multi-party video conferencing has become a blooming market catching everyone’s attention. However, though many video conferencing products have been available and related technologies have matured, it is still non-trivial to provide a high-quality multi-party video conference, satisfying a comprehensive set of requirements.

Table 1.1 briefly summarizes the requirements of several mainstream online services. Unlike traditional point-to-point multimedia transferring services, multi-party video conferencing raises more requirements on network bandwidth, in order to carry high definition video, audio and other multimedia streams. Besides, conferencing clearly differentiates itself from other online services by stringent delay requirements, and extra demand on synchronization and fairness.

<table>
<thead>
<tr>
<th>Services</th>
<th>Rate</th>
<th>Delay</th>
<th>Fairness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Variation</td>
<td>Average</td>
</tr>
<tr>
<td>Web browsing</td>
<td>-</td>
<td>△</td>
<td>-</td>
</tr>
<tr>
<td>Multimedia sharing</td>
<td>△</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Video-on-demand</td>
<td>○</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Live video streaming</td>
<td>○</td>
<td>○</td>
<td>△</td>
</tr>
<tr>
<td>Conferencing/Gaming</td>
<td>○</td>
<td>○</td>
<td>△</td>
</tr>
</tbody>
</table>

N.B. △: critical ○: important -: tolerable

To fulfill these Quality of Service requirements, a significant amount of efforts, from both industrial designers and academic researchers, have been devoted to designing and studying solutions for multi-party conferencing services. Among the entire conferencing system, the transport protocol is without a doubt one of the most important components. It determines the overall efficiency of utilizing accessible network resources. Therefore, a substantial amount of work has been proposed to design efficient solutions for transport protocols. This thesis presents our effort to find a better solution for an application layer protocol to achieve reliable, ordered and high-speed multicast services with end-to-end delay constraints.
1.1 Existing Solutions for Video Conferencing

To deliver video streams among multiple participants, most video conferencing products\cite{31} either send live multimedia streams via direct point-to-point connections, or rent a set of central servers for relaying as in Microsoft Office Live Meeting. The client-server approach ensures that the entire upload bandwidth of each user can be used for delivering a video stream. However, it places a heavy CPU and network bandwidth burden on the central servers. Therefore, a large amount of costs is needed to purchase and maintain these high-end equipments. In contrast, an alternative is to use application-layer multicast (ALM) in peer-to-peer (P2P) networks, as researchers proposed in \cite{14, 9, 33, 25}. These approaches are proposed to improve the quality of conferencing services by sharing bandwidth among different participants.

The end system multicast \cite{14} is one of the first work proposing solutions to share bandwidth between users. This work introduces a tree-based multicast approach to set up an overlay multicast architecture for multimedia delivery. Following this concept, SplitStream \cite{8} builds multiple overlay multicast trees to enable a more robust and fair system, where all nodes provide roughly the same contribution. Similarly, Coopnet \cite{33} forms either random or deterministic node-disjoint trees and adopts Multiple Description Coding layers \cite{21} to optimize the quality of service. From theoretical aspects, creating and maintaining shortest-path multicast trees is the optimal approach for real-time multimedia delivery, since the minimal delay is achieved. However, such tree-based architecture is usually bandwidth-limited, and the transfer rate is no more than the capacity of the bottleneck link along the multicast tree. Besides, such "bottleneck" links are difficult to be found in real-world networks, since multiple dis-joint overlay data flows are likely to share the same underlay connection.

As an alternative to tree-based solutions, a number of mesh-based peer-to-peer architectures have also been proposed. Mesh-based architectures, like parallel downloads \cite{34}, can substantially improve the efficiency of using network resources by setting up additional connections, and hence, additional bandwidth and better opportunities of sharing desired contents. But, one major drawback of mesh cooperative approaches is the overhead brought by collision. Since no multicast tree is predefined, each node makes local decisions on delivering data blocks. Therefore, due to the lack of global scheduling, it is highly possible that the same block is transmitted over competing paths, and thus, network resources are under-utilized and the end-to-end performance is degraded. It is known that BitTorrent \cite{1} is a most popular application based on such a cooperative architecture. In spite of its excellent performance of speeding-up file downloads, servers are highly possible to be overloaded, when clients are close to finishing their download, and attempt to obtain the missing blocks from the server.

In short, compared with a centralized network architecture, these distributed approaches achieve higher efficiency in using the network resources by collaboration and decentralizing the network loads. However, these proposals are originally designed for multimedia content sharing services, with little concern on the low-latency constraint of multi-party conferencing.

Besides, most of the existing works on video conferencing do not take the diversity of current network architectures into account. In addition to the network organized by ordinary endpoint terminals, an increasing number of data centers have also been globally observed. In the beginning of 2012, Apple, Amazon, Facebook, Google, and Microsoft all announced their bold new data center building plans. With these powerful data centers, network companies, either giants like Google and Microsoft or startups like Dropbox, have chosen to operate their network services on data centers, which essentially form a big cloud in the Internet. Unlike ordinary terminals, the instances provided by these data centers
typically provides superior computing capabilities, service reliability, energy efficiency, and better cost-effectiveness. In addition, data centers are usually well inter-connected by dedicated fiber-optic networks deployed by the data center operators, or high-speed backbone rented from multiple Internet Service Providers (ISP). Due to these appealing advantages, deploying conferencing services within the clouds will be an alternative for service providers, in addition to current favors of purchasing expensive high-end equipments or relying on users’ personal properties. For example, OnLive [2] uses co-located data centers to provide a novel interactive on-demand cloud gaming platform, which renders high-quality gaming scenes according to received user’s interactions using high-end GPU in the cloud, and streams produced videos to a player’s low-end computer or portable devices with OnLive client. Therefore, when a solution for video conferencing is proposed, its performance in cloud-based networks is also a key concern for judgment.

1.2 Conferencing with Network Coding

To support the conferencing services, we believe that a high-quality protocol design must achieve the full potential of the multi-path communication paradigm, and must guarantee the stringent requirements of low end-to-end delays, with the highest possible source coding rates that can be supported by dynamic network conditions over arbitrary network topologies over the Internet. To maximize the achievable multicast rate while minimizing streaming delays with guaranteed reliability, we believe the use of network coding is promising, as it is able to take full advantage of the all-to-all broadcast nature of multi-party video conference. Since its inception in information theory [26], network coding shows its potentials in the following aspects:

The first advantage of network coding is to improve throughput. Such advantage can be demonstrated by the well-known Butterfly Network as shown in Figure 1.1, where node $s$ is the source, and nodes $r_1$ and $r_2$ are the two receivers. All edges in the figure have capacity 1, which means that each edge can transmit only 1 unit data (bit) per unit time (second). Source $s$ has two bits, $b_1$ and $b_2$, to multicast to both $r_1$ and $r_2$. In Figure 1.1, traditional multicast without network coding is shown. Bit $b_1$ is able to reach $r_1$ in two seconds. Also, bit $b_2$ can reach $r_2$ in two seconds. When node $c$ receives both bits, it forwards them sequentially. Without loss of generality, we suppose it forwards $b_1$ first. Then $r_1$ is able to receive both bits in 4 seconds and $r_2$ receives both bits in 5 seconds. Now take network coding on link $c-d$ into consideration, as shown in Figure 1.1. When node $c$ receives both bits, it first mixes them by performing an exclusive OR(XOR) operation on the bits. Then it sends the mixed bit $b$ to node $d$. When nodes $r_1$ or $r_2$ receive the mixed bit, they can recover the original bits $b_1$ and $b_2$ by XORing the mixed bit and the other received bit. In this way, all the transmissions can be completed in 4 seconds.

Second, network coding also tends to achieve low latency. To some extent, network coding can be seen as an extension of erasure code, like digital fountain [7, 6]. Endpoint users are able to reconstruct the original symbols, as long as adequate coded blocks have been received, no matter which paths the blocks come from. Therefore, it is able to deal with packet loss, and to collaborate with multi-path. Furthermore, such a property is helpful to reduce the latency. For example, when a packet is lost during the transmission over a TCP connection, the receiver has to reply the exact sequence number of the lost packet to the source node. Also, the source node has to choose the shortest downstream path for retransmission. But, in a practical system, it is not easy to find the ”shortest” path, especially when the network is dynamic and large-scale.
When network coding is adopted, such questions are avoided. If one block is lost during the transmission, the destination just keep receiving till an adequate number of blocks are collected. Since all received blocks are equally useful for decoding, the recovery process is finished as soon as the expected received number is reached. Thus, the "best" path is automatically selected, and thus the end-to-end latency is intuitively reduced.

Third, network coding is helpful to reduce the complexity of routing problems in a multicasting system. It is known that the high utilization of a given topology can also be achieved using multiple edge-disjoint distribution trees (especially in the case where all nodes are receivers). In fact, several schemes (e.g. Splitstream, CoopNet) have proposed to utilize multiple multicast trees to deliver data streams from the source to its all receivers. Steiner tree packing is a well-known technique, which is often used to maximize the video flow rate from a single source to other participants in a video conference. This paradigm does help to improve end-to-end throughput, but, computing the strategies to achieve optimal throughput using multiple trees has been shown to be NP-complete [10, 11]. Therefore, only depth-1 and depth-2 trees are packed in existing works [11, 28].

However, when network coding is used, such problem can be greatly simplified. Li et al. [27] depicts that, with the idea of conceptual flows, network coding has shown its strong power of resolving conflicts competing for bandwidth resources in bottleneck links. In other word, thanks to the advantages of network coding, the problem of maximizing throughput with delay constraints can be formulated as a linear program, and then be easily solved by using a standard LP solver.

