Efficient and resilient multimedia streaming on peer-to-peer networks

Li, Jiaming

2010

Li, J. (2010). Efficient and resilient multimedia streaming on peer-to-peer networks.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/47496>

<https://doi.org/10.32657/10356/47496>

Nanyang Technological University

Downloaded on 26 Apr 2023 21:42:07 SGT

3 ↑
2.17

EFFICIENT AND RESILIENT MULTIMEDIA STREAMING ON PEER-TO-PEER NETWORKS

LI JIAMING

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2010

Abstract

In the last decade, Peer-to-Peer (P2P) structure has gained a lot of popularity for data-intensive services, like file sharing applications. In P2P structure, every peer contributes its resource and helps to relay the data exchange. The load at the central server in the conventional server-client model is being distributed across all users. The service capacity increases with the number of users. Thus, the more number of users using the service, the more capacity the service can provide. The overlay of peers is built on top of physical network. Since peers have computational resources, it gives flexibility for peers to manage the structure of overlay. However, streaming applications, especially live streaming applications, have more stringent temporal requirement than normal file sharing applications. Physical network failure and data loss also need to be considered when building the overlay structure. Moreover, it is difficult to keep the P2P structure stable due to dynamic peer behavior. The three challenges hinder the quality of streaming services in P2P overlay.

This dissertation deals with three challenges from both underlying physical network and the application layer P2P overlay structure in order to improve the stream quality at each peer. It focuses on three critical components in P2P streaming system, namely, scheduling algorithm, data exchanges mechanism and P2P overlay construction to improve stream data distribution.

A scheduling algorithm is proposed to minimize the average latency of the streaming data to all users and improves the data quality. The data content and overlay structure are both considered when the peer sorts the order of the data in its sending queue. It is independent of the data codecs used. It sends the data which is requested by more number of peers and more important to the stream is first.

A data exchange mechanism incorporating the transmission efficiency of the push mechanism and the responsive protection of data integrity from the pull mechanism is

proposed. It pushes the stream data across a data distribution tree while publishing the data availability to the neighbors periodically. The mismatch between received data of the child peer and the notifications published by the parent peer reflects the lost data. Retransmission request is issued for prioritized missing data. The prioritized retransmission helps to maintain good stream quality and low overhead in a severely lossy and dynamic environment.

Finally, a resilient overlay construction scheme which provides a distributedly deployable solution for peer connection selection is designed. The usefulness of a new connection is calculated based on the existing source pool rather than evaluating the connections individually. The decision of acceptance of a new connection in the overlay is based on whether the new connection can improve the stream quality and robustness of all affected peers in the source pool comprising a peer and its current connections. The selection of peer connection is thus more cooperative in nature and not only improves the data quality and overlay robustness but also enables more efficient utilization of network resources.

Through comprehensive simulations, the three proposed mechanisms are proven to make the entire P2P streaming system more efficient and resilient.

Acknowledgments

Four years of Ph.D study is the most challenging and wonderful time in my life. Along the way, so many people have offered their help to me. I would like to express my sincere gratitude to my supervisor, Associate Professor Yeo Chai Kiat. Thanks to her for committing lots of her effort to support me in my study. Her critical insights and guidance inspired me along the way of my research. Her continuous patience and encouragement motivate me during the most stressful periods of my candidature. This dissertation would not have been possible without her guidance and support.

I would also like to express my appreciation to Associate Professor Francis Lee Bu Sung for his help rendered in my research at the beginning of my Ph.D study. My thanks also go to Assistant Professor Anwitaman Datta and Associate Professor Clement Chia Liang-Tien for their insightful discussion on some of the research problems faced.

I have worked in Center for Multimedia and Network Technology (CeMNet) for four years. I thank all my friends in this centre for the very many cherishable moments I have shared with them. They are Dr. Xu Min, Dr. Hu Yiqun, Zhang Yu, Wang Huan, Xia Yang, Huang Yi, Wang Danqi, Zhang Junwei, Zhong Feng, Liu Song, Zhou Zongzhao, Guan Wei, Dang Duc Nguyen, Luong Trung Tuan, Zhang Peng, Abhik Banerjee, Andi Widyo Cahyono, Zhang Da, Cheng Xiangang, Zhang Juyong, Yang Shengbo and Wang Lei.

This dissertation is dedicated to my parents and my wife, Zhao Li. Thanks for their constant support, understanding and love. I could never thank them enough for all the sacrifice they have made.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	viii
List of Tables	xi
List of Acronyms	xii
1 Introduction	1
1.1 Motivation	1
1.2 Issues of Streaming Applications in Overlay Multicast	4
1.3 Our Approach and Contributions	8
1.4 Organization of Thesis	11
2 Literature Review	13
2.1 A Typical Peer in P2P Streaming Services	13
2.2 Classification of Overlays	15
2.2.1 Tree based Overlay	16
2.2.2 Tree-Mesh based Overlay	20
2.2.3 Embedded Structured Overlay	21
2.2.4 Mesh based Overlay	23
2.2.5 Comparison of Different Overlay Structures	24
2.2.6 Single Tree vs Multiple Trees	26
2.2.7 Independent Multiple Trees vs Shared Multiple Trees	33
2.2.8 Multiple Overlays	35
2.3 Data Exchange Mechanism	36
2.3.1 Push vs Pull Mechanisms	36

2.3.2	Scheduling	39
2.3.3	Data Loss Protection	41
2.4	Membership Management	42
2.4.1	New Peer Join	43
2.4.2	Dynamic Membership Maintenance	44
2.4.3	Peer Failure Recovery	47
2.4.4	Centralized vs Distributed	48
2.5	Peer Behaviors	49
2.5.1	Peer Incentive	49
2.5.2	User Behavior	52
2.6	Summary	54
3	Content and Overlay-Aware Scheduling	55
3.1	Problem Formulation	56
3.1.1	Streaming Data Latency Modeling	56
3.1.2	P2P Delay Components	58
3.1.3	Weighting Packets by Content Importance	60
3.2	Algorithms	61
3.2.1	Scheduling Factor for Prioritization of Packets	61
3.2.2	Optimization Proof	63
3.2.3	Implementation of Algorithm	64
3.3	Content-Aware Weighting Scheme	65
3.4	Evaluation	69
3.4.1	Simulation Setup	69
3.4.2	Overhead of Proposed Scheduling Algorithm	71
3.4.3	Experiment Results	72
3.5	Summary	82
4	Light Weight Prioritized Retransmission Mechanism	84
4.1	Problem Formulation	85
4.1.1	Merits of Retransmission - Quantitative Comparison	86
4.2	Prioritized Retransmission Request	89

4.2.1	Retransmission Scheme	89
4.3	Performance Evaluation of Scheduling Algorithm with Retransmission . .	92
4.3.1	Simulation Setup	92
4.3.2	Results	93
4.4	Summary	99
5	Resilient Streaming Overlay Scheme	100
5.1	Problem Formulation	100
5.2	Source Pool Modeling and Stream Quality Maximization	104
5.3	Robustness of Peer's Source Pool	107
5.3.1	Iterative Robustness Calculation	108
5.3.2	Best <i>Robustness</i> Quality of Source Pool	112
5.3.3	Less Fine Scale <i>Robustness</i> Calculation	114
5.3.4	Selection of Substream with Bandwidth Conflicts	115
5.3.5	Heuristic Algorithm to Improve Calculation Efficiency	119
5.3.6	Meaning of Backup Substream Copy and Its Application	124
5.4	Best Source Pool Construction Scheme under Constrained Environment .	125
5.5	Deployment Issue	130
5.6	Performance Evaluation	134
5.6.1	Simulation Setup	134
5.6.2	Simulation Results	136
5.6.3	Discussion	144
5.7	Summary	145
6	Conclusion	146
6.1	Contributions	146
6.2	Future Work	148
6.2.1	Merging of Multiple-Stream Overlays	148
6.2.2	Unification of Live Streaming and Video on Demand Streaming Overlays	150
6.3	Summary	151

References	152
List of Publications	163

List of Figures

1.1	An example of IP Multicast Stream Blocked by ISP	2
1.2	Example of Overlay Multicast	3
2.1	IP Multicast Vs. Overlay Multicast	16
2.2	An example of NICE	19
2.3	An example of ZigZag	19
2.4	An example of ALM-CAN	22
2.5	An example of ALMA	28
2.6	An example of SVC and MDC	29
2.7	A example of LSONet	30
2.8	Random and Deterministic Overlay Construction	31
2.9	Comparison of Push and Pull Mechanism	36
2.10	The stream scheduling in PALS and its extension work	40
3.1	Delay of Packet p_k	59
3.2	Example of Queueing Time with Scheduling	60
3.3	Scheduling in Single Connection	62
3.4	Content-Aware Weighting Scheme	66
3.5	Content-Aware Weighting Scheme suitable for H.264	66
3.6	An example of content and overlay-aware scheduling	67
3.7	Mismatch of Data Weighting Scheme With Video Codec	68
3.8	Average Latency under Different Fluctuation Conditions and Overlay Structures	73
3.9	Average Data Latency for Different Levels in the Multicast Trees	73
3.10	An Example of SPT Structure	74

3.11 Average Stream Quality under Different Fluctuation Conditions and Overlay Structures	76
3.12 Average PSNR with Different Video Coding Schemes in SPT Overlay Structure	77
3.13 Stream Quality Distribution in Different Overlay	77
3.14 Stream Quality of Mismatched Weighting Scheme	79
3.15 Data Recovery Ratio	81
4.1 Example of Data Protection Comparison under Different Mechanisms . .	86
4.2 An example of DataMap for one connection	90
4.3 Data Quality in Different Fluctuating Environments	94
4.4 Data Quality in Different Lossy Environments	95
4.5 Average PSNR under Different Video Coding Schemes in Different Lossy Environments	96
4.6 Data Quality at Different Peer Levels	97
4.7 Data Quality by Different Retransmission Request Frequency	98
5.1 Comparison of independent and shared tree structure	101
5.2 Three Cases of Peer A's Source Pool	102
5.3 The Max Flow Model for Peer n_k 's Source Pool	105
5.4 An Example of Max Flow Solution in Graph	107
5.5 Impact of Connection Failure to the Source Pool	108
5.6 Selection of the 1st Set of Backup Substream Copies	109
5.7 Example of robustness computation	111
5.8 The Difference in <i>Robustness</i> between Computation of One Possible Solution versus the Best Solution after Investigating All Possible Solutions .	113
5.9 An Example of a Final Residue Graph	116
5.10 Another Example of a Final Residue Graph	116
5.11 Process for non-bandwidth sustainable substream selection	117
5.12 Optimal Solution for Source Pool with Both Bandwidth Supported and Non-Bandwidth Supported Copies	118

5.13 Example of keeping the current request instead of adopting the optimal one. Both have the same number of original (Set 0) downloadable substreams. 133

5.14 Example of adopting the optimal one when there is different number of original (Set 0) downloadable substreams. 133

5.15 Average Stream Quality for all proposals under Dist-2 expiration time distribution and loosened temporal constraint for Coolstream 138

5.16 Average Stream Quality under 5 Different Expiration Time Distributions 139

5.17 CDF for Start-up Latency Dist-2 Time Expiration in Scenario B 141

5.18 CDF for Recovery Time under Dist-2 Expiration Distribution in Scenario B143

List of Tables

2.1	Summary of Different Overlay Structures	27
2.2	Difference of Push and Pull Mechanisms	39
5.1	Notations	104
5.2	Peer Expiration Time Distribution Cases	136
5.3	Overhead Load of Different Overlays under Different Expiration Time . .	144

List of Acronyms

- P2P** Peer-to-Peer
- IP** Internet Protocol
- ISP** Internet Service Provider
- QoS** Quality of Service
- RTT** Round Trip Time
- MST** Minimum Spanning Tree
- SPT** Shortest Spanning Tree
- DHT** Distributed Hashing Table
- RP** Rendezvous Point
- SVC** Scalable Video Coding
- MDC** Multiple Description Coding
- FIFO** First in First out
- LIFO** Last in First out
- PSNR** Peak Signal-to-Noise Ratio
- FEC** Forward Error Correction
- CAN** Content-Addressable-Network

GOP Group of Pictures

RTP Real-time Transport Protocol

R-S Reed-Solomon

BIP Binary Integer Programming

VoD Video on Demand

Chapter 1

Introduction

1.1 Motivation

The infrastructure of Internet has developed very fast during the last decade. More and more broadband access networks are available to Internet users, which makes data intensive applications possible to the end users. However, the conventional server-client model is non-economical for content providers to buy large amount of bandwidth in order to support such dense data distribution to a large number of users at the same time.

To cope with the bandwidth problem in server-client model, Internet Protocol (IP) multicast[1] is proposed. A tree structure is built among stream data receivers and intermediate nodes, like proxies and routers. Data packets are replicated only at the intermediate nodes and only one copy of the data is distributed to the different branches in the multicast tree. It is one of the most efficient ways for distributing large volume of data to multiple users at the same time. However, Internet Service Providers (ISP) do not wish IP multicast traffic to flood their network and such traffic is often blocked at the ISP, like in Figure 1.1. Hence, although IP multicast has been proposed and researched for over one decade, there is still limited deployment of IP multicast over large scale network.

In the last decade, peer-to-peer (P2P) technology[2–6] has gained a great popularity in network applications. In P2P network, every user downloads different parts of a

Figure 1.1: An example of IP Multicast Stream Blocked by ISP

large data chunk from different source providers. Meanwhile, the data downloaders are contributing their upload bandwidth and acting as service providers to provide the partial data they have in their buffers to the other users. Since the data downloaders are also source providers, more connections are made available to each user than the server-client model. Congested connection can be bypassed through downloading from other source providers. Hence, the download bandwidth for each user increases. The more users join the network, the more download bandwidth they will get and the faster the downloading process will be. Users in the network are considered as peers. The logical connections among peers for data distribution form an overlay network, or P2P network, on top of physical network. Data is distributed through the overlay to each peer. Since the data request and distribution are all happening in the application layer, the underlying structure is transparent to the peers. The abstraction of the network infrastructure from P2P applications facilitates easy accessibility for the end users to receive the service from the other users and provides flexibility for the data service provider to organize the structure of its data distribution channel.

P2P network requires no support from the underlying infrastructure. The data exchange is considered as unicast between hosts across different ISPs. Thus, the application level multicast data can bypass the restriction imposed by the ISPs to replicate and distribute data at peers instead of at the routers, which bypass the blocking issue in IP multicast. In application level multicast deployed across overlay networks, the peers are no longer pure receivers as in traditional IP multicast. Besides receiving data, they also

CHAPTER 1. INTRODUCTION

replicate data and distribute to multiple receivers, as illustrated in Figure 1.2. The shift of multicast operation from IP layer up to application layer leads to the feasibility of deploying multicast applications to large number of peers across large scale networks.

Figure 1.2: Example of Overlay Multicast

Comparing with conventional server-client model, the P2P structure has a lot of benefits, especially when the number of serving clients is large. The server, in P2P network, only needs to provide limited bandwidth and computational resource which is usually only enough for several peers in conventional server-client model. The peers contribute their own bandwidth and computational resources to the overlay, and help to forward the data while receiving the data. It is thus a self-expanding distribution system. The more peers join the overlay, the more resources are available, the larger the capacity of the overlay, and the more peers can be supported.

Being peers without special resource provisions and given the large number of peers, it is impossible for peers and server to hold full information about all peers in the overlay. Hence, the server cannot build an optimal overlay throughout the peers, but can only provide limited information for the new peers and guide them to find their starting positions in the overlay network. Peers have to periodically update the information about surrounding overlay conditions in order to place themselves in a better position and receive proper QoS. The ability of self organization for each peer helps them cooperate with each other and maintain an operational overlay structure.

Though the P2P network is efficient for data intensive applications, the streaming service has more stringent requirements than normal data delivery services. The stream

data has stricter temporal requirement than data downloading. Network congestion and fluctuation make the underlying network unpredictable, which affects the overlay structure. Peer joins and departures are unpredictable, which make the overlay structure prone to partitions. The strict temporal requirement of stream data imposes tight constraints on peers to respond in a timely manner to the changes in the overlay, rendering the overlay more fragile. To cope with the problems, traditional P2P streaming systems would buffer data for a longer period to provide proper service quality. We shall examine the problems faced by the important components of P2P streaming network, and discuss the solutions to optimize the operations of these components comprehensively. We will discuss how such solutions can be deployed in a distributed manner at each peer in order to provide better quality of service with little incremental cost and overhead.

1.2 Issues of Streaming Applications in Overlay Multicast

The P2P network provides an alternative to deploy multicast to deliver intensive data services to large number of users at the same time. However, the streaming application is different from the normal data delivery in the following manner.

- The stream data has more stringent temporal requirement than the normal data downloading services. If the stream data fails to arrive at the destination before the data's expiry time, the data is meaningless to the users. The bandwidth used in the delivery of the stream data is thus wasted.
- Though the underlying network is transparent to the P2P network, the changes in the underlying network still affect the quality of service (QoS) of the P2P network. Congestion in the physical network is unpredictable. It leads to bandwidth fluctuation in the overlay connection, and results in an unstable overlay.

CHAPTER 1. INTRODUCTION

- The incongruent mapping of the overlay structure to the underlying network means that the overlay constructed for peers will not be able to fully utilize the network resources from the underlying structure, since the latter is being abstracted from the higher layer overlay network.
- Peer churns. New peers joining the overlay and existing peers leaving the overlay are unavoidable during a service session, since user behavior cannot be controlled. The newly joined peer will first require resources from existing peers to deliver the stream data to it without being able to contribute back to the overlay at the onset. After that, it will continuously update its position in the overlay until a relatively proper position is discovered. It leads to the relocation of newly joined peers and existing peers in the overlay. When existing peers leave the overlay, they may not inform the others in advance. It may cut off the connection between them and their neighbors immediately, which also leads to the reallocation of the existing peers in the overlay. Such reposition of peers in the overlay further contributes to the overlay instability and degrades the stream quality at each peer and their neighbors.
- The stringent temporal requirement of stream data allows peers little time to adapt to network dynamics and peer churns, hence rendering overlay to be more fragile to the unpredictable changes.

Conventional solutions for existing problems try to abstract the negative impact of the network and overlay dynamics from the end user by improving the data quality while sacrificing other quality of the stream services as follows:

- The temporal requirement for the stream data is loosened in order to give peers longer time to recover from the changes in the overlay. Currently, P2P streaming

CHAPTER 1. INTRODUCTION

applications, like PPLive[7] and PPStream[8] require latency of several minutes in order to buffer up enough stream data to playback at a stable data rate. For some time critical streaming applications, such as live broadcast, the minute-granular latency may be unacceptable.

- In order to minimize the negative impact of unstable bandwidth, stream data is partitioned into several substreams. The substreams are distributed through different paths. The loss resulting from an unstable peer connection is thus only confined to the substreams delivered through the path, like in SplitStream[9] and CoopNet[10]. It can improve the utility of resources in the heterogeneous environment and isolate the negative impact of connection congestion or failure. However, managing multiple substreams increases the complexity of overlay construction and management and involves more control overhead.
- Pulling data from neighbor peers allows a peer to quickly recognize a connection failure, since peers frequently contact each other bidirectionally, like in CoolStream[11]. Thus, the time taken for a peer relocation in the overlay is reduced. However, since the pull mechanism uses bidirectional contact, it needs three sets of message exchanges before a stream can reach the destination. They are: (i) the data provider peer publishes what it has in its buffer to the neighbors; (ii) the neighbor peers request for the missing parts from the data provider peer; (iii) the data provider peer delivers the stream data according to the requests. Such three-way contacts triple the delay of streams through the connection. The actual delay incurred will also depend on the frequency of publishing the information and requests for information. Though higher frequency can reduce the streaming delay along the connection, it inevitably increases the control overhead and soaks up bandwidth.

CHAPTER 1. INTRODUCTION

- The push mechanism for data exchange requires only a pair of message exchange during the initial peer connection. The stream will be forwarded as soon as it arrives at the data provider peer according to the request set at the beginning. There is no need for periodic contact before stream data delivery. Thus, the push mechanism is more efficient than pull mechanism for stream data, especially in live streaming applications. However, the push mechanism takes a longer time for the overlay to recognize connection degradation or failure because of the infrequent bidirectional contacts to exchange the connection status. The overlay needs a longer time to recover from network and peer dynamics.
- Overlay structure is a well studied topic in existing research works. Though the resilience is emphasized in every overlay structure, most of work that have addressed the allocation of network resources for backup streaming service[12] are based on the selfish model which will exploit network resources as much as they can at the expense of the QoS of other peers. This results in some peers having abundant resources while other peers are highly deficient in backup resources. The network resources are thus not efficiently distributed and utilized.