It has widely been accepted that, with network coding, the cut-set bound of information flow rate in a multicast communication session can be achieved. Also, network coding is useful for protection from packet losses and link failures. By allowing the sharing of network resources among different flows, network coding can improve resource usage. Successful examples have also been reported by
experimental simulations as well as some production systems, in the fields of file sharing [20] and on-demand video streaming [37]. However, in the presence of strict constraints of low latency and high bandwidth for smooth live chatting experience, little was known about the benefits of network coding from the literature. Such lack of results probably comes from its complexity in both theoretical analysis and practical implementation. Also, it is still debatable that if network coding is beneficial in real-world systems. We cannot help wondering what are the potential benefits brought by network coding, how to integrate it into real-world multicast services with different network architectures, and whether it is possible to bring a new wave of innovation, like what peer-to-peer paradigm did. Keeping these questions in mind, this thesis intends to explore the benefits of network coding in the context of multi-party video conferencing in real-world scenarios.

1.3 Contributions

Although network coding sounds beneficial for multi-party video conferencing, the paradigm of network coding is still a controversial technique in real-world networks, due to its complexity. However, as far as we know, no existing literature has made a comparison between network coding and the TCP-based approach, which is the current favor of data transmission in our daily Internet. Therefore, this work intends to evaluate the benefits of network coding in the context of multi-party conferencing within large-scale data centers, by making comparison between TCP-based approaches.

Due to the diversity of existing solutions for hosting conferences, making comparison between a widely-accepted approach for data transmission in practical network scenarios is helpful to convince readers of network coding’s potential benefits. Our ultimate goal is to provide a benchmark for researchers studying network coding in real-world networks; and to provide suggestions for system/application designers, who plan to deploy new services over the Internet, and making decisions between traditional TCP-based approaches and the new paradigm of network coding.

Our contributions in this thesis is three-fold. First, a framework to evaluate protocols in the context of multi-party video conferencing is proposed with multiple criteria reflecting desired requirements of service. Second, experiments are set up to make comparison between an existing network-coding-based solution and its TCP-based counterpart. Also, the experiment results are analyzed based on designed framework to evaluate their performance and to figure out the major bottlenecks. Third, a modified protocol is proposed to tackle the major problem which is found in the previous step. Also, the improved protocol is compared with TCP-based solutions to evaluate its performance.

The remainder of the thesis is organized as follows. To begin with, Chapter 2 reviews the literature related to deploying a conferencing system with network coded solutions. Then, Chapter 3 reviews an existing generation-based network coded solution as an alternative to traditional sliding-window based solutions. Next, Chapter 4 introduces our framework of evaluation in the context of multi-party video conferencing, followed by experiment results and an investigation on bottlenecks in Chapter 5. To tackle the bottleneck against better performance, Chapter 6 details our improved solution with experimental results. Finally, this thesis is concluded in Chapter 7.
Chapter 2

Related Work

2.1 Multicasting with Network Coding

Network coding is a new paradigm allowing nodes inside networks to send mixed data across different flows instead of simply forwarding incoming packets. Since it was first introduced by Ahlswede et al. [5] with examples demonstrating its benefits in terms of throughput improvements, a substantial amount of works have been proposed to study its potential benefits. Koetter et al. [24] proposed an algebraic framework for network coding. They also showed that network coding is able to achieve the min-cut bound for a single-session multicast in directed networks. Next, Li et al. [26] proved that linear codes usually suffice in achieving maximum rate for multicast transmission. Since Ho et al. presented a random linear network coding approach with no compromise on throughput, possibilities of easy-to-implement solutions were put on the horizon. Meanwhile, Li’s observation et al. [27] supported the potential of network coding in practical use. It proved that network coding could substantially decrease the complexity of the finding best routing strategies to achieve optimal throughput, while this problem has been shown as a NP-complete problem with traditional Steiner tree packing solutions [10, 11]. Besides the benefits of improving achievable throughput, Wu et al. [38] also pointed out network coding’s inherent ability of dealing with network dynamics and uncertainty, like random packet loss or link failure. Koetter et al. [24] discussed robust networking and analyzed the resilience of network coding. Also, Lun et al. [32] theoretically analyzed the benefit of network coding on a directed acyclic graph with lossy links. In terms of potential applications, Acedanski et al. [4] showed that network coding could entitle a peer to connect to fewer other peers to retrieve an entire file. Similar successful examples have also been reported in the fields of file sharing [20, 39], on-demand video streaming [37, 13] and distributed storage [16, 17, 15]. But most of these investigation works were based on theoretical assumptions and analysis, many practical problems or engineering concerns were not taken into account.

Besides analysis from theoretical aspects, the idea of network coding is also brought into practice. Ho et al. extended their idea of random linear network coding to a distributed solution [22]. Katti et al. [23] put forth the idea of local opportunistic coding to present another practical implementation of network coded system. Chou et al. [13] proposed a unicast system with embedded coefficients in packet header, and an accumulative acknowledgment (ACK) mechanism to ensure transmission reliably. As for rate and congestion control, Sundarajan et al. [35] modified the ACK mechanism of classic sliding-window-based Transmission Control Protocol (TCP) to fit network coding in unicasting scenarios. Lin
et al. [29] proposed an opportunistic routing protocol to transmit useful coded packets during ACK propagation. Feng et al. [19] depicted another window-based broadcast protocol targeting multimedia streaming services. On the other hand, Chen et al. [10] proposed an alternative of equation-based rate control algorithm within a utility optimization framework. Also, Feng et al. [18] proposed a solution to deliver multimedia streams among data centers using a migrated approach of equation-based rate control algorithm and batch-based network coding.

With the idea of mixing, network coding manages to improve the efficiency of sharing network resources and boost robustness of transmission over lossy links. However, due to the lack of widespread implementation, this new paradigm is still a controversial technique in practice. In recent studies, Chiu et al. [12] demonstrated that network coding cannot theoretically further increase throughput in P2P networks. It has also been shown that the coding complexity escalates when an increasing number of blocks are used in network coding, and that even with high-performance processors, coding more than a few hundred blocks may not be computationally feasible [36]. In short, the technique of network coding is still controversial in terms of practical use. Therefore, this thesis intends to present an in-depth study on this topic, and differentiate itself from other related works by narrowing its target on a specific scenario of multi-party video conferencing in data center networks, and setting up comparisons with the classic Transmission Control Protocol (TCP), which is rarely discussed in other works.

### 2.2 Conferencing in the Cloud

In the debate on the potential of network coding, a substantial amount of works have been proposed with different arguments. Among existing literature, Feng et al.’s proposal [18] is a most convincing proof for multicasting with network coding. Also, this is the closest work related to ours. This work proposed an approach for delivering multimedia streams among multiple instances within large scale data centers. Also, they demonstrated the superiority of their proposal over another successful peer-to-peer based solution. To some extent, our work is an extension of Feng et al.’s solution by diving into bottlenecks against improving performance in real-world networks. However, the work introduced in this thesis is not a trivial enhancement. Compared with the original design, the contribution of this work mainly contains three parts. First, different network architectures are studied. In Airlift, a combined network architecture with both ordinal terminal endpoints and instances in data center networks is considered. However, this work is focused on networks which are composed of instances in data center networks only. Due to the difference between these two types of network, different performance could be achieved. Second, the original work was focused on verifying its superiority over an existing P2P-based solution, but no efforts were taken to dive into the challenges that motivate further improvements. Inspired by their approach, we reproduced the original solutions, inserted additional implementation to probe extra metrics, analyzed the collected experiment data, and finally figured out the main bottleneck against further improvements. Thus, we could propose a solution to improve its performance in real-world scenarios. Third, unlike the existing work choosing a P2P-based solution as a benchmark, we come up with an idea of evaluating network coding by comparing its performance with the Transmission Control Protocol (TCP), which is the current transport protocol in the Internet. The main idea of TCP is to use ACKs from receivers to trigger retransmissions and the congestion control loop. Based on our knowledge, TCP is excluded as a candidate of benchmark when network coding is to be evaluated. One reason for this decision is probably the fact that TCP could bring long delay, which is a byproduct of
keeping a sequential order of original data streams. But, we believe TCP is still a promising solution for streaming in multicast scenarios, because its sliding-window-based approach has been widely proven as a successful solution of rate and congestion control in today’s Internet.

2.3 Choosing TCP as A Benchmark

As far as we know, little existing work on network coding has involved TCP as a benchmark. Sundarajan et al.’s work [35] is an exception. In that work, the authors proposed an approach to modify the classic TCP by adding a new layer of network coding into the protocol stack, and justified their modification by choosing TCP as a benchmark. Although both works take TCP and network coding into account, our work is not a simple extension of the existing work. Actually, these two works are totally different. First, the design objects are different. Sundarajan’s work aimed to propose a solution to achieve compatibility between network coding and TCP with the smallest amount of changes. Therefore, no optimization in terms of improving performance was intentionally made. However, the fundamental objective of our work is to evaluate the benefits of network coding. In our work, many design details are refined in order to optimize its performance. Second, targeting scenarios are different. Sundarajan’s work just focused on a single-hop unicast scenario with ordinary endpoint terminals, while our solution is designed to fit multicasting in a multi-path data center network. Third, the methodologies in the proposed solutions are different. Since Sundarajan’s work aimed to fit classic TCP, the proposed solution based their rate and congestion control on the sliding window. Yet, this work proposes an equation-based solution to avoid unstable sending rate in high-speed networks. In short, both work happen to involve TCP as a benchmark, but the ideas are totally different.

2.4 Bellini: A Platform for Evaluation

In our work, all experiments are run over the foundation of Bellini [30], a reusable and flexible framework that is designed to facilitate the rapid development, cloud deployment, and instrumentation of a wide range of media applications. Governed by design objectives of re-usability and flexibility, Bellini supports plugging in customer-tailored transport protocols, routing policies, and rate allocation strategies, in order to meet the needs of different applications. The implementation of Bellini is fine-tuned for performance, so that resources in cloud VMs can be efficiently utilized. With as few as 9000 lines of C++ code, Bellini supplies most of the features and components that are desired by a variety of media applications, including video conferencing and messaging. Some of design choices related to this thesis are explained as follows.