Much work[10, 11, 13, 14] is focused on overlay structure and stream quality. Other important issues in streaming applications are sacrificed in the process. Examples are overly-delayed stream cannot satisfy certain users' experiences; utilization of network resources; large control overhead wastes precious bandwidth resources; fragile overlay needs more network and computational resources to backup the streaming services. It is thus necessary to consider from the overall view of the overlay, and to solve the problems in a distributed manner such that it is easy to be deployed at each peer under the circumstance that every peer holds only partial information about the entire overlay.

1.3 Our Approach and Contributions

This work addresses the network and peer dynamics in overlay multicast streaming services and designs the algorithms to optimize the operation of the peers' important components in P2P streaming services in a distributed manner, which helps to improve the overall performance of the entire P2P streaming system.

We focus on three basic operational components in P2P networks: scheduling, data exchange mechanism and overlay structure. Our approach to optimize each component is as follows:

- **Scheduling** Scheduling in P2P network focuses on prioritizing the stream data based on different criteria to differentiate data of varying importance in the stream and distribute the data according to the priority. The most common scheduling algorithms are FIFO, LIFO or rate-distortion prioritized. However, most existing scheduling mechanisms do not consider overlay information while they deliver the stream data. In our proposed approach, the scheduling algorithm considers both the content delivered in the stream data as well as the overlay condition as perceived by each peer. In particular, we consider the number of descendent peers served by the current peer. The priority for each data block is then calculated on the fly, and the data is delivered based on the priority.
- **Data Exchange Mechanism** To manage the pros and cons of the push and pull mechanisms in traditional P2P streaming applications, the push mechanism with light weight prioritized retransmission request is proposed. Peers forward the stream as per push mechanism while periodically publish the sent data to the child peers. Child peers response with acknowledgement for the received data and issue prioritized retransmission request for the lost data. It not only inherits the efficiency of data delivery characteristics of push mechanism, but also incorporates

CHAPTER 1. INTRODUCTION

the quick response to network dynamics and transmission loss which characterizes the pull mechanism.

- **Overlay Structure** In order to cope with peer churn while considering fair resources allocation, a robustness oriented peer selection scheme is proposed. The quality of a peer connection is not determined by what can be provided by that connection itself but is also dependent on how the stream data from the connection can complement the existing connections from other peers. The mechanism unifies the ability to provide the necessary streaming quality with the ability to provide as complete as possible backup copies of stream data that can be supported from the existing connections. Every peer tries to build a robust pool of connections for itself which can provide the more complete stream and the more number of copies of such stream as backup from the pool. The distributed optimization solution for resource allocation facilitates the prudent distribution and better utilization of network resources than the common exploration and contention scheme which is confined to the quality of individual peer connection without consideration for existing peer connections.

The optimization in the operation at each peer improves the performance of the overlay multicast system as follows:

- Our proposed content and overlay aware scheduling algorithm shortens the queueing delay of streaming data at each peer, and thus reduces the experienced stream latency. The improvement in the transmission efficiency further helps to increase the possibility for stream to arrive on time and improves the stream quality.
- The scheduling algorithm considers the overlay structure in addition to differentiating the stream data. This not only improves the QoS at each peer but also

CHAPTER 1. INTRODUCTION

incentivizes peers to contribute more if they want to receive better QoS in the system.

- The scheduling algorithm works independent of the codecs used in delivering the video stream. Existing methods which exploit the feature of a particular codec in the prioritization will perform badly under mismatched prioritization of the codec.
- The bidirectional data exchange with prioritized retransmission mechanism in our approach can work under extreme network conditions where bandwidth fluctuations are huge and loss ratio in the peer connection is high. It mitigates the avalanche impact of data loss through the tree structure overlay. The bidirectional information exchange keeps the transmission efficient as in push mechanism, inherits the responsiveness to losses from pull mechanism and maintains the control overhead at a low level. The mechanism outperforms the existing methods and provides better streaming quality to the peers in the overlay.
- Our proposed connection quality measurement scheme in the overlay construction integrates the criteria of measuring stream quality as well as the robustness of the peers' stream providers. The uniqueness of our method lies in that it evaluates the quality of peer connections by how the connection can complement other existing connections rather than just evaluating the connections' profile individually. The new measurement helps peers to find the best combination of existing connections to achieve the best robustness while receiving the streaming service in a resource constrained environment.
- The proposed peer connection selection scheme helps to maximize robustness in the overlay and also allows resource to be allocated among the neighbor peers in a cooperative manner. It prevents contention among peers and enables network

resources to be fairly allocated throughout the overlay for both streaming and backup purposes. The utilization of the network resources is thus improved.

The research methodology adopted here is as follows: First, a comprehensive literature survey is conducted on P2P systems to identify the strengths and limitations of the current proposals. This is followed by identifying the problems to be addressed and then conceptualizing and designing the algorithms. The algorithm is then analysed quantitatively through analysis and extensive simulations. The performances of the proposed solutions are then evaluated against some representative existing proposals through extensive simulations.

1.4 Organization of Thesis

This dissertation comprises 6 chapters. In Chapter 2, existing research works on P2P streaming services are presented. The characteristics of P2P streaming systems are introduced first. The overview of several important components in P2P streaming system is presented, such as overlay structures, data exchange mechanism, task scheduling and membership maintenance together with the respective representative literature.

Chapter 3 proposes the content and overlay aware scheduling algorithm for P2P streaming system. The scheduling impact in P2P streaming services is analyzed from a single peer connection and is then extended to the whole overlay. The analysis is formalized into a mathematical model which helps to find the optimized solution for the problem. The solution is proved with the mathematical model and corroborated with simulation results under dynamic network environment.

Chapter 4 examines the pros and cons of existing streaming mechanisms and proposes a bidirectional data exchange with prioritized retransmission mechanism. It exploits the efficiency of data delivery from the push mechanism and integrates the quick response of

CHAPTER 1. INTRODUCTION

the pull mechanism to data loss and link failure. It is tested both in a normal operating scenario and an extremely lossy network scenario and performs much better than using either one of the traditional mechanism.

Chapter 5 focuses on improving the robustness of the overlay structure for P2P streaming services. Network resources are more fairly allocated to facilitate the provision of streaming and backup connections than existing overlay structures. The proposed overlay structure is more resilient to network dynamics and peer churns. It also provides peers with better stream quality under tight temporal requirement.

Conclusions and recommendations for future work are detailed in Chapter 6.

Chapter 2

Literature Review

P2P overlays have emerged as promising solution for live data streaming on the Internet. Many challenges faced by P2P streaming systems compared to traditional P2P data delivery as been discussed in Section 1.2. This chapter provides a brief introduction on the role of a typical peer in a P2P data streaming system followed by a survey of the different approaches which have been proposed to address issues in P2P streaming.

2.1 A Typical Peer in P2P Streaming Services

When a new peer joins the streaming service session, the peer contacts the central server for registration in the service and requests for information of the stream and candidate peers to contact. The server registers the new peer in its peer list, sends the information about the stream to the new peer, and sends out a set of candidate peers from the peer list for the new peer. After receiving the candidate peers list, the new peer contacts these candidates and check whether a connection is possible between each other. If the connection is feasible, the connection is built.

After the connections are set up, the newly joined peer is a part of the overlay in the P2P streaming system. It will request stream data from the existing peers. The peers, which receive the request, will respond and send out the data. Data is streamed out by parent peers based on a one-time request at the beginning of the connection

CHAPTER 2. LITERATURE REVIEW

construction, if the data distribution is based on the push model. On the other hand, the pull mechanism which is receiver-driven requires the new peer to periodically request for the desired data from existing peers.

While the peer is receiving the stream data, it still periodically exchanges information about more potential peers in the overlay with its neighbor peers and server, and attempts to build connections with those which can be more helpful in improving the data service and data quality. In the meanwhile, due to resource limitation at the peers, physical network failure or peers quitting the overlay, a peer may lose contact with its connected neighbors. To dynamically adapt to the changes in the overlay network, peers should periodically monitor the connection status. Once a disconnection is detected, peers will switch their data requests from the lost neighbors to other existing connected neighbors.

When a peer leaves the service session, it disconnects with all the neighbor peers with or without notification. If the peer notifies the neighbors about the departure, the peer gives its neighbors some time to seek out alternate data sources and adapt to the changes resulting from peer leaving the overlay, thereby minimizing data loss at its neighbors, if any. Otherwise, the peer simply closes all connections and leaves the overlay. This will result in a partition in tree-based overlays.

Since the peers and server have limited resources, they can only have partial knowledge of the whole overlay. Thus, it is neither possible for the server to organize the overlay structure to be optimal, nor for the peers to find their best position in the overlay. Moreover, even if peers can be placed at the optimal positions, it is impossible for them to maintain such positions all the time due to the physical network dynamics and peer churns. In P2P streaming, server can at most provide brief instruction and exercise loose control on the peers due to scalability issues. Meanwhile, peers should employ their own strategies to seek out the proper positions for themselves. The strategies should be deployable in every peer as well as facilitating the cooperation among peers.

There are many ways to classify overlays. In the following sections, we will classify overlay structures based on the topology design, followed by discussions on overlay data exchange mechanisms, membership management and peer behaviors.

2.2 Classification of Overlays

Though the overlay structure is self-organized by the independent peers, every peer has to follow some organization rules in order to form a working and efficient structure to enjoy the appropriate service quality. Usually, the design starts from the definition of the properties in the framework to fulfill the application targets. These properties are then decomposed into the applicable rules for each peer to follow so that the peers form an overlay consistent with the design goals.

Overlay topology can be based on tree, tree-mesh, mesh or embedded structures. Data distribution across the overlay topology is either through the pull or the push mechanisms or a combination of both. From the perspective of data streaming, the pull mechanism is usually adopted in overlays with a stable data distribution channel and data is distributed through the channel constantly based on a one-time request. This is referred to as the push mechanism. The data distribution usually takes the form of a tree or other structured entities. The pull mechanism is often adopted by less structured or unstructured data channel and is data-driven i.e. data delivery is based on periodic data request. Different overlay structures exhibit different characteristics and hence can support the different applications under different scenarios.

Tree based overlay is somewhat similar to IP multicast tree. The difference is that data duplication and dispatch in IP multicast tree are performed in the intermediate routers and proxies of the network infrastructure as opposed to at the end hosts in a P2P overlay in the application layer as shown in Figure 2.1. Tree based overlays can also be classified by single tree overlay and multiple-tree overlay. The latter can further be

Figure 2.1: IP Multicast Vs. Overlay Multicast

categorized into independent tree and overlapped trees based on shared paths. Trees are deemed as efficient for data distribution but are more fragile as they are vulnerable to partition in the event of a peer failure.

Tree-mesh based overlays usually adopt a 2-step design where peers first organize themselves distributedly into a mesh after which trees are built across this mesh. Mesh structure is more robust to node failure given its characteristic multipoint connectivity. Hence the underlying mesh network serves to strengthen the trees and facilitate quick recovery in the event of a tree partition.

In the embedded structured overlays, peers are set and connected based on a logical address space on top of the physical network. The underlying logical address space follows certain rules to keep the structure intact. Overlay utilizes the functions provided by the underlying logical structure to search for neighbors and exchange data.

Mesh based overlays as defined in this dissertation, build data distribution channel randomly without adopting any regular structures such as trees or embedded mechanisms. The data distribution channel can also be dynamically built upon pull requests by peers.

2.2.1 Tree based Overlay

In tree based overlay, peers are organized in a tree structure topology. There are explicit hierarchical relations between peers. Server is located at the root of the tree. Some peers are located at the branches of the tree as both data receivers and data forwarders. The

CHAPTER 2. LITERATURE REVIEW

rest of the peers are located at the leaves of the tree. The stream data are distributed from the root through the branch peers to the leaf peers.

Application Level Multicast Infrastructure (ALMI) [15] is a centralized tree structure system where a central controller builds a minimum spanning tree (MST) across the peers. Round Trip Time (RTT) between the links are taken as the cost of an edge in the graph when building the MST. To adapt to network changes, the RTT of the peer connections are monitored by the individual peers and reported to the centralized controller. The controller will periodically re-compute a new spanning tree based on the updated peer data. This ensures the efficiency of the overlay as peers are always located in the best positions of the tree in a dynamic environment.

RITA[16] is another tree structure overlay which also monitor the QoS at each peer. When network congestion occurs, only the closest peers response to the changes and request for other peers through a Distributed Hashing Table (DHT) infrastructure [17]. The streaming connection is directed to the unaffected peer. The handover process at the closest peer does not affect the descendent peers in the tree branch and the transition is smooth.

Owing to the preference of selecting short RTT links for peer connections, the stress on such links increases and there is a need to balance the load on network to ensure the efficiency of the entire overlay. The work in [18] considers application level priorities as well as the network characteristics. The optimization algorithm builds a multicast tree which balances the characteristics of MST and Shortest Path Tree (SPT).

Application Layer Multicast Architecture (ALMA)[19–21], aiming at reducing latency and data loss, builds the connection between peers based on loss rate first and RTT second. When new peer finds the peer which can provide the best perceived quality, it joins the peer and becomes part of the multicast tree.

Overlay Multicast Network Infrastructure (OMNI) [22] forms a tree among media streaming proxies in the network. It builds the tree based on the delay between proxies

CHAPTER 2. LITERATURE REVIEW

as well as the priorities granted to the proxies, giving higher priorities to proxies which support the most number of peers. Each proxy in turn supports a tree of peers. The more peers a proxy supports, the higher is the priority of the proxy. Peers which have high priority can preempt the low priority peer connection and are likely to move up in the tree structure. The constant updating of the overlay improves its efficiency and hence reduces the overall stream data latency experienced by the clients.

LagOver [23] focuses on how to efficiently organize the peers in a differentiated temporal constrained scenario. In order to support as many peers as possible in a resource limited environment, peers in LagOver will try to move the peers which can support more number of child peers to the upper level of the tree close to the root so long as the movement does not violate the temporal requirement of other affected peers.

In NICE[13] and its derivatives[24–26], a B+ tree like hierarchical tree structure (without links between different groups), as in Figure 2.2, is built. Peers group into clusters at the lowest level, as level 0 in Figure 2.2(a). Peers communicate with each other within the cluster. Each cluster picks a cluster leader who can communicate with other cluster leaders. These cluster leaders in turn form clusters at a higher level of the NICE tree. The same process takes place up the hierarchy of the tree to the highest level. When a new peer (i.e. the white node in Figure 2.2(a)) joins NICE, it is bootstrapped to the peers in the highest level (i.e. layer i) of the tree. It will measure the latency between itself and these peers and choose the closest peer to graft itself to. This closest peer who is the cluster leader of a cluster in the lower level (i.e. layer $i-1$) will inform the new peer of its cluster members. The new peer will repeat the process of choosing the closest peer to itself within this cluster. Hence, the new peer iterates the process from the root of the NICE tree until it selects a peer in a cluster at the lowest level (i.e. layer 0) to join. The streaming channel is built along the control channel. The received stream data at each peer is forwarded to every peer in the cluster that the peer belongs to iteratively, except

CHAPTER 2. LITERATURE REVIEW

the cluster where the white node resides, since the white node would have distributed the stream to all peers as shown in Figure 2.2(b).

Figure 2.2: An example of NICE

Another hierarchical tree structured overlay is ZigZag [27], which also organizes peers in a B+ tree like structure, as shown in Figure 2.3(a). However, the stream data is not distributed through the tree branches as in NICE, but through the connections set up between child peer and foreign parents in different cluster in a higher level as in Figure 2.3(b). This is to avoid bottleneck from a single connection and to protect against tree partition.

Figure 2.3: An example of ZigZag

CHAPTER 2. LITERATURE REVIEW

DagStream [28] is based on RandPeer[29]. The latter groups the peers in a hierarchical manner into a distributed trie based on QoS data supplied by the peers. Peers can choose the QoS characteristics which they wish to use as metric to organize themselves in the trie. A trie is a data structure which is basically a tree with hierarchically labeled node ids. For DagStream, network locality is used as the metric to build the trie. The position of peers in RandPeer is a soft state, which is updated periodically to avoid any collision in peer id.

Since all peers in the overlay have limited resources, load balancing is necessary for overlays where peers are clustered such as NICE, DagStream. Clusters will be split when the number of peers exceeds the set threshold for cluster size to avoid overloading. Conversely, small clusters will be merged to improve the efficiency of the overlay and reduce overhead.

2.2.2 Tree-Mesh based Overlay

Peers in Tree-mesh based overlay also build the connections between their neighbors. However, unlike tree based overlay, there are no explicit hierarchical relations between different peers within the same group. Loop is allowed in the overlay structure. The stream distribution channel is built in the form of a tree structure through the mesh of peer connections. Adaptation of the tree structure is thus based on the changes in the mesh connections.

Peers in Narada [30, 31] have the full information about one another so that they can build a mesh across all peers. This information is periodically updated to ensure that broken mesh connections are mended and short latency connection with good QoS can be established in the mesh overlay. A Shortest Path Tree (SPT) is built across this mesh to distribute the data stream. Every peer is connected to the peer which can provide the shortest latency. Since peers have to maintain full information about everyone, the scalability of Narada is limited.

Kudos [32] extends the scalability of Narada by employing a two-level structure. Peers are organized within approximately $N^{1/2}$ clusters with $N^{1/2}$ peers per cluster on average. Each cluster has a cluster leader responsible for membership management in the cluster and communicates with other cluster leaders. Peers within each cluster and the group of cluster leaders are organized using Narada. Hence the scalability of the original Narada is squared in Kudos.

2.2.3 Embedded Structured Overlay

For embedded structured overlay, a logical structure is built on top of the physical network. It provides the connection between different parts and the ability to search within the structure. The peer overlay is built on top of such logical structures. Since the underlying structure can provide some basic functions, it saves peers efforts in searching for peer neighbors. The reliability of the underlying structure determines the quality of the overlay. The maintenance overhead incurred for the underlying structure also adds to the overall system overhead.

Application-Level Multicast using Content-Addressable Networks (ALM-CAN) [33] is built on top of Content-Addressable Network (CAN) [4]. The peers are placed in a multi-dimensional Cartesian coordinate space, and are in charge of forwarding the streaming data to all its adjacent neighbors in the space. A 2-dimensional space CAN is shown in Figure 2.4(a). A new peer Z, who wishes to join the multicast group will be bootstrapped to an existing member say X. Z then picks a random point in the 2-D coordinate space (say Y's space) and sends a join request through X to locate Y. Y's zone is then split into 2 with Z owing one part of it, as in Figure 2.4(b). Merging of zones occur when a peer leaves the system. The disadvantage of ALM-CAN is that the overlay is built without taking into account the physical network distance between the peers. Hence the data distribution paths may be very long and thus inefficient.

Figure 2.4: An example of ALM-CAN

Scribe [34] is a large-scale event notification system that uses application layer multicast to disseminate data on topic-based publish-subscribe groups. Scribe is built on top of Pastry [5] which is itself a P2P object location and routing substrate overlay. Each member in Pastry is assigned a random node identifier, which may be generated by computing the cryptographic hash of the member's public key. Messages are routed among members by knowing the node identifiers. Each member has a routing table, a neighborhood set and a leaf set. The routing table for a member, h , contains information about a set of members in the overlay with which the member, h , shares a common prefix. Scribe uses Pastry for its underlying route management and host lookup to provide multicast services. A multicast group in Scribe consists of a subset of the members that have already joined the Pastry overlay. Each multicast group in Scribe has its own topic identifier. The member whose node identifier is numerically closest to the multicast group identifier becomes the Rendezvous Point (RP) (i.e. root node) for that group. When a member on the overlay joins a multicast group, it routes a join message using the

multicast group identifier as the destination identifier. This message gets routed by the Pastry substrate towards the RP. All members on this unicast path that are not already a part of the Scribe multicast data delivery tree for the group add themselves to the tree. Scribe maintains multiple connections for robustness.

2.2.4 Mesh based Overlay

Mesh epitomizes an unstructured overlay. Peers in mesh overlay form a flat structure where the peers connect to multiple neighbor peers. There is no explicit hierarchical or structured relation between the two ends of a connection. Thus, there is no stable parent and child pair in each connection. Stream data is divided as data chunks and distributed across the overlay similar to file distribution. Each peer publishes the data held in its buffer to the other connected neighbors periodically. For the data chunk which the peer does not have but is available in others' buffers, the peer will send a pull request for that data chunk.