- **Transport Protocols**: Bellini is designed and implemented to support both TCP and UDP, to satisfy the needs of different media applications. With UDP, it supports the TCP-Friendly Rate Control (TFRC) protocol as the flow control protocol, as well as random network coding to improve performance and as a rateless erasure code. To be specific, the coding engine in Bellini generates coded packets by linearly combining original media data packets in a finite field \( GF(2^8) \), and decoding is performed progressively using Gauss-Jordan elimination, as coded packets are received. Any relay node can also produce coded packets by performing similar random linear combinations on received coded packets.

- **Flexible Source Routing**: In order to provide more flexibility to support various media services, Bellini is designed to allow sophisticated routing policies. A packet can be forwarded at relays in a
multi-hop path, or duplicated in any intermediate node in a multicast path. Among multiple concurrent media transmission sessions, each session is allowed to have its own set of routing policies. Furthermore, Bellini enables each individual packet can be transmitted following an arbitrary routing policy, generated by a given routing protocol.

▷ **Adjustable Flow Rates:** Since routing decisions can be made at the granularity of individual packets in Bellini, a transmission session may involve a mix of routing policies. For example, a media flow may be transmitted by splitting into several sub-flows, each of which following its own paths. Bellini achieves it by allowing the assignment of customized statistical distributions for each data packet at each hop, indicating the portion of a media flow being transmitted over each link. The distribution can be adjusted in a dynamic fashion, so that the rate of each sub-flow can be adjusted dynamically.

▷ **Adaptive Media Sources:** Besides the support of transmitting media content from an actual media source, Bellini provides a flexible data generator that facilitates the testing and experiments of new media transmission protocols in the cloud. To emulate different types of media sources, the data generator is regulated by a linear bounded arrival process (or equivalently, a token bucket), characterized by the long-term average rate $\rho$ and the maximum burst size $\sigma$. For example, for media files that can be transmitted with any arbitrary rate, $\rho$ is set to be $\infty$; and for video streams with an adaptive encoder, $\rho$ can be set to the maximum source rate of the adaptive encoder.

### 2.5 Summary

This chapter reviews several existing literature related to network coding. Although much work has been proposed on the benefits of network coding, few of them has taken delay into account. Also, most of works were focusing on traditional peer-to-peer networks which are composed of ordinary endpoint terminals. Few investigations have been made in data center networks. Even some research results have based their work on the new network architecture, little effort has been made to propose an optimized solution for practical use. Therefore, this thesis intends to provide some in-depth thinking based on experimental results in real-world scenarios. Before detailing our works, the next chapter first introduces some preliminaries on providing conferencing with network coded solutions.
Chapter 3

Conferencing with Network Coding

To provide high-quality conferencing services, we believe an ideal protocol must achieve the full potential of the multi-path communication paradigm to maximize the source rate while fulfilling the stringent delay requirements over a dynamics network. Therefore, we decide to adopt network coding to take full advantage of the one-to-all broadcast nature of multi-party video conference. Before introducing our work, this chapter first reviews *Airlift* [18], a successful network coded solution to provide multi-party conferencing. Actually, our work presented in later chapters is based on the foundation of *Airlift*.

3.1 Path Selection

To begin with, a conferencing system could be modeled as follows. Consider a network with multiple endpoint users. These users form a directed graph \( G = \{V, E\} \), where \( V \) indicates the set of these endpoint users, and \( E \) is the set of directed links in-between. For each link \( e \in E \), we use a positive real-valued function \( C(e) \) to denote its link capacity, which is the maximum available rate of packet transmission on link \( e \). Besides, another positive real-valued function \( d(e) \) is used to denote the propagation delay on link \( e \). Next, we use \( S^i \) to denote the source node of session \( i \), and \( R^i_j \) to denote the \( j \)th receiver of session \( i \). Then, our objective is to maximize the overall throughput of all session sources in \( G \), while the end-to-end delay \( d^i_j \) from a source node \( S^i \) to its receivers \( S^j \) is limited within a tolerable range, i.e., \( d^i_j \leq D_{max} \).

Given the delay constraints, we first narrow paths selection down to a feasible set. Assuming that queuing delay at each intermediate node can be neglected with small buffers, the end-to-end delay from source \( S^i \) to each receiver \( R^i_j \) can be estimated as the sum of all propagation delays along the acyclic routing paths. Concerning the existence of multiple routing paths connecting each source-destination pair, we have to ensure the overall delay along each path is bounded within the tolerable range. In other words, before arranging the routing paths for each session source, we first exclude the paths whose overall delays are larger than the given delay constraints \( D_{max} \), and only take the subset of feasible paths, \( P^i_j \), into account.

Inspired by the idea of conceptual flow [27], the routing process could be formulated as a problem of linear programming. With network conditions i.e., delay and link capacity of each overlay link and the end-to-end delay requirements as input, we could find weights of paths between each source-destination pair to maximize the one-to-many throughput for each session source, as Figure 3.1 shows.
Figure 3.1: Illustration for the procedure of path selection: (a) Given a network with four nodes, where link A-C is blocked and delay over each three-hop connection is intolerable; (b) The set of all paths is split and narrowed down to multiple eligible path sets for each destination and the source; (c) Two multicast trees are built up with eligible paths to cover all destinations.

To implement multicast service, each outgoing packet would be assigned to a set of eligible paths, one for each destination. Thus, multicast trees are built up to carry each packet to corresponding destinations. To achieve normalized path weights computed in the previous step, multicast trees are built up with different weights. Then, each outgoing packet will be assigned to one of these multicast trees based on corresponding probability. In order to build up these multicast trees and determine their corresponding weights, we use the algorithm as Figure 3.2 illustrates.

Let \( D(A) \) represents all destinations of source \( A \). For each destination \( i \in D(A) \), \( R(A, i) \) is the collection of all eligible paths in-between. Each element in the collection contains two types of information: (1) list of nodes along the path and (2) normalized weight of this path. In the example, there are three paths connecting source \( A \) and destination \( B \): \( A \to B \) (weight: 50%), \( A \to C \to B \) (weight: 25%), and \( A \to D \to B \) (weight: 25%), and another three paths between \( A-C \) and \( A-D \). To build up multicast trees, first elements with positive weight are chosen from each collection of eligible paths. Then, these selected paths are merged by overlapping same intermediate hops. Finally, the weight of this multicast tree is determined as the minimal weights among these selected paths. For example, paths \( A \to B \) in \( R(A, B) \), \( A \to B \to C \) in \( R(A, C) \) and \( A \to B \to D \) in \( R(A, D) \) are selected and merged to build up the first multicast tree with normalized weight equal to 25%. Next, weights of selected paths are updated to remove the portion used in previous multicast trees. And, these steps are repeated till the weights of all eligible paths are equal to zero. Finally, each outgoing packet will be allocated to one of these multicast trees based on the probability of 25%, 25%, 25%, 25%.

### 3.2 Coding Strategy

After multicast trees are determined, intra-session network coding is adopted at source nodes to mix original contents. At the receiver-side, original symbols could be recovered once sufficient coded blocks are collected.

With network coding, \( k \) data blocks \( b = [b_1, b_2, ..., b_k]^T \), each with \( s \) bytes, are to be transmitted...
from the source to multiple receivers in a network topology. The source transmits coded blocks, each coded block $x_j$ is computed as a linear combination of original data blocks $x_j = \sum_{i=1}^{k} c_{ij} \cdot b_i$ in a finite field (typically $GF(2^8)$), where the set of coding coefficients $[c_{j1}, c_{j2}, ..., c_{jk}]$ is randomly chosen. A relay node performs similar random linear combinations on received coded blocks with random coefficients, in order to produce recoded blocks to be relayed to a receiver. Coding coefficients related to original blocks $b_i$ are transmitted together with a coded block. A receiver decodes once it has received $k$ linearly independent coded blocks $x = [x_1, x_2, ..., x_k]^T$, 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 $b = [b_1, b_2, ..., b_k]^T$ 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), after receiving a new coded block. Figure 3.3 is an illustration for such procedure. When each coded block is transmitted, the identifier of the broadcast session, the starting sequence number of coded blocks, and all random coefficients are embedded within the same packet so that it is self-contained.

To implement network coding in real systems, sliding-window-based solutions are usually adopted to ensure original data could be successfully decoded. More specifically, when a new block of original data
unit is generated at session source, it is appended to the coding window. A maximum window size, \( W \), is imposed to guarantee successful data recovery at receivers, and to manage the coding complexity as well. The sender performs random network coding on original data blocks within the coding window, and transmits coded blocks to a relay or a receiver as newer original blocks are being added to the coding window. The coding window advances itself by removing the earliest original data block from the window, only when the sender has received acknowledgments from all the receivers in the broadcast session, confirming that the earliest original block has been correctly decoded by all participating nodes.

Therefore, another question worth mentioning is how should receivers acknowledge the sender of a broadcast session, in which an original data block has been correctly decoded? The existing solution is to acknowledge the degrees of freedom in all in-progress generations. Upon receiving each coded packet, the destination uses Gauss-Jordan elimination to achieve the reduced row-echelon form (RREF), and immediately replies an acknowledgment to the source. The acknowledgment contains the degrees of freedom in all yet-to-be recovered generations, 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.

The final question is about when should the source remove an existing generation and advance its sliding window. Since an aggregated session is a multicast session from a session source to multiple destinations, the source will only remove the oldest existing generation and advance its sliding window once all the destination in the session have indicated that they have successfully decoded the generation.

However, it is observed that these sliding-window based solution is inefficient in terms of bandwidth utilization, especially in high-speed networks, as Figure 3.4 (a) depicts. Due to the existence of propagation delay between sources and destinations, acknowledgments from receivers need extra time to return
source nodes and indicate the completion of recovery. During this period, not given any signal to stop and forward the window, source nodes keep generating and sending off coded blocks. Yet, these sent packets only contain linear combination of previous packets. As a result, these coded packets make no contribution to the decoding process, but wasting limited bandwidth resources.

Therefore, a *generation-based* network coding is adopted in the *Airlift* to relieve the problem of bandwidth redundancy, as Figure 3.4 (b) shows. The main difference between the *generation-based*
network coding and other traditional methods is the way to advance the coding window. With a
generation size of $n$, the source only transmits $n$ coded packets from the current generation by using
intra-session network coding. Afterwards, the source starts to transmit $n$ coded packets from the next
generation, instead of waiting for the completion of previous generations. Besides, in order to ensure
the reliability, blocks belonging to previous generations could be regenerated and sent off, if packet loss
is recognized based on feedbacks from receivers. To some extent, the generation-based mechanism is an
example of pipeline.

Finally, in order to limit end-to-end delay and ensure reliability in case outgoing links are blocked,
the coding window at the source side is configured with a maximum window size. If an outgoing link is
blocked, all coded blocks transmitted over that link are likely to be lost, thus the original data units of
these coded blocks cannot be removed from the coding window because the destination keeps requesting
retransmission. However, at the other end of the coding window, new batches of original data are added
to the coding window, once coded blocks of previous generations have been sent off. As a result, the
window size keeps growing until it reaches the maximum size. Then, no new original data is allowed to
added into the window, and all outbound bandwidth is reserved for the coded blocks belonging to the
earliest generation, until this generation is recovered by the destination.

### 3.3 Packet Redirection

As section 3.1 mentioned, in order to take advantage of network coding, the outgoing stream from a
source node is split into multiple conceptual flows, which share link bandwidth with each other using
network coding. Since each source has full knowledge on all of its conceptual flows, outgoing packets
are allocated to different conceptual flows based on corresponding weights. Also, since each
flow supposes to cover all destination nodes, multicast trees are constructed based on the weights of
conceptual flows. To indicate the list of intermediate nodes, information of multicast trees is embedded
to packet headers. Thus, intermediate nodes could easily decide next hops of a received packet by
examining its header.

![Diagram](image)

Figure 3.5: Three basic operations when multiple conceptual flows passing through an intermediate
node: (a) Replication, (b) Split, (c) Mix
From the perspective of how packets in an actual flow are processed in reality, there are three distinct cases as Figure 3.5 shows: (1) Packets in an actual flow are replicated and forwarded to multiple outgoing edges. This case is triggered by packet headers containing two or more downstream nodes. Payload of the incoming packet will be replicated and wrapped by different headers indicating different routing paths; (2) An actual flow is split to sub-flows with various portions, and each sub-flow is destined to different outgoing edges. Given different next downstream nodes, packets are be allocated to different outgoing queues; and (3) packets in multiple actual flows from the same source and destined to different destinations are to be merged with linear random network coding.

Since the first two cases do not require encoding actions, they could be simply realized and processed by parsing and following the instruction in the packet header. In other words, each incoming packet is forwarded or duplicated by the intermediate node as its routing header indicates. To realize the third case, all nodes find path overlaps between different conceptual flows if they are destined to different destinations in the same session. If such an overlap exists, the corresponding node 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. To merge the packets from different incoming flows, random linear combinations are performed on packets from the same generation at the source. The routing paths are merged by mixing the overlapped edges, while keeping others, like the union operation on sets. Also, the bandwidth of outgoing flows is equal to the maximum of different incoming flows, because only one coded blocks is generated for all blocks in the same position in different flows.

3.4 Rate Control

Although the reliability has been ensured by the coding strategy above, a rate control protocol is also essential to improve the efficiency of utilizing bandwidth resources of outgoing links, and to handle congestion at intermediate nodes. One solution is to use UDP at the edge of endpoints. More specifically, the rate of each outgoing link is controlled by an implementation of TCP-friendly Rate Control (TFRC) [3] protocol at the application layer. Due to the fact that random linear codes are rateless erasure codes, i.e., this is a perfect match to UDP as a transport protocol, since losing some coded packets is no longer a concern.

3.5 Summary

An existing network coded solution to provide multi-party video conferencing is reviewed in this chapter. This solution is mainly composed of three parts: first, a routing strategy to provide multicast service with bounded end-to-end delay; second, a coding strategy to ensure reliable transmission; and third, a rate control protocol to full utilize the bandwidth resources. After reviewing the existing solution, we will introduce a custom framework which is designed to evaluate a new transport protocol for multi-party conferencing in data center networks.
Chapter 4

Evaluation Framework

After introducing the general idea of network coded solution for multicasting, this chapter depicts our framework to evaluate a transport protocol in the context of multicasting. Our framework is composed of three major parts: 1) criteria to measure; 2) scenario to investigate; and 3) benchmark to compare with. To begin with, several criteria for evaluation are defined in the following section.

4.1 Criteria

To evaluate the performance of a transport protocol, throughput, without a doubt, is the most crucial criteria indicating the quality of service. It is always the most important metric to evaluate services like multimedia file transferring and video streaming services. Yet, video conferencing is much more complicated than other online services, not only for its stringent delay requirement inherited from live video streaming services, but also for some additional fairness-related demands like fairness and synchronization among multiple sessions involved in each conference. To evaluate a proposed solution in the context of conferencing, a set of criteria are designed in the mathematical forms as follows.

\[ \Delta \text{Maximum Achievable Rate:} \quad \text{The importance of throughput can never be understated, since it reflects the efficiency of using network resources. In this project, \textit{achievable rate} of a session source is defined as the minimal bit-rate of recovered data at receiver-sides among its all destinations. For example, as Figure 4.1 illustrated, node } A \text{ is broadcasting its data stream to other three nodes. During the transmission procedure, } r(A, j)[n] \text{ denotes the } n^{th} \text{ sample of receiving rate at destination } j. \text{ Then, the achievable rate of source } A \text{ is defined as } R(A) = \min_{j} \{\eta_n\{r(A, j)[n]\}\}, \text{ where } j \in \{B, C, D\} \text{ and } \eta_n\{S\} \text{ means the expectation of data set } S \text{ over dimension } n. \]

\[ \Delta \text{Achievable Rate Jitter:} \quad \text{This metric is designed to present fluctuation of source rates. Given a sequence of source rates in a time range, this value is computed as the ratio of standard deviation over the average of these records, e.g., } \delta_R(A) = \sigma_n\{R(A)[n]\}. \text{ where } R(A)[n] \text{ is the } n^{th} \text{ entry in the history of source rate, and } \sigma_n\{S\} \text{ is the standard deviation of data set } S \text{ over dimension } n. \text{ Fluctuation of source rates is also an important metric in practical systems, since smooth achievable rate means that short buffer is needed at receiver-side to ensure the smoothness of received data stream. In other words, consistent achievable rate is helpful to reduce end-to-end delay. Besides, with the blooming market of mobile devices, conferencing systems have to support a wide range of screen definition and network condition at different endpoint devices. Thus, scalable video coding is becoming a must. Therefore,} \]

17
the quality of a video is closely related to instant achievable rates. In order to provide smooth user experience, consistent achievable rate at receiver-side is highly expected.

- **End-to-end Delay:** In addition to source rates, delay-related metric is another key dimension of evaluation. In practical use, system designers always express their concern about the constraints of end-to-end delays, which is a water-shed between live streaming applications and other multimedia distribution services. Here, delay experienced by a data unit from node A to B, \( d(A, B)[n] \), is defined as the duration between the moment when the \( nth \) data unit is generated at source node A and its recovery time at destination B. Then, end-to-end delay is derived as the average delay of all data units transferred over the given path. The end-to-end delay from node A to B is computed as \( D(A, B) = \eta_n\{d(A, B)[n]\} \), where \( n \) is the sequence number of the recovered data unit and \( \eta_n\{S\} \) means the expectation of data set \( S \) over dimension \( n \).

- **End-to-end Delay Jitter:** Similar to the achievable rate jitter, this metric explores the variation of delay over a link. Due to the network dynamics and uncertainty, delay over each link is unstable. Therefore, an additional buffer is often embedded at the receiver-side to provide smooth playback experiences, but, at the cost of extra delay. To minimize extra delay cost, length of this buffer is usually designed to cover the gap between maximum and minimum delay only. Therefore, the end-to-end delay jitter over a link is involved as metric in this project. More specifically, the delay jitter over the link from node A to B is computed as \( \delta_D(A, B) = \eta_t\{\max_n\{D(A, B)[n]\}[t] - \min\{D(A, B)[n]\}[t]\} \), where \( n \) is the index of sample in the statistics window, \( t \) is timestamp of statistics window, and \( \eta_t\{S\} \) means standard deviation of data set \( S \) over dimension \( n \).

- **Unfairness:** Fairness-related criterion is another vital part within the evaluation framework, as a complementary viewpoint to efficient-oriented metrics, like throughput and delay. Within a multi-party conferencing, each participant receives data from multiple session sources, and its utility is mainly determined by the session of "lowest-quality". Therefore, with the concern of max-min fairness, the criterion of unfairness index is defined as the ratio of standard deviation over the average in terms of achievable source rates of all sessions within the system. More specifically, the unfairness index is computed as \( \varphi_R = \frac{\sigma_i\{R(i)\}}{\eta_i\{R(i)\}} \), where \( i \in \{A, B, C, D\} \) and \( \sigma_i\{S\}, \eta_i\{S\} \) are the standard deviation and expectation of data set \( S \) over dimension \( i \) respectively. It is obvious that protocols that achieve the less unfairness index are better.