It is to be noted that there is a difference in the definition of mesh structure in this dissertation and other works which may also classify multiple-tree structure and tree-mesh as a mesh. Though overlapped multiple trees and tree-mesh can be perceived like a mesh across the peers, it is still different from the mesh structure from the perspective of stream data. Multiple-tree overlay builds stable data delivery trees when peers first join the system. Data is streamed across such trees and each peer knows where it should forward the received data. The stable tree structure distribution channel is also built in the tree-mesh overlay. Hence, for data in the same substream, the distribution path from the source server to every peer is along the same tree. In our defined mesh structure, the data distribution channel is dynamically built upon a pull request from a peer and a sourcing peer will only know where to forward the data when it receives a pull request from other peers. There is thus no pre-defined data distribution path.

Chainsaw [35] is an example of mesh based overlay. It eliminates the streaming tree built across the peers. The connection is randomly built among peers. The stream data is considered as normal data as in P2P file sharing applications, like BitTorrent [2]. Peers notify the neighbors when new data packets arrive, and randomly select neighbors with data availability to pull the desired data from.

PRIME [36, 37] is a classic representation of mesh based overlay. Peers pull part of the stream data through parents in an implicit sub-tree structure, and exchange the other parts of the stream data with other peers which belong to different sub-trees. The sub-trees in PRIME are dynamically organized as peers initiate the data request and are not constructed at the point when peers join the session, unlike tree based systems where the data distribution is pushed via the tree to all the peers.

CoolStream [11] is considered another mesh based overlay. In neighbor peer selection, peers give preference to those who have larger upload bandwidth and more available data segments. The data is pulled based on locally rarest first in order to help distribute the rarest data faster and improve the overlay throughput.

PALS [14] is also a receiver-driven mesh structure overlay. The receiver peers try to build the connections with sender peer which can improve its overall throughput. The load of data request is proportional to the senders' upload bandwidth.

2.2.5 Comparison of Different Overlay Structures

2.2.5.1 Topology Perspective

Tree and tree-mesh design can be viewed as network topology-aware design while embedded structure such as ALM-CAN is generally network topology-agnostic. Network topology-aware design uses active measurements to gather network information to construct efficient data distribution overlays to match the physical network topology as close as possible in order to better utilize the network resources. Topology-agnostic approach

CHAPTER 2. LITERATURE REVIEW

ignores network characteristics. Hence the former has the advantage of a more efficient overlay but at a cost of increased management overhead. Peers in embedded structured overlay leave the membership management issues to the underlying structure. Most of the overhead is generated from the underlying structures.

In tree and tree-mesh overlays, membership management, detection and repair of partitions caused by departing or failing peers incur substantial overheads. Quick actions have to be taken to maintain the overlay in order to ensure its continuous operation and efficiency. Some structured overlay, like NICE, also generates overhead to maintain the structure in merge and split operations. Embedded structured overlay also generates overheads from the underlying structure for the same purpose. In mesh overlay, since there is no structure to maintain, such overhead is less than the others. The overhead for monitoring and maintaining peer connections as well as in measuring network metrics is the same, since it is necessary for all types of overlays.

Performance of overlay which has a fixed structure to maintain is very much dependent on the construction of the overlay as peers join the system since data is streamed along the overlay. Unstructured mesh overlay on the other hand depends very much on whether the peers can find the best connection combination and whether peers can locate the best positions in the overlay, both of which are dependent on the criteria used in the selection.

2.2.5.2 Data Streaming Perspective

Since most of the mesh overlays do not have explicit and stable parent and child relation between peers, there is no need to maintain any structure. Comparing with tree, tree-mesh and embedded structures, the mesh structure does not impose a fixed position for peers in the structure. Peers can connect and disconnect with the other peers more freely without worrying if the disconnection may lead to partitions in the overlay as in the case of tree based structures. This property enables the peers to adapt their positions to network changes more promptly.

CHAPTER 2. LITERATURE REVIEW

As most of the mesh overlays are receiver-driven, it gives peers more flexibility to pull data from the best connections than others. Peers can promptly adjust the data request combination among multiple connections according to the changing bandwidth. The adjustment scale can be as fine as packet level, while the adjustment scale for tree based overlays can only be a change in the data rate of one substream.

It is easier for mesh overlay to recover from connection failure than tree overlay. When connection failure is detected, the impact can be mitigated by redirecting the pull request of the missing data to other connections immediately. In tree structure, peers cannot divide the load of one substream into packet level and distribute the load to multiple connections. Thus, the recovery from the missing substream in tree overlay could only be accomplished by redirecting the substream request to another single connection.

The receiver-driven data request also leads to some drawbacks. The periodic request initiated from the receiver incurs 3 one-way delays between sender and receiver to pull the requested data from sender. This is 3 times longer than the push mechanism in structured overlays. Thus, peers in mesh overlay experience longer latency. Moreover, since peers pull data from multiple connections, it has to keep a large buffer to mitigate the asynchronization of the stream. Overheads incurred in publishing data availability and requesting for data are unavoidable in mesh structure, which consumes additional computational and network resources.

The strengths and weaknesses of the different overlay structures are summarized in Table 2.1.

2.2.6 Single Tree vs Multiple Trees

2.2.6.1 Single-Tree Overlay

Overlay structures, like ALMI, ALMA and MSN, build single-tree structures to distribute the stream data. Data is distributed from a single parent peer to all child peers as a single

CHAPTER 2. LITERATURE REVIEW

Table 2.1: Summary of Different Overlay Structures

Overlay Type	Topology Aware	Resilience	Overhead	Maintenance	Data Exchange
Tree	yes	weak	light	easy	push
Tree-Mesh	yes	good	normal	normal	push / pull
Emedded Structure	no	best	heavy	hard	push / pull
Mesh-based	yes	good	heavy	easy	pull

stream. The connection bandwidth between parent and its child peers should be at least equal to the stream rate.

Since there is only a single connection for the peers to obtain stream data from the source peer in single-tree overlay, the structure is fragile to peer and link failures along the data distribution path. Thus, some backup connection is necessary to help peers quickly recover from the failures in order to improve overlay robustness. ALMA employs gossip-style protocol similar to [38, 39], to provide peers with additional connections to other parts of the tree as shown in Figure 2.5. When a peer detects a link failure with the parent peer or when stream quality is degraded, the peer will switch to the backup connections, thus avoiding a tree partition. Peers in NICE use Probabilistic Random Forwarding (PRM) [24] to proactively forward some random data packets to other neighbors and build contact with other peers across the tree structure to improve the robustness of NICE. Though peers in Narada receive stream from a single parent, it builds multiple connections for every peer. When a failure occurs, Distance Vector Multicast Routing Protocol (DVMRP) [1] like mechanism is employed to recalculate the routing table in order to find the shortest latency path for the affected peer.

2.2.6.2 Multiple-Tree Overlay

Data stream can be divided into multiple substreams, delivered through different paths and reassembled at each peer for display. Multiple trees are built across the peers to

CHAPTER 2. LITERATURE REVIEW

Figure 2.5: An example of ALMA

provide different channels to distribute each substream. Different substreams have their own trees. New peers join different multicast trees for different substreams.

The negative impact of peer or link failure is isolated within the connection. More options are available to peers to recover from such failures compared to a single-tree structure. The missing substreams can be requested from other connected peers, or the peer can use the recovery mechanisms in single-tree overlays to recover from the disconnection. Since substreams from different paths may experience different delays, buffering is necessary for peers to mitigate the temporal jitter of received data from the different substreams.

In PROMISE [40], peer selects the best combination of bandwidth based on a topology-aware algorithm which provides maximal throughput for the peer with the least bandwidth collision between connections. The stream is divided into packet-scale substreams. Each substream has the packet bit rate in unit time. The portion of substreams within the whole stream requested by the peer from a single connection is proportional to the ratio of the connection bandwidth to total download bandwidth.

ChunkySpread [41] focuses on how to balance the load when distributing multiple

CHAPTER 2. LITERATURE REVIEW

substreams. Each peer has a capacity for the forwarding load it can contribute to the overlay. If the current load is way below a threshold, the other peers will try to become the current peer's child. If the current load is too high above the threshold, the child peers will try to find other peers who are not overloaded, as parents.

The development of video coding technology helps to further improve the stream quality of the P2P stream application through multiple channels. The scalable video coding (SVC) scheme [42] separates the whole stream into one basement layer and multiple hierarchical enhancement layers, as in Figure 2.6(a). It provides a solution to successfully display partially received stream in a heterogeneous network environment. The basement layer is most important in determining the stream quality, shown as the black box in Figure 2.6(a). The higher enhancement layer requires the support from the basement layer as well as all the enhancement layers below it as illustrated in Figure 2.6(a). The more higher enhancement layers the peer can get, the better is the stream quality it receives.

(a) Scalable Video Coding

(b) Multiple Description Coding

Figure 2.6: An example of SVC and MDC

CHAPTER 2. LITERATURE REVIEW

Multiple description coding (MDC) [43] provides another solution to display partially received stream. Under MDC scheme, the substreams are equally important and can be decoded independently, as shown in Figure 2.6(b). Any combination of different description substreams can recover partial stream data. The more substreams received, the better is the stream quality. The property of MDC coded stream facilitates stream distribution in heterogeneous environments and improves resiliency of stream quality in dynamic network environment.

LSONet [44] utilizes the hierarchical layered feature of SVC, and proposes a multiple-tree join optimization algorithm to get maximum stream quality combination through multiple connections according to their bandwidths. Figure 2.7(a) shows an example of connections with several stream layers available at one peer. The boxes in each column show the stream layers available at each peer. The gray boxes show the number of stream layers that can be delivered from that connection based on the download bandwidth. The connections are sorted based on the parent peers' available stream layers, as illustrated in the transformation from Figure 2.7(a) to Figure 2.7(b). A peer's join request starts from the lowest stream layer from the parent with the lowest quality within the bandwidth constraint. The process is iterated until all the download bandwidth is consumed or all the stream layers have been requested as shown in gray boxes in Figure 2.7(c).

Figure 2.7: A example of LSONet

SplitStream [9] also focuses on evenly distributing forwarding load across the peers through multiple-tree overlay. The difference between it and other multiple tree overlays

CHAPTER 2. LITERATURE REVIEW

is that, in SplitStream, one peer contributes all its resources to forward data in one substream through one multicast tree while joining all other trees as leaf node and receives data only. The overlay is built based on Pastry [5] to locate close peers as neighbors in the overlay.

Peers in overlay could only contribute limited resource for the whole service session. Thus, if each peer contributes its bandwidth to all substreams, it could only support a small number of child peers. The depth of multicast tree is large when the peer number in the service is big. It leads to longer latency for stream data and nodes located far from the root node in the multicast tree are more likely to experience data loss due to the data arriving after the expiration time in live streaming applications. CoopNet [10] leverages the idea from SplitStream, and builds a deterministic multiple multicast tree structure to make the tree as short as possible.