- **Asynchrony:** Similar to the Unfairness, this metric is another fairness-related metric in terms of delay among multiple destinations from a standpoint of session source. Asynchrony of a session source is defined as the ratio of standard deviation over average in terms of end-to-end delay to different destinations. For example, the asynchrony of source A is computed as \( \varphi_D(A) = \frac{\sigma_i\{D(A,i)\}}{\eta_i\{D(A,i)\}} \), where \( i \in \{B, C, D\} \) and \( \sigma_i\{S\}, \eta_i\{S\} \) are the standard deviation and expectation of data set \( S \) over dimension \( i \) respectively. Similarly, with a given protocol, less asynchrony is expected.

### 4.2 Scenario

In order to take full advantage of limited network resources, most of practical scenarios could be treated as a peer-to-peer networks where endpoint terminals assisting each other by sharing bandwidth on edge links. It is true that the scenario of peer-to-peer networks have been well studied in the past decades, and many successful off-the-shelf products, like BitTorrent, are released. However, little efforts have been made in the context of providing multi-party video conferencing over instances in data centers. Unlike
architectures composed of ordinary endpoint terminals, links between cloud instances have better quality, including higher bandwidth, lower delay, etc. Besides, the instances are connected via multiple paths, and have access to multiple Internet Service Providers (ISP). As a result, provider of cloud services can easily relieve network burdens and improve quality of service by merging the network resources which globally distributed. Keeping these difference in mind, we focus our investigation on the performance of network coding in a data center network.

Therefore, we set up experiments over multiple instances in the Amazon Elastic Computing Cloud (EC2) as the topology illustrated in Figure 4.1. Also, these instances are configured as Table 4.1 summarizes. In this scenario, each node generates and broadcasts a unique data stream to all other three nodes via multiple links in-between. In order to take full advantage of network resources, peers are mutually assisted by sharing the edge link bandwidth. This scenario is designed to represent a generic case of data transmission among instances in a data center. Each instance in data-center is shared by a wide variety applications, many of which are globally deployed. Users of these applications need to share or exchange data with others located in remote countries or other continents. Therefore, data centers in different regions help to duplicate and transfer the user data. Since the number of such applications might be large, each instances merges data requests into integrated streams and broadcasts them to the instances in other regions.

![Figure 4.1: The scenario for evaluation is a network composed of four instances in the Amazon Elastic Computing Cloud (EC2). This scenario is set up to mimic real-time traffic among data centers in different regions.](image)

Table 4.1: Node configuration

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Zone</th>
<th>Location</th>
<th>Node Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>us-east-1c</td>
<td>N. Virginia</td>
<td>t1.micro</td>
</tr>
<tr>
<td>B</td>
<td>us-west-1a</td>
<td>N. California</td>
<td>t1.micro</td>
</tr>
<tr>
<td>C</td>
<td>us-west-2a</td>
<td>Oregon</td>
<td>t1.micro</td>
</tr>
<tr>
<td>D</td>
<td>ap-northeast-1a</td>
<td>Tokyo</td>
<td>t1.micro</td>
</tr>
</tbody>
</table>

4.3 Benchmarks

It is doubtless that setting up comparison between the proposed solution and existing protocols is helpful for readers to understand potential benefits and figure out drawbacks. However, selecting a proper benchmark to evaluate network coded solutions is not an easy task. It is known that a huge
amount of solutions have been proposed for communication systems. It is unwise to involve all of them into account, not only for the large amount of time to implement and set up experiments for them, but also for the fact that most of these proposals are designed and optimized for some specific objects. Thus, the benefits of network coding might be hidden by the optimization. In order to highlight the potential benefits of network coding, the classic Transmission Control Protocol (TCP) is selected in this thesis. We choose TCP as a benchmark mainly for two reasons: 1) TCP has been widely proven as a successful protocol for the rate and the congestion control in the Internet, but no existing work has compared its performance with network coded solutions in the context of multicasting services; 2) TCP is easy to implement and extend in order to support complicated scenarios for practical use.

To focus on the effects of coding, the benchmark is design with minimal difference between the network coded solution. More specifically, each participant within the system is playing three roles: \textit{Source} generates and broadcasts original data stream to all other members; \textit{Receiver} receives data streams from all other participants and replies sources acknowledgments to indicate the completion of recovery; \textit{Relay} assists other members by forwarding data streams. Similar to the network coded solution, each original data block is allocated to a pre-defined routing path, and appended to the corresponding \textit{outgoing link}. The routing information is embedded in the packet header. Each relay node parses the header of incoming packets, and redirects them to the next stop as the header indicates. But, unlike the design of network coded solution, all blocks are transmitted in plain formats, \textit{i.e.}, no coding is adopted. Also, flows to different destinations will not be merged. All flows at the relay nodes evenly share bandwidth of each outgoing link. Besides, the network coded solution and the benchmark share the same routing paths, in order to avoid the effects brought by choosing different routing paths.

\section{Summary}

This chapter clarifies the procedure of evaluation adopted in this thesis. Experiments are set up over instances in the Amazon Elastic Cloud to reflect real-world network conditions; a TCP-based benchmark is carefully designed to highlight the effects of network coding; and a series of metrics, including throughput, delay and fairness, are selected to ensure the evaluation could cover various aspects. Now, it is ready to look into the experimental results and figure out the main challenges of deploying network coding in real-world multi-party conferencing.
Chapter 5

Investigation

After the introduction of the preliminaries of evaluation, experiments in real-world scenarios are set up and results are collected and listed in this chapter. Also, an in-depth investigation is made based on these results, in order to figure out the bottleneck against further improvements.

5.1 Evaluation

For the purpose of evaluating network coding in data centers, a network is constructed with four Amazon EC2 instances as Figure 4.1 shows. To study the influence brought by multiple/overlapped paths in this scenario, both direct links and two-hop paths among instances are enabled. Also, equal weights are initially assigned to all routing paths.

The evaluation is started with throughput-related metrics, which are always the first concerns when a media communication is deployed in real systems. To measure the maximum achievable rate, experiments are launched with "flooding" mode, i.e., source nodes keep sending data with their best effort to saturate the capacity of their outgoing links. The measurement results are collected and summarized in Figure 5.1.

![Figure 5.1: Maximum achievable rate at different session sources](image)

Next, to compare the performance in terms of end-to-end delay, same fixed rates are assigned to all source nodes. Then, end-to-end delays statistics are probed and visualized in Figure 5.2. Also, based on
collected measurements on delay and throughputs, metrics related to *fairness* and *synchronization* are computed and summarized in Table 5.1.

The results indicate that network coding is promising in reducing the *end-to-end delay/jitter* in the scenario of *peer-to-peer networks*. Such advantage benefits from the given properties inherited from the rateless code, network coded protocols enable receivers to decode original data once adequate coded blocks are received, no matter from which path come the blocks. Therefore, with given routing strategy, network-coding-based solutions can always adaptively, automatically achieve the best load allocation over different paths. While TCP-based solution is limited by the connection with poorest connectivity. However, unlike the experiments shown in the proposal of *Airlift*, the superiority is not shown in these results, instead the results show that the TCP-based solution achieved better performance of the network coded solution, especially in terms of *throughput*. There are two possible reasons for the poor performance of network coding: First, the advantage of network coding is not taken. Because of the rich resources at these "super nodes", the overlapped links between them are of high-quality. In contrast, due to limited computing ability, and poor connectivity to endpoint users, bottlenecks of the network are located at the edge part, while shared links between "super nodes" are yet under-utilized. Second, the benefits of using network coding is overwhelmed by its accompanied overhead. Based on the results of *source rates*, it is obvious that there is a large amount of overhead in bandwidth cost. Therefore, the next section focuses on figuring out the root causes for the discrepancy between theoretical conclusions and the real experiment results.
5.2 Investigation

Based on the experiment results shown in the last section, it is known that network coding is promising in reducing the end-to-end delay, but its performance is defeated by the TCP-based solution. In order to figure out the root causes, this section dive into the components of these end-to-end performance to locate the bottlenecks.

5.2.1 End-to-end Traffic Decomposition

Based on the current design, traffic transmitted between user-pairs can be broken down into the following categories:

- **Control packets:** packets replied from receivers updating transmission progress;
- **Headers of data packets:** embedded bytes carrying general information, like identities of the packet’s sender and receivers, and its pre-assigned routing paths;
- **Payload of outdated data packets:** payload of packets that reach destinations after its carried data is locally recovered.
- **Payload of dependent data packets:** payload of packets reach destinations before the recovery of carried information. But, the carried coded blocks are just linear combination of previous ones that have reached the receiver-side;
- **Coefficients in the payload of useful data packets:** key information that indicates sequence numbers and coefficients of encoded blocks;
- **Coded blocks in the payload of useful data packets:** the actual payload containing mixed information of original data units.

In order to figure the reason for low throughput, traffic of each category is measured when various traffic load is carried by the network. Figure 5.3 unveils the interplay between source rates and the weights of different end-to-end traffic components. These diagrams shows that Traffic of outdated blocks is the major component of bandwidth overhead. Especially when the network is fully-loaded, i.e. in ”flood” mode, over half of the network resources are used to carry useless outdated traffic.

5.2.2 End-to-end Delay Decomposition

When network coding is enabled, life cycle of each data unit is composed of the following stages:

- **preparation phase:** period from the birth of a data unit to the moment when its first coded block is sent;
- **transmission phase:** duration from the first decoded block leaves the source to its arrival at the receiver-side;
- **recovery phase:** period from a coded block reaches the destination to the recovery of its original data. More specifically, length of this phase contains two parts: decode delay, which is the time for collecting adequate coded blocks to decipher original data, and reorder delay, which is the extra cost of waiting for recovery of all previous generations in order to provide the ”in-order” property.

To tell the main components of the end-to-end delay in various network conditions, length of each phase is probed with different network loads. Figure 5.4 summarizes the decomposition results of end-to-end delay.

From these diagrams, it is known that transmission delay is the main components of delay at low source rates, while the main part shifts to decoding delay when the network is heavily loaded. These...
results also come with the observation that 1) preparation delay is affected by queuing delay at the session source. When the network load is heavy and unstable, more blocks will be stuck at the source nodes; 2) transmission delay is mainly determined by propagation delay inside the network and the queuing delay at intermediate nodes; 3) decoding delay, as the corner stone of recover phase, is quickly increased, the coded blocks are densely interleaved. Thus, more blocks belonging to later generations are received before adequate number of first in-progress generation are recovered.
5.3 Analysis

Based on the experiment results, it is surprising to find that network coded solution doesn’t perform well in the cloud. Unlike the conclusion of most theoretical analysis, the network coded solution only achieves a low efficiency in terms of bandwidth utilization. After looking into the components of transmitted packets, it found that such low efficiency is caused by the high ratio of outdated blocks. Even worse, these redundant blocks not only reduce the end-to-end throughput, but also prolong the end-to-end delay for the network resources are wasted to transmit these useless packets. After looking into the behavior of sources and destination, it is found that the root cause of these problem is the imperfect mechanism to ensure communication reliability.

It is widely accepted that, as a special case of rateless code, network coding is born with the ability to deal with packet loss. Thus, the reliability of communication supposes to be easily ensured: the session source keeps generating and sending coded blocks until acknowledgments indicating the completeness of recovery at the receiver-side are the received. However, this is not the case in real communication systems due to its low efficiency of utilizing network resources. As we mentioned in chapter 3, given the existence of round-trip delay between source and destination, quite a lot of coded blocks could be injected by the session source before the acknowledgment is replied by the receivers. If the size of original data is small, while the sending rate is high, it is highly possible that most of these coded blocks are linear combination of each other. In contrast, when the size of original data is large, each original data unit has to experience a long delay before it is recovered at the receiver side, since long time is needed for the receiver to collect sufficient coded blocks for decoding.

Therefore, a basic protocols was proposed with the idea of splitting original data stream into multiple batches, and coded blocks are generated and decoded within each batch. Thus, possibility of long decoding delay could be limited by choosing small batch size, while the efficiency of using bandwidth resources could be improved by scheduling sending coded blocks in a pipeline way. In order to reduce the number of redundant blocks, the number of coded blocks are not larger than the size of each batch, unless packet loss is reported by the receivers. When the coded blocks of one generation has been sent out, the coding procedure will move onto the next batch, instead of waiting for the feedbacks from the receivers side. Thus, the efficiency of utilizing network resources could be improved.

To ensure the reliability of the communication, receiver nodes keep replying the number of outstanding blocks belonging to all in-progress batches. These acknowledgments help session sources to judge if packets are loss, and launch retransmission to compensate these lost blocks. In the original design, a coded block is treated as lost if any blocks belonging to a later batch has been received at the receiver side. Idea behind this mechanism is closed to the Auto Retransmission reQuest (ARQ) protocol, whose basic idea is to indicate the erasure of packet based on the absence of corresponding acknowledgment (ACK), and in this case, the sender simply regenerate coded blocks within the reported batch to compensate the outstanding degree of freedom.

This mechanism works fine in single-path scenarios for which the ARQ mechanism was designed. However, based on the experiment results, it is found that this mechanism doesn’t work in multi-path scenarios, where high ratio of false alarms for packet loss is raised. After diving into the behavior of session sources, we found this problem is caused by the fact that the original assumption on sequential order doesn’t apply in multi-path scenarios. Briefly, since coded blocks of each generation is transmitted over different paths with various latency, when a coded block of a new generation is received at one destination, it is highly possible that blocks belonging to previous generations are still on their way
Figure 5.5: An illustration for disordered packets at the receiver-side due to the existence of multi-paths to the destination. But, these blocks are transmitted over paths with longer propagation delays. For example, as Figure 5.5 illustrated, the source node is sending data to its destination via two paths with different propagation delay, without loss of generality, the path above has longer delay. Since the source node keeps sending blocks of next generations once previous batch has been sent out, and the blocks belonging to each generation are carried over two paths, it is highly possible that packets carried by the lower path will reach destination earlier, even it belongs to the next generation. Then, the destination node will reply acknowledgment indicating the outstanding numbers of different generations. And, the source node will launch retransmission for these "lost" packets, which are actually still on their way to the destination. As a result, redundant blocks are sent out and finally be treated as linear dependency at the receiver side. A more detailed example is illustrated in Figure 5.6 (a). Due to the existence of multiple paths, one coded block of first generation (blue) reaches the receiver-side after the arrival of two coded blocks belonging to a later generation. As a result, the session source is mis-led by the "false" acknowledgments, which are replied before the lagged coded block is received. Finally, two redundant coded blocks are created and sent off, even no actual packet loss happens.

Even worse, this problem is magnified by the design decision that source node will answer all acknowledgment in order to avoid loss of these feedback, and the retransmission is launched based on received feedback independently. As a result, when a replied feedback is eligible to launch retransmission, it is highly possible that the next acknowledgment will trigger retransmission as well, if these two feedbacks are generated by coded blocks transmitted over the same path. Figure 5.6 (b) is an example in this case. When a coded block is lost on its way to the destination, session source launches retransmission to reply the eligible request. But, this request is followed by another two requests which are caused by other coded blocks belonging to later generations. As a result, three coded blocks in total are sent to compensate one packet loss.

Because of these two reasons, the efficiency of the naive protocol is greatly degraded, and a large amount of bandwidth resources are wasted to carry these linearly dependent coded blocks. To improve efficiency of using bandwidth resources and realize the theoretically proven benefits of network coding, we have to find a way to incorporate the coding strategy, reduce redundancy and ensure the reliability in communication.

5.4 Summary

In this chapter, the performance of network coded solution is evaluated by comparing with the TCP-based benchmark. Yet, it is surprising to find that, unlike theoretical analysis, the performance of network
Figure 5.6: False retransmission is triggered due to the existence of multi-path between the source and the destination.

coded solution is disappointing, especially in terms of end-to-end throughput. After decomposing the end-to-end traffic, it is found that the root cause is the imperfect acknowledgment mechanism to ensure reliability. Although the reliability of end-to-end communication is ensured with current ACK/ARQ mechanism, a large amount of useless coded blocks are injected into the network. These redundant blocks not only reduce the end-to-end throughput, but also prolong the delay experience by each data unit. To improve the bandwidth efficiency while keep the reliability, an improved ACK/ARQ mechanism is introduced in the next chapter.
Chapter 6

Improvements

Based on the investigation in last chapter, we found the original design suffers from a severe problem of high bandwidth redundancy, which is caused by an imperfect acknowledgment mechanism to ensure communication reliability. It is true that some acknowledgment protocols have been proposed in the existing work [29]. But, these existing protocols do not apply in this work, because they were designed for sliding-window-based solutions, whose major challenge is to determine an optimal time to advance the coding window. However, this work is based on a generation-based solution, whose major challenge is to determine the correct time for retransmission. Therefore, this chapter introduces a tailor-made solution for the generation-based network coding, in order to improve its efficiency of utilizing bandwidth resources while ensuring communication reliability.

6.1 A Naive Time-out-based Approach

Based on our analysis in the previous chapter, it is known that the native retransmission protocol does not work well, as it will lead to an excessive amount of bandwidth overhead. The root cause for such an overhead is that, it takes a round-trip time for the source to learn any status updates about whether its most recent retransmissions from a given generation, say \( G \), have been successfully received. In the meantime, the destination will receive more coded packets from other generations, yet the acknowledgments associated to all these packets will still show that generation has not been fully decoded and is missing coded packets.

Therefore, one possible solution is to make source nodes hold off launching retransmission when a "valid" request arrives. More specifically, the source node should not retransmit coded packets unless the acknowledgments received after a period of time still indicates packet loss. Figure 6.1 shows how this idea is helpful to reduce bandwidth redundancy when (a) coded blocks carried by one path are severely lagged or (b) some packets are lost before they reach the receiver-side. The basic idea is to reserve enough time for packets carried over different paths have enough time to reach the destination. Thus, the feedbacks from the receiver could truly reflect the progress of packet recovery. However, this solution does not work in practice, because it is difficult to determine a proper duration for waiting. Theoretically, in order to ensure packets over all paths have reached the destination, the source node has to wait for extra time, say \( T \), after receiving a "valid" retransmission request. Here, \( T \) supposes to be larger than the gap between maximal and minimal values of propagation delay over different paths.
due to the network dynamics and uncertainty, it is difficult to probe or estimate accurate value of this gap. In addition, when the system is deployed in large-scale networks with a large set of eligible paths, this gap could be so large that overwhelms the maximum delay requirements. Yet, when the system is deployed over a mini-scope network with small gap, since the solution is timeout-based, it success hinges upon the accuracy of measuring round-trip times. If the round-trip time is over-estimated, the solution will delay the retransmissions unnecessarily, which is detrimental in a delay-sensitive video conference. If it is under-estimated, the problem of bandwidth overhead cannot be solved.

6.2 A Batch-based Acknowledgment Mechanism

In order to address the difficulty of probing, we propose an alternative solution, which does not depend on accurate round-trip delay. Briefly, we use the sequence numbers of received packets to help indicate
packet loss. This idea is quite similar to the classic Automatic Retransmit reQuest (ARQ) protocol, which has been widely proven as a successful solution to ensure reliability of communication systems. However, this solution doesn’t work in multi-path scenarios, where packets are not expected to reach receiver-sides in a sequential order. Yet, the idea of this protocol could be used in our generation-based network coded approaches.