Figure 2.8: Random and Deterministic Overlay Construction

For example, every peer, except root server, can only contribute upload bandwidth to support one full stream or two half streams, as in Figure 2.8. Root server can support two peers to get full stream. Stream is evenly divided into two substreams and delivered in two multicast trees. If all peers contribute their bandwidth to both trees as in Figure 2.8(a), the maximum depth for the overlay is 3, while the average depth is 2. If the

CHAPTER 2. LITERATURE REVIEW

overlay of CoopNet is built under the same condition as in Figure 2.8(b), the maximum depth for a peer is 2, while the average depth is 1.67. Less peer hops in the overlay reduce data latency and provide more reliable data delivery for peers in the system. In CoopNet, parent peers are selected based on estimating the host location by using a simple delay-coordinates technique.

2.2.6.3 Discussion

The difference between the single-tree structure and multiple-tree structure leads to different techniques for peer management. As there is only one tree to maintain, it is easier for peers to join and be updated within the overlay. There is also less overhead generated. Multiple-tree overlay requires maintenance of all the trees and hence generated much more overhead. Peers joining one tree should avoid contention for bandwidth with other multicast trees.

Multiple-tree overlay is more resilient than single-tree overlay. In the absence of backup connections in single-tree overlay, any failure will cause partition and terminate the sole stream source. It will take the orphaned peer some time to rejoin the tree. Gossip-style peer discovery could facilitate fast recovery from partition in single-tree overlay although the negative impact of loss stream is still significant, unless a huge buffer is employed.

Compared to single-tree overlay, the negative impact of connection failure in multiple-tree overlay is isolated within the connection. Since the peer is still connected to other peers, the recovery process is much faster. Buffering time of multiple-tree overlay is relatively longer than single-tree overlay as the latter streams out data sequentially while the former has to manage the out-of-order arrival of the different substreams. Although the buffering needs in both cases are due to different reasons, the net difference in buffering time will be minimal.

As a single parent peer provides the full stream in single-tree overlay, connections with small bandwidth will be excluded from the overlay. On the other hand, in multiple-tree overlay, since the stream can be divided into substreams with low stream rate and are delivered separately, the connections whose bandwidth cannot support the full stream rate can still be utilized. Thus, single-tree overlay is confined to a more homogeneous network environment, while the multiple-tree overlay is feasible for both homogeneous and heterogeneous environments. The latter is also more efficient in maximizing the use of network bandwidth.

2.2.7 Independent Multiple Trees vs Shared Multiple Trees

2.2.7.1 Independent Multiple-Tree Overlay

In independent multiple-tree overlay, peers join different trees by requesting the substreams from different parent peers. There are no shared branches between different two different trees. Since the loss of connection may have negative impact on the stream quality, requesting different substreams from different independent connections helps to isolate negative impact within a single substream.

Examples of independent multiple-tree overlay are SplitStream and CoopNet, which focus on how to isolate the negative impact and keep the overlay resilient when connection failure occurs. Thus, peers join the distribution tree by connecting to different parent peers.

2.2.7.2 Shared Multiple-Tree Overlay

The shared multiple-tree overlay presents a different structure where peers can join multiple trees through one connection. Thus, distribution trees for different substreams may overlap. The sending rate of the stream data can be adjusted to the available bandwidth by controlling the number of substreams which are allowed to be delivered through the connection.

CHAPTER 2. LITERATURE REVIEW

Examples are ChunkySpread, SPIDER [45] and ID-Host [46] which focus on improving the stream quality under heterogeneous environment. The difference is that ChunkySpread takes care of load balancing at each peer. SPIDER emphasizes on how to improve the total throughput of the overlay in a given topology. The work in [47] uses linear programming to optimize routing structure and maximize the throughput at the receiver while ensuring fairness within and across different substreams. The design of ID-Host prioritizes peers based on their contribution in all trees. The higher priority peers are moved closer to the root in each tree. The prioritization mechanism is also employed in LBTree [48]. From real streaming application statistics, a long tail feature for peer duration time in the overlay is observed in [49, 50]. It means that the peer who has stayed in the overlay for a long time is likely to stay longer. Thus, the long-duration peers in LBTree are prioritized with higher probability to be moved up in the tree and to be a part of the stable infrastructure at the upper level of the tree near to the root.

2.2.7.3 Discussion

The number of substreams that can go through one connection affects how serious the negative impact will be when connection fails or peer leaves. In independent multiple-tree overlay, the impact is confined to only one substream. In shared multiple-tree overlay, a peer failure affects all substreams that the peer is forwarding and link failure affects all substreams that are delivered to the peer.

Recovery from failures are different in both cases. Since in the independent multiple-tree overlay, only one substream is allowed in one connection, the missing substreams have to be requested from a new connection which may take some time to find. In shared multiple-tree overlay, the missing substreams can be directly requested from the existing connections which can provide the data and have available bandwidth.

Since the independent multiple-tree overlay has to maintain more connections for different substreams than shared multiple-tree overlay, overhead for independent multiple-

tree structure will be relatively larger. Meanwhile, the single substream in each connection cannot fully utilize the connection bandwidth in heterogeneous network environment where bandwidth may vary widely. By adapting the number of substreams delivered through each connection, the bandwidth is better utilized when shared multiple-tree overlay is employed.

2.2.8 Multiple Overlays

The previous sections focus on the scenario where one stream is requested by every peer in the overlay. In the real P2P streaming services, there are multiple streams generated from different servers and serving different groups of users concurrently. If the streams build their own overlay independently, there will be contention for the limited network resources. Moreover, the contention for precious network resources will mean that large groups with more members are likely to be allocated more resources thereby starving the small groups.

AnySee [51] addresses the multiple-overlay problem by merging the management of overlays. All peers are grouped by a location-aware strategy [52] and build intra-overlay among the peers with the same interest first. When the stream quality is below a certain threshold, the connection exploration process is initiated from the peer and ends at the stream server traversing different overlays. The new inter-overlay connection better utilizes undiscovered network resources between overlays and improves stream quality at poor quality peers.

Another approach to prevent conflicts between overlays is to place an auction strategy as in [12] for resource allocation at each peer and guide them to act prudently in the contention. Peers have to balance their budget with the desired service quality and not obtain as much resource as possible. The strategy prevents the-winner-takes-all situation from happening and helps overlays share the available bandwidth in the network fairly.

2.3 Data Exchange Mechanism

After the overlay infrastructure of P2P streaming system is built across the peers, the stream data is able to be delivered from server to each individual peers in a relay manner. Data delivery is another important component in P2P streaming system. The efficiency of data delivery and the resilience against loss determines the quality of service received at each peer. From the prospective of who initiates the data delivery, the peers can be grouped into two main categories, namely, push and pull. When the sender peer knows the data the receiver needs, it will schedule the data chunk and stream out the data based on the scheduled order. The data loss should be mitigated through loss protection and error correction to maintain the data integrity and to minimize the snowball effect of cumulated data loss through the overlay. The following sections will discuss these 3 aspects in detail.

2.3.1 Push vs Pull Mechanisms

Figure 2.9: Comparison of Push and Pull Mechanism

In push mechanism, when the receiver joins the substream distribution tree through the connection between the sender and receiver, the sender will know the data the receiver is interested in as shown in Figure 2.9(a). As soon as the stream data belonging to the

CHAPTER 2. LITERATURE REVIEW

substream arrive at the sender, the sender will initiate the send operation and send the data to all peers in the distribution tree. The relation of sender and receiver in delivering the data is somewhat stable, unless the receiver leaves the tree or the sender stops its service to the tree.

In pull mechanism, as the connection is built between two peers, there are no explicit sender and receiver roles on either side. Peers periodically publish the available data which they have cached through the existing connections to their neighbors. As the peer compares its buffer with its neighbors', it will issue the request to the neighbors for its interested data. At that point in time, the receiver and sender roles are established. As the sender receives the receivers' request, it will respond by sending the requested data to the receivers as shown in Figure 2.9(b). Thereafter, the sender and receiver roles are dissolved. There is no stable sender and receiver relationship between the two ends of the connection, and every data delivery is initiated by the receiver.

The push mechanism is more efficient than pull in terms of delay. In push mechanism, data is immediately streamed out upon its arrival. Only a one-trip time is incurred for data delivery. In pull mechanism, from the publication of data availability to the data arrival at the receiver, there is a three-trip latency incurred. Meanwhile, the data available information is periodically published to the peers' neighbors. Hence, the delay in pull mechanism is larger than the three-trip latency as one may have to wait for the announcements of data availability before a request can be made. Moreover, the delay accumulates as the data traverses down the levels of the distribution tree. Thus, the pull mechanism incurs a longer latency at each peer than push mechanism. Data buffered by pull mechanism also tends to have larger latency jitter which requires a larger buffer in each peer. The push mechanism is widely used in tight temporal requirement scenarios and is adopted in SplitStream, CoopNet and NICE.

CHAPTER 2. LITERATURE REVIEW

Overhead is larger in the pull mechanism. In push mechanism, overhead is generated once when child peer asks to join the distribution tree, whereas in pull mechanism, overhead is recurrent. Ignoring the overhead for overlay construction and maintenance since this is not related to data exchange mechanism, and given that connection measurement and network monitoring are necessary for both two, the overhead of pull mechanism is thus much more than push mechanism.

However, since the pull mechanism is initiated from the receiver, the receiver has better knowledge about the current status of each connection. Data request can be promptly adapted to the changing bandwidth. The adaptation can be as fine as on a packet level in pull mechanism. In other words, if the bandwidth of a connection is low, the receiver can simply request for just a few packets to be transmitted and when the bandwidth of a connection is high, an entire stream can be requested. In push mechanism, since the rate of each substream is fixed, the adaptation is limited to the number substreams within one connection. The adaptation in push mechanism also takes a longer time since the child peer needs to locate a new parent with adequate bandwidth to start delivery and terminate the data delivery from the old parent with inadequate bandwidth.

The robustness for pull mechanism is also better than push, which is concluded in [53]. Since there are no explicit sender and receiver roles in pull mechanism, peers can ask for data from any connection at each round of data request. When connection failure occurs, peers can issue request to unaffected connections as usual without any additional effort. For peers in push mechanism, the peers have to perform operations to recover from substream loss. It takes a longer time than pull mechanism if no backup mechanism is in place. The characteristics of pull mechanism fit the scenario where temporal requirement is loose and network environment is dynamic or unknown. CoolStream is an example which employs pull mechanism and is deployed in the Internet where the network is unpredictable.