As the previous chapter introduced, it is highly likely that packets within each generation are not received in a sequential order due to various propagation delays over different paths, as Figure 6.2 (a) shows. However, from a generation point of view, coded blocks of one generation cannot be surpassed by those belonging to a later generation, since these coded blocks of different generations are sent out in a sequential order, and all paths between the source and destination are used to carry these packets. In other words, even though the order of packets within one generation could be scrambled, the order of recovery from a generation viewpoint is still supposed to be sequential, as Figure 6.2 (b) shows.

![Figure 6.2: Illustration for the basic idea of batch-based solution](image)

In our protocol design, the source will only retransmit the missing packets from a particular generation $G$, if any of the later generations that were sent after $G$ has been completely decoded. Since the destination acknowledges each coded packet, the source has precise (but delayed) knowledge on which generation has been decoded on the destination. In the example shown in Fig 6.3, once the source observes that generation 2 has been completely decoded but generation 1 is still missing coded packets, it will retransmit a sufficient number of coded packets from generation 1. In case the retransmitted packet is lost once again in the network between the source and the destination, the source will retransmit it once another generation that is later than generation 1 - such as generation 3 - has been completely decoded.

Therefore, a packet erasure could be recognized when a generation of original data blocks have been recovered at the receiver side, while some generations before are still waiting for more coded blocks. Actually, this mechanism could be understood as an extension of ARQ protocol by merging multiple paths as one integrated connection, and treating one generation as a macro-packet in original one-to-one
communication model. With this solution, session sources require no pre-knowledge on accurate delays over different paths. Instead, packet loss could be recognized based on the history of acknowledgments, which greatly facilitates the decision make procedure.
6.3 Distinguishing Valid Acknowledgments

But, this generation-based ACK/ARQ mechanism alone cannot fix the problem of false retransmission. Briefly, this problem is caused by those outdated acknowledgments which were sent off before the retransmitted blocks arrives the destination. Without filtering these misleading feedbacks, session source would keep sending out useless coded blocks. Figure 6.4 (a) is an example illustrating the negative effects of outdated acknowledgments. Because two packets belonging to generation 1 are lost, the session source retransmits another two coded blocks when the first acknowledgment indicating the completion of generation 2 is received. If no acknowledgment filtering mechanism is adopted, as Figure 6.4 (a) shows, acknowledgments indicating the completion of generation 3 and 4 will trigger retransmission as well. But, these two acknowledgments are sent off before the arrival of previous batch of retransmission. In other words, these acknowledgments are outdated and should not be replied. As a result, redundant coded blocks are generated and sent off.

In order to deal with this problem, each source node is equipped with a dynamic indicator to filter out outdated acknowledgments. As the previous section mentions, to keep bandwidth overhead at a low level, coded blocks of one generation will not be retransmitted unless a "later" generation is recovered. Here, if generation $B$ is "later" than generation $A$, it means the last coded block of generation $B$ is sent off later than the last block of generation $A$. And, the earliest generation "later" than $A$, it is
Chapter 6. Improvements

denoted as A’s successor. Actually, the successor is a dynamic pointer indicating the boundary of valid acknowledgments. When generation A needs retransmission, its successor needs to be updated, because the time to send its last coded block is deferred. And based on the definition of successor, this indicator supposes to be moved to another qualified generation. In order to find the new successor, we first categorize all batches into four categories: 1) pending: this batch is newly generated, but no coded blocks have been generated and sent off; 2) sending: some coded blocks have been generated and sent off, but the number of sent blocks is still less than the batch size; 3) waiting: all scheduled coded blocks (including retransmission) have been sent out, but still waiting for feedbacks from receivers; 4) recovered: based on feedbacks from receiver-side, this batch of data has been fully recovered, and will be removed from the coding window. Also, a state transition diagram is illustrated in figure 6.5.

![State Transition Diagram of one network coded generation](image)

Figure 6.5: State Transition Diagram of one network coded generation

When a batch A is retransmitted, the source node will search all batches after, and assign the first batch, say B, belongs to type I or type II as A’s successor. It means batch B should be recovered later than batch A. Because neither of these batches have sent out enough valid blocks. Also, batches with lower indexes always have higher priority over later ones, in order to reduce the end-to-end delay. In other words, if the acknowledgment for batch B’s recovery arrives earlier than batch A, it means some of the retransmitted blocks are lost on their again. On the other hand, the acknowledgments indicating the recovery of batches before batch B will not trigger batch A’s retransmission, since this information alone is not sufficient for the source to tell if these retransmitted blocks have reached the destination successfully or not. An example illustrating the process of updating successors is in Figure 6.4 (b). After first batch of retransmission, the ”successor” is pointed to generation 5, which is in pending state when the retransmission is triggered. Therefore, acknowledgments indicating the completion of generations before ”successor” will be ignored. Thus, the problem in case (a) is fixed.

On the other hand, it is obvious that the proposed batch-based solution has many disadvantages. One major disadvantage is the lagged response to packet loss, since one packet loss cannot be recognized until sufficient coded blocks of a later generation have been received. Thus, a trade-off between delay and bandwidth is observed under such conditions. If larger generation size is selected, more eligible routes could be covered, thus the less unnecessary retransmission would be triggered. However, as the
generation size growing, longer decoding delay would be experienced when a packet lost does occur on the fly.

6.4 Evaluation of the ACK/ARQ Mechanism

To figure out if the benefits of proposed solution is overwhelmed by its drawbacks, several experiments have been set up in real-world networks. Also, the performance of proposed protocol is evaluated via comparison with the naive solution and two TCP-based protocols, one is the TCP-based solution with the same routing strategy, and the other is the standard TCP-based broadcast protocol, where no relay is involved. Similar to the previous chapter, an one-to-many scenario has been set up as the environment. Both throughput and delay related metrics are covered in the evaluation.

Figure 6.6: Illustration of Scenarios for comparison

From these results, it is observed that the proposed solution has greatly downgraded the ratio of

Figure 6.7: Comparison on achievable throughput between TCP-based and batch-based solutions

Figure 6.8: Comparison on the bandwidth redundancy between naive and improved retransmission solutions
redundant traffic, and thus the end-to-end performance has been greatly improved as well. Therefore, even suffering from some drawbacks, this proposed solution could greatly improve performance in real networks. Firstly, compared with measuring unstable round-trip delays over different paths, listening feedbacks from receivers could achieve higher accuracy of controlling. Also, with a small generation size, the burst of decoding delay could be reduced as well. In addition, due to the high quality of network conditions in the large-scale data center networks, the actual loss rate over different paths is quite low. Thus, the expectation of end-to-end delay will not be substantially affected by these busy decoding delays.

### 6.5 Load Balancing

In the previous section, an improved acknowledgment mechanism is proposed to reduce the number of outdated packets at the receiver side. It has been demonstrated that this improved ACK/ARQ mechanism is helpful to improve the efficiency of utilizing bandwidth resources. But, this mechanism alone is insufficient to take full advantage of bandwidth resources. For example, assume a source and a destination are connected via two links of different capacity. If the source node always evenly distributes its data stream over both links, it is obvious that one link will have idle bandwidth, while the other one suffers from packet loss. Therefore, even if all packets received by the destination is useful, extra effort is essential at session sources to assign proper weights over different outgoing links in order to maximize the multicast throughput.

To begin with, we first consider a unicast scenario with a pair of source and destination, which are connected via multiple paths in-between. In this case, the throughput between the source-destination pair is determined by two factors: 1) amount of received packets and 2) ratio of "useful" blocks to these received packets, as Equation 6.1 shows:

$$ T = \eta \cdot R $$

Where, $T$ denotes the bit-rate of recovered data stream at the destination; $R$ is the sum of receiving rate over all incoming links, $\eta = \frac{T}{R}$ indicates the ratio of useful data blocks to all received packets. As section 5.2 mentions, the end-to-end data streams between the source and the destination is composed of different types of traffic. Besides the coded blocks contributing to the recovery of original data, the end-to-end traffic also contains overheads like routing headers, encoding coefficients, outdated coded blocks, etc. Actually, the improved ACK/ARQ mechanism introduced in the previous sections is to maximize the value of $\eta$.

Also, as Equation 6.1 indicates, another crucial factor affecting the throughput is, $R$, the sum of receiving rate over different incoming links. This value is subject to two aspects: one is the sending rate at the source side, and the other is the network constraints like link capacities. This constraint is reflected in Equation 6.2 indicates, where $R_i$ is the bit-rate of packets received from the $i$th path; $S_i$ is the bit-rate of packets injected to the $i$th path; $C_i$ is the capacity of path $i$.

$$ R = \sum_i R_i = \sum_i \min (S_i, C_i) $$

Besides, as chapter 3 introduces, when a coded block is generated, it is allocated to one outgoing link based on the flow weight of that link. Let $\alpha_i$ denotes the flow weight over path $i$, the sending rate over
the \(i\)th path could be computed as follows:

\[ S_i = \alpha_i \cdot S \]

(6.3)

Where, \(S\) is the total sending rate at the source side. Since new coded blocks are created and sent off once the coded block belonging to the previous generations have been sent off, the sending rate is determined by the rate of adding new coded blocks into the coding window. Besides, due to the coding window at the source side is configured with a maximum window size, the rate of adding new data is almost the same as the rate of recovery at the destination side, no matter when the transmission is stable or congested. After taking packet loss and other overheads into account, the entire sending rate could be represented as Equation 6.4, where \(T\) is the throughput between the source and the destination, \(\eta'\) is a constant factor reflecting the overheads.

\[ S = \eta' \cdot T \]

(6.4)