Table 2.2: Difference of Push and Pull Mechanisms

	Delay	Bandwidth Adaptation	Overhead	Robustness
Push	short	slow	light	normal
Pull	long	quick	heavy	good

The comparison of the push and pull mechanisms is summarized in Table 2.2.

2.3.2 Scheduling

Scheduling is another important issue in P2P streaming. When multiple delivery tasks are stacked at the sender side, there should be an order to fulfill the tasks. The common solution for P2P streaming is the First-in-First-out (FIFO), where the sending order is based on the time of receipt of the requested data. Since it leaves the scheduling to the operating system, FIFO is easily implemented and widely employed in P2P systems like NICE, SPIDER, ALMA[19–21], OMNI, etc.

Though FIFO is easy and simple to be deployed as in IP multicast, streaming through P2P overlay is not the same as IP multicast. The peer has access to more detailed information about the stream data and other peers' buffer. Hence it is possible to manipulate the order of carrying out tasks to improve the streaming service quality.

In pull mechanism, since every peer publishes its data availability, peers can use that information to schedule their data requests. Peers randomly select their interested parts in Chainsaw [35]. In CoolStream, the request is started from the scarcest first. Such local-rarest-first (LRF) solution helps to distribute the rarest data faster and increase the data availability and throughput. It improves the data delivery efficiency in CoolStream. The work in [54, 55] uses linear programming to calculate the best local request combination across multiple connections. It maximizes the throughput of the overlay and improves the stream quality at each peer.

In encoded data streaming applications, especially video streaming, SVC not only helps in bandwidth adaptation but also provides more information to improve the schedul-

CHAPTER 2. LITERATURE REVIEW

ing mechanism. Data in PALS[14], is scheduled from the basement layer data to enhance-ment layers, layer by layer, as shown in Figure 2.10(a). Its extension work [56] considers not only the data importance, but also the time importance. The horizontal ordering, vertical ordering and diagonal ordering, as shown in Figures 2.10(a), (b), (c), give preference to stream layer, playback time, and both factors respectively. The diagonal ordering which properly balances the buffering efficiency and stream quality performs the best among the three.

Figure 2.10: The stream scheduling in PALS and its extension work

Work in [57] proposes an optimal media data assignment algorithm to schedule equal sized sequential segments among multiple suppliers with heterogeneous available bandwidth. The scheduling at the sender side considers the playback time and arrival order of the data chunks at the receiver side, and schedules the data chunks to minimize the buffering delay while ensuring continuous playback. The work in [58] proposes a fixed-length slotted scheduling scheme, which also aims at minimizing the buffering delay, and scheduling variable sized and fixed-temporal length data segments.

RUBEN [59] is designed for balancing the load across multiple peers, while providing end-to-end soft real-time guarantees to multiple requests. The order of tasks is scheduled based on task urgency. Feedback of resource utilization is propagated among peers for estimating queuing latencies of tasks at remote peers. Based on the estimation, requests are scheduled and sent to the peers with the least load, and thus, have the best probability of meeting the tasks' deadlines.

The work in [60, 61] looks into the content of stream data encoded in SVC. The distortion of stream quality caused by different parts missing in delivery is calculated as in [62]. The congestion in the network is also considered when calculating the sending order as in [63]. The streaming quality is optimized at each peer in terms of peak signal-to-noise ratio (PSNR).

In our work [64, 65], we propose a solution which takes care of both the content in the stream data and the overlay structure. In our solution, packet which is the most important, requested by the largest number of peers and consume the least transmission time is given highest priority and sent out first at each peer. It proves to be the optimal order to minimize the queuing delay at each peer. Simulation results show that overall latency is reduced and more packets could be received before expiration.

2.3.3 Data Loss Protection

Due to network congestions or failure, data loss and error are unavoidable. Some recovery schemes have been proposed to protect the data integrity. In order to provide reliable data delivery service, TCP is preferred to UDP. However, due to heavy overhead in TCP and long latency for data delivery, it is not suitable for streaming application because of the stringent temporal requirement. Though the light weight UDP fits the temporal requirement, the delivery is vulnerable to data loss and error if there is no additional data protection. Retransmission is one of the easiest way to recover the missing or error data by requesting for the missing data to be delivered again. Retransmission uses relatively lesser network resource to provide the data protection.

Forward error correction (FEC) mechanism adds redundant data as error-correction code to the stream data to protect the data integrity. The partial missing data could be recovered without the need for retransmission. It saves on delivery time since the encoded data can self recover in the event of a partial loss. However, the delivery of

CHAPTER 2. LITERATURE REVIEW

extra codes consumes extra bandwidth, and the recovery ability is constrained by the coding scheme.

Hence, in unknown or lossy network environment, retransmission can save the bandwidth resources and provide better QoS. In known, less lossy network environment, a proper coding scheme can mitigate the loss effect and maintain the latency by sacrificing some bandwidth for the extra redundancy.

2.4 Membership Management

Membership management plays an important role in P2P streaming service in a dynamic environment. Embedded structured overlays make use of underlying mechanisms to construct, maintain and refine the overlay. Usually, they use some forms of distributed hashing tables (DHT). Examples of such overlays are CAN [4], Chord [66], Pastry [5], Tapestry [6]. The embedded structure is constantly updated in order to ensure that the upper overlay works properly. Since DHTs are proposed for general distributed applications, they generate substantial overhead in the maintenance of the structure. For specialized applications such as data streaming, a light weight specialized membership management scheme is necessary.

New users join the session with little knowledge about the whole system. Contacting some Rendezvous Point (RP) nodes, peers will learn some startup knowledge about the other peers. While in the session, peers periodically update their knowledge about the other peers and build connections with selected peers. By dynamically changing their positions in the overlay, it allows the overlay to operate efficiently based on its designed goals. When peers detect failure of other peers and are isolated from the overlay, they need some recovery mechanisms to re-graft themselves to the overlay and to reconnect to existing peers. Though all P2P streaming systems have all the three peer membership

management components: new peer join, overlay maintenance and peer failure recovery, there are still some differences in the implementation of the different schemes.

2.4.1 New Peer Join

In some systems [28, 67–69], servers or dedicated membership management services have complete knowledge of peers in the service session. They take full charge of new peer join process. Thus new peers could find their optimal position when they first join the service. In Dagster [69], the source peer decides which peer candidates the new peer should contact based on the new peer’s resource donation. DagStream collects peers’ QoS characteristics and maps their QoS characteristics to a prefix of their peer id. Its membership management protocol assigns peer id to every peer. New peers can look up neighbors with certain desired QoS characteristics from membership management service by examining their peer id to build a QoS-aware overlay. Though these kinds of systems provide quick startup for new peers, the server needs to maintain all peers’ information and constantly update their latest status in order to keep the information accurate during the service. It introduces significant overhead traffic and heavy load on the root server in the system.

In some other mechanisms [11, 13, 14, 30, 70], new peers are bootstrapped to some candidate peers to initiate the first contact and locate their positions through an iterative estimation. New peers in PALS will get a set of randomly selected peers from active peers in the current session. To avoid heavy load on the central server, source peer, in CoolStream [11], randomly selects a deputy peer for newly join peer after receiving its join request. New peer contacts the deputy peer and gets a list of candidate peers to initiate the overlay construction. In systems like NICE [13] and HMTP [70], new peers contact RP to get a list of the highest level peers. After examining all peers in the highest level, the new peer will contact the closest peer to get to the next level peers in that cluster.

CHAPTER 2. LITERATURE REVIEW

The new peer iteratively estimates its relative distance with peers level by level until it finds the peer at the leaf level by measuring latency between them. Although a new peer in Overcast [71] also contacts peers from the root to the leaf node like NICE and HMTP, the new comer tries to locate the peer with broad bandwidth far away from the server in order not to exhaust resources near the root node. New peer in ALM-CAN [33], registers with a random point in a 2-D coordinate space. It is assigned with an area in the 2-D space and contact adjacent existing peers to learn more about its neighborhood. New peers in [72] measure the distance with several landmarks and group the peers into a topology-aware bin based on the measurement. Peers in the same bin will be placed in a close proximity in the logical structure, like CAN, Chord. Thus, the physically close peers will be close logically.

2.4.2 Dynamic Membership Maintenance

In dynamic changing network environment, peers need to periodically probe for better sourcing peers and change their positions in the overlay in order to keep the overlays working efficiently. There are three steps to get to know better data suppliers. The first step is to have the basic knowledge of the potential peers. The next step is to estimate the availability of the potential peers. Finally, the current peer makes a decision to adjust itself to some optimal position based on the availability of the peers it examined.

To enable peers to gain knowledge of other peers, they obtain this information from their connected neighbors or RP-like nodes, and propagate it to other peers. To contact RP-like nodes is similar to the peer join process. However, it is not encouraged since the process introduces a lot of traffic in the networks and place a heavy load on the central server. Some distributed mechanisms, like SwapLinks [73], help to alleviate load on the central server and reduce traffic in network. Peers may spontaneously publish their status to their neighbors, or passively send out their status information in response to other

CHAPTER 2. LITERATURE REVIEW

peers' requests. Peers may also forward received status information about other peers to their instant neighbors to make that information known by more peers. The information may be forwarded to all peers, or forwarded within limited hops, or forwarded to some directed or random peers. To improve the QoS of overlay, some schemes [22, 74] assume that the neighbors' neighbors may also be close to the peer itself. Thus, exchanging and propagating neighbors' information help peers know their surrounding peers better. The information could be peer's latency, Round Trip Time (RTT), bandwidth, data loss rate, number of supporting peers etc. according to the needs of overlay construction algorithm. To improve robustness, some schemes [24, 25, 39] propagate information of some random peers in the session. As the random peer may not be close to the current peer, a peer can recover easily from a local peer failure by contacting such remote peers. SwapLinks considers the control load balance in the heterogeneous environment and distribute the load proportionally based on peers' capability.

After knowing the other candidate peers, peers estimate the availability of these peers based on the metrics used by the overlay construction algorithm to ensure the overlay is working efficiently. To test a connection QoS between peers, one peer may issue some data packets and observe the behavior and status of these transmission through some forms of feedbacks, or resort to some network protocols. In Overcast [71], bandwidth is measured by sending 10 Kbytes of data and monitoring the downloading time. ICMP [75] helps to estimate RTT and hops in physical network between peers. RTCP [76] provides data loss statistics in streaming services. Having the necessary information about the candidate peers, the current peer calculates their suitability to be its new data supplier. In some selfish schemes [32], peer only selects suppliers with abundant resource to maximize its service quality. In some cooperative schemes [22, 30], peer selects resources which not only benefit itself but also improve local or global performance.