In order to get information about the link capacity, loss rate along the \(i\)th path, \(l_i\), is measured at the receiver side. In real implementation, each outgoing packet is tagged by a path-wise sequence number and the index of the path between the source-destination pair. Then, examining the path sequence number at the receiver side, the loss rate is measured as Equation 6.5:

\[ l_i = \frac{\text{Number of missing blocks}}{\text{Number of received blocks} + \text{Number of missing blocks}} \]

(6.5)

Therefore, when packet loss along a path is reported by the destination, the capacity along this path could be estimated using the sending rate and the loss rate. If no packet loss is reported, it is difficult to estimate the link capacity. Inspired by the practice of the TCP-friendly Rate Control (TFRC) protocol, we estimate the capacity by doubling current sending rate along this link. If no packet loss is reported in the next round of feedback, it means the link capacity is under-estimated, and the capacity will be doubled again to approach its real value. If packet loss is reported, then the estimation will be refined based on the information of sending rate and loss rate. In short, the idea of estimating link capacity is summarized in Equation 6.6:

\[ C_i = \begin{cases} (1 - l_i) \cdot S_i & \text{if } l_i > 0 \\ 2 \cdot S_i & \text{if } l_i = 0 \end{cases} \]

(6.6)

After combining the equations above, the throughput between the source and the destination can be represented as a function of protocol overheads, path weights, path-wise loss rate, and the current value of throughput as Equation 6.7 shows:

\[ T[n+1] = \frac{\eta}{\eta'} \cdot R[n] \sum \min(\alpha_i[n+1], \alpha_i[n] \cdot \beta_i[n]) \]

(6.7)

Where,

\[ \beta_i = \begin{cases} (1 - l_i) & \text{if } l_i > 0 \\ 2 & \text{if } l_i = 0 \end{cases} \]

(6.8)

Therefore, the design objective of the load balancing mechanism is to maximize the throughput \(T\) by updating the flow weights over different paths, \(\alpha_i[n+1]\), on the base of existing flow weights, \(\alpha_i[n]\),
and current network conditions, like the path-wise loss rate $l_i$. More specifically, this problem could be formulated as Equation 6.9:

$$\max_{\alpha_i[n+1]} \sum_i \min (\alpha_i[n+1], \alpha_i[n] \cdot \beta_i[n])$$

$$s.t. \sum_i \alpha_i[n+1] = 1$$  \hspace{1cm} (6.9)

After solving this problem, we found the maximum value could be achieved if the flow weights are allocated as Equation 6.10 shows:

$$\alpha_i[n+1] = \frac{\alpha_i[n] \cdot \beta_i[n]}{\sum_i \alpha_i[n] \cdot \beta_i[n]}$$  \hspace{1cm} (6.10)

To implement this result, Algorithm 1 defines the initialization behavior at the source side to assign default values of path loss and path weights; Algorithm 2 is triggered when a feedback reporting path loss from the receiver is received; Algorithm 3 indicates the periodic process of updating path weights based on reported path loss and current value of path weights. At the receiver side, Algorithm 4 is the process of initialization; Algorithm 2 is designed to check packet loss by examining the embedded path-wise sequence numbers. This function is triggered whenever a data packet is received from the source side; and Algorithm 6 indicates the process of computing the path-wise loss rate and reporting the statistics to the source node.

```
begin
  for path ∈ eligiblePathSet do
    path.lossRate ← 0;
    path.weight ← 1/eligiblePathSet.size;
  end
end
```

Algorithm 1: Initiate path weights at the source

```
begin
  pId ← feedback.pathId;
  lossRate ← feedback.lossRate;
  if pId ∈ eligiblePathSet then
    path ← eligiblePathSet[pId];
    path.lossRate ← lossRate;
  end
end
```

Algorithm 2: Update path statistics on receiving a feedback

Besides the unicast scenario, this proposed load balancing mechanism also works in multicast scenarios. As section 3.3 mentions, when network coding is adopted, if a link is carrying multiple flows belonging to the same session, i.e., these flows are sent from the same source node, but having different destinations, this link could be regarded as a combination of multiple independent sub-links. Each of them has the same bandwidth of the original link, carrying only one of these flows. Therefore, a multicasting service could be regarded as multiple concurrent unicasting tasks, which are transparent to each other in terms of accessing bandwidth resources.
begin
   normalizedSum ← 0;
   for path ∈ eligiblePathSet do
      if path.lossRate ≠ 0 then
         path.weight ← path.weight × path.lossRate;
      else
         path.weight ← path.weight × 2;
      end
      normalizedSum += path.weight;
   end
   for path ∈ eligiblePathSet do
      path.weight ← path.weight / normalizedSum;
   end
end

Algorithm 3: Refresh path weights every 30 seconds

begin
   for path ∈ eligiblePathSet do
      path.lossCount ← 0;
      path.recvCount ← 0;
      path.lastSeqNum ← 0;
   end
end

Algorithm 4: Initiate path statistics at the receiver side

To balance the loads over different outgoing links, the source node could regard the multicasting service as multiple concurrent unicasting tasks. Then, based on Algorithm 1 to Algorithm 6, expected path weights between each source-destination pair could be separately computed. After running the algorithm in section 3.1 with expected path weights as input, the source node will construct a series of multicast trees with normalized weights. Next, each outgoing packet will be assigned to one of these multicast tree to cover its all destinations. Also, to adapt network dynamics, the weights of multicast trees are periodically updated based on latest path statistics.

begin
   pId ← packet.pathId;
   pSeq ← packet.pathSequenceNum;
   if path ∈ eligiblePathSet then
      path.recvCount ++;
      if pSeq > path.lastSeqNum + 1 then
         path.lossCount += pSeq - path.lastSeqNum;
      end
      path.lastSeqNum ← pSeq;
   end
end

Algorithm 5: Update path weights on receiving a data packet
begin
    for path ∈ eligiblePathSet do
        pathId ← path.Id;
        lossRate ← path.lossCount / (path.lossCount + path.recvCount);
        path.lossCount ← 0;
        path.recvCount ← 0;
        reply pathId & lossRate;
    end
end

Algorithm 6: Report path-wise lost rate every 5 second

6.6 Evaluation of the Load Balancing Mechanism

With the proposed mechanism of load balancing, the weights of eligible paths are expected to be updated to fit current network conditions. To verify this solution, a one-to-many multicast experiment is set up over instances in the Amazon EC2. In order to test the functionality of load balancing between different outgoing paths, each pair of the source and its destinations are connected via multiple paths. Also, in order to check the performance of sharing bandwidth of overlapped links, paths between different source-destination pairs are overlapped. Topology of the experiment is summarized in Table 6.1. Also, a benchmark of direct TCP is set up for comparison. In this benchmark, the source node broadcast its original data stream via direct TCP connections to its destinations. During this experiment, sustainable throughput are probed and plotted in Figure 6.9. Based on the results, it is obvious that the improved solution achieves superior performance to the TCP solution.

Table 6.1: Topology of the one-to-many experiment

<table>
<thead>
<tr>
<th>Session</th>
<th>Source</th>
<th>Destination</th>
<th>Eligible Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>A → B, A → C → B, A → D → B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>A → C, A → B → C, A → D → C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>A → D, A → B → D, A → C → D</td>
</tr>
</tbody>
</table>

Also, a many-to-many conference is set up with four nodes. Each node is a session source broadcasting its data stream to all other nodes via multiple paths. The configuration information is summarized in Table 6.2. The sustainable source rates of different sessions are measured and plotted in Figure 6.10. Based on these results, we know that the network coded solution achieves better performance than traditional TCP-based solutions.

Table 6.2: Topology of the many-to-many experiment

<table>
<thead>
<tr>
<th>Session</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B, C, D</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>A, C, D</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>A, B, D</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>A, B, C</td>
</tr>
</tbody>
</table>
Figure 6.9: Comparison on achievable rates between the direct TCP solution and the improved network coded solution in a one-to-many scenario.

Figure 6.10: Comparison on sustainable source rates of different sessions between the direct TCP solution and the improved network coded solution.

6.7 Summary

This chapter proposes an acknowledgment mechanism to improve the efficiency of utilizing bandwidth resources, while guaranteeing the reliability. This proposed solution improves the original design from three aspects: 1) modifying the algorithm of detecting packet loss to fit the multi-path scenarios; 2) filtering outdated acknowledgment to avoid unnecessary retransmissions; 3) updating the weights of data flows over different paths to adapt to the network dynamics. Based on the experimental results, it is demonstrated that the proposed solution fixes the high bandwidth redundancy problem in the original design, and beats the direct TCP solution in terms of multicast throughput.
Chapter 7

Concluding Remarks

This thesis presents a practical solution to improve the quality of service in the context of multi-party conferencing. This proposed solution is designed with network coding naturally embedded. Our choice of network coding is motivated by some of its widely-accepted advantages, like bandwidth improvements and reliability to deal with packet loss. The work presented in this thesis is based on the foundation of the Airlift, which is a successful network coded solution for multi-party conferencing. But, little efforts were taken in the existing work to dive into the bottlenecks against further improvements. And this failure motivates the work in this thesis. In order to make further improvements, the Airlift is reproduced and evaluated in a data center network. By comparing its performance with a TCP-based solution, we found the original design has a severe problem of high bandwidth redundancy. After diving into the components of end-to-end traffic, we found the high bandwidth overhead is mainly caused by the imperfect acknowledgment mechanism to ensure communication reliability. To tackle this problem, a batch-based ACK/ARQ mechanism is proposed to keep bandwidth overhead at a low level while ensuring reliability during communication. Besides, another load balancing mechanism is added at the source side to adapt network dynamics and maximize the multicast throughput. Based on experimental results, it is observed that the bandwidth overhead in the improved network coded solution is kept at a low level. Also, this improved solution manages to beat the TCP solution in terms of multicast throughput. Therefore, based on these experimental results in real-world scenarios, we draw that conclusion that network coding does bring benefits in the context of multi-party video conferencing.
Bibliography