Based on the estimated availability of candidate peers, peers build connections with the efficient peers, and disconnect with less efficient ones. Proxies, in OMNI [22], adjust

CHAPTER 2. LITERATURE REVIEW

their position in the overlay by swapping parents within 2 tree levels in order to smoothly improve overall latency among all peers. To prevent the swapping process from falling into regional optimization while losing sight of global efficiency, peers also probabilistically swap to some random members to achieve global latency optimization. Peers in PALS [14] periodically add another random peer to the subset of active senders. To avoid congestion of two senders sharing a common path, peers monitor the variation of both overall and individual throughput. They keep the one which improves overall throughput, and drop the one which compromises overall throughput. Peers in Yoid [68] switch parent based on observed loss rates and latencies. In Kudos, peers learn of other cluster leaders from their current cluster leader, and migrate to other cluster if the migration yields significant improvement in latency.

In some mechanisms where peers are organized in groups, peers leaving or switching from one group to another might lead to imbalance of group size among groups. Membership management load is heavy in the groups with large number of peers, which will degrade streaming efficiency in these groups. Hence it is necessary to maintain group size within a certain range. Merge and Split operations are performed where necessary during peer join and peer departure to keep the group size within a certain range and keep the overlay efficient. NICE [13] sets their group size within $[k, 3k)$, where k is an assigned constant integer. To avoid big differences in group management, a small range is preferred. However, for smaller upper bound such as $2k$, frequent join and leave in the system may incur frequent split and merge operations. Considering both reasons, NICE chooses a relatively small range with some tolerance for peer number variation. In ZigZag [27], group size is maintained within $[k, 3k]$ at each level of the tree for the same reason. Split is called when a new peer joins the group and the number of peers in that group exceeds the upper bound. The cluster leader assigns a peer in that group to be another leader in the new generated group. Peer group is split into 2 equal size

groups and managed separately by these 2 cluster leaders. The result will be sent to all peers that need to change their connections. When the number of peers in some group falls below the lower bound, the merge process kicks in. The group leader will find an appropriate group and peers in these two groups are mixed into one group within the group size limitation. One of the group leaders relinquishes its group leadership after the merge, and the other leader informs all peers in the new group about this change. In ALM-CAN [33] which makes use of Content-Addressable-Network (CAN) [4], peer will split its 2-D Cartesian coordinate space when a new peer joins nearby. When a peer leaves the session, its space is released. Neighbor peer will merge that space with its own.

2.4.3 Peer Failure Recovery

Owing to network dynamics and users' unpredictable usage behavior, peers may accidentally lose their connections with the others. The peers may be isolated from the service and could not receive data stream from other peers. Thus, recovery from peer isolation is necessary in membership management. Root server, as a wide-known peer, will exist in the system until no service is provided. It is easy for isolated peer to re-contact the root and join the service as a new peer. TBCP [77] uses such a mechanism for peer failure recovery. Some other systems utilize their robust overlay design to prevent peers from being fully isolated from the system. Each peer always has multiple peers as data stream suppliers in multiple tree overlay. Thus a single peer failure can be recovered by issuing new requests to other existing peers. Alternatively, some pre-cached inactive peers, in tree-mesh overlay, can take charge of the failed peer. However, some overlay multicast systems focus on designing an efficient transmission overlay. Peers only know its nearby neighbors. Hence, when failure happens within a local region, the peers in that region might be fully isolated as its neighbors are likely to fail at the same time. Thus, schemes

CHAPTER 2. LITERATURE REVIEW

like GoCast [39] which always gossip with some randomly selected peers from remote area is resilient to regional peer failure. PRM [24, 25] improves robustness in NICE by spontaneously and randomly forwarding data with variable probability to random peers, which facilitates data recovery.

2.4.4 Centralized vs Distributed

Membership management can be categorized into centralized or distributed. Different types of membership management schemes have different advantages, and are suitable for different application requirements.

In centralized membership management, root server, RP or dedicated service supplier takes full charge of the peer connection organization according to the design of overlay. In ALMI [15] and Dagster [69], central server constantly updates all peers with the necessary status information, calculates the most efficient overlay structure and guides peers to adjust to their optimal position. Since the central server has all information about peers existing in the system, there is no need to build extra loop prevention mechanism. Meanwhile, cheating can be easily detected in the centralized membership management by verifying all peer information at server. Compared to distributed overlay maintenance, the centralized membership management only needs one copy of peer status information to be stored at server, and do not require multiple copies to be exchanged among neighbor peers. In some centralized mechanisms, the server also takes charge of streaming data and managing peers at the same time. The server might experience heavy load and degrade service quality when the number of peers in the system grows. Even if there are RPs or dedicated service providers for centralized membership management, the scalability of the whole system, which highly depends on the capabilities of the RP or service provider, is still limited. Thus, centralized membership management is suitable for small or medium number of peers in order to provide efficient membership service.

To overcome the scalability limitation in systems with large or super-large number of peers, distributed membership management is employed. Based on the peer information cached in every distributed peer, peers spontaneously adjust their position in the overlay by themselves. Less communication with the streaming server is needed in distributed mechanism. Hence, the system can support many more peers than the centralized mechanism with less burden on streaming server. In some schemes, like Narada [30], each peer has all others' information and can find its optimal position in the overlay by itself. In other schemes like Coolstream [11], OMNI [22], every peer only holds peer information of some overlay neighbors. Hence, peers in the system need to periodically exchange information and probe new overlay path to find their optimal position. Meanwhile, in mesh structure overlay, loop prevention is necessary in membership management.

2.5 Peer Behaviors

Besides the design of overlay structure and the membership management to improve the service quality, understanding peer behavior in a P2P system will enable overlays to be designed to exploit and address the issues arising from these behaviors.

2.5.1 Peer Incentive

Peer behavior can be classified as altruism and selfish. In the altruism scenario like in Narada and OMNI, peers contribute all they have to the others. The bandwidth in the network can be fully utilized. In selfish scenario, every peer is not willing to contribute their resources to the others. In order to enable the overlay to scale, the peers have to contribute some upload bandwidth resource in exchange for download bandwidth from other peers. In systems like CoopNet, peers are assumed to contribute the same amount of resources as they receive from the others. In altruism scenario, overlay structure can expand and shrink promptly as the peer number increases and decreases. However, in

CHAPTER 2. LITERATURE REVIEW

selfish scenario, since peers are conservative in the sharing of their resources, it is hard to find the proper source provider for a newcomer. Thus, the overlay cannot adapt to the peer number changes as fast as in the altruism scenario. However, in altruism, some powerful peers are more likely to play more important roles than normal peers in the overlay. The roles for peers are hence unequal. The failure of such powerful peers will have a larger impact than an average peer. In the selfish scenario, the failure of peer may not cause a big impact since the individual peer's contribution is limited.

In both altruism and selfish scenarios, to ensure the overlay works properly, peers still cooperate with others and contribute their resources. But, in real world, there are two types of non-cooperative behavior in P2P streaming: (i) contribute little resources and provide true self-information, called free-riders, (ii) not willing to contribute resources and provide false self-information to cheat for better service quality, called white-washers [78]. Such peers are very harmful to the whole system. In order to model the negative impact of free-riders and white-washers on the service quality, work in [79] uses game theory to let peers self-justify whether the neighbor peers are good or not. The peers' behavior is modeled as repeated games based on the assumption that all peers are selfish and they all want the overlay to exist in the future. The model gives an approach to study how to encourage peers to contribute more resources.

The incentive models for P2P systems can be broadly categorized into inherent generosity, monetary payment scheme and reciprocity based scheme[80]. [81] devises a modeling framework that studies the phenomenon of free-riding in P2P systems while taking user generosity into account. The model analytically determines the resulting percentage of free riders in the system based on the distribution of generosity in the population. The finding is that if the societal generosity is below a certain threshold, then there are too many selfish peers and the system collapses. However, if it exceeds the threshold, the contribution level yields diminishing returns. In monetary payment schemes,

CHAPTER 2. LITERATURE REVIEW

users are required to pay in some form of virtual currency to get specific services from other peers. The virtual currency can be earned by contributing resources to the system. KARMA[82] proposes a general economic framework for combating free-riders in P2P systems by keeping track of resource contribution and resource consumption of each member of the system and the virtual currency used is called Karma. The third category is reciprocity-based schemes where peers maintain behavior histories of other peers and utilize this information for decision making processes. eMule[3] has been one of the most successful file distribution tool with explicit incentive mechanisms (Tit for Tat reciprocity strategy) to provide incentives to users to contribute authentic resources to the system by determining that a peer's download bandwidth is proportional to his/her upload performance.

Dagster [69] proposes a novel incentive scheme to provide better service quality to peers who contribute more resources. Nodes that donate more bandwidth have lower rejection rate and are closer to the root node of source in the overlay. This incentive will maximize the service capacity in the whole system even when peers are not willing to contribute more resource. Meanwhile, it encourages peers to contribute their resources by rewarding them with better service quality. Inspired by a performance and salary model [83], Habib and Chuang [84] propose their overlay construction scheme. The incentive is to rank the peers by their relative performance, and award those who contribute more. Users can only select equal or lower rank peers as suppliers. Hence, free-riders will be given limited options in peer selection.

Several P2P systems integrate reputation scheme to minimize the negative effects of white-washing behavior from malicious peers. Lee et. al. [26] adds reputation information into their previous NICE system. With the help of reputation information, peers can calculate how much the others can be trusted. The reputation information is stored in a fully distributed manner and can be updated by the history records of exchanging data.

CHAPTER 2. LITERATURE REVIEW

Based on the degree of trustworthiness, peers decide how much resources they should contribute to the others. Hence, communication between highly trusted peers will be at high speed, while communication between untrustworthy peers will be at low speed. Good users in the system will eventually form cooperating groups, and not lose large amount of resources to malicious users.

Another kind of malicious peer in the P2P streaming system are those who destroy stream data integrity. Work in [85] proposes a selective validation protocol by verifying the abstract of stream data, which reduces the integrity damage probability while maintaining the overhead at a low level.

2.5.2 User Behavior

Besides the study of peer incentive, user's behavior is also an important means to understand how the overlay performs and how to improve its performance. Real world studies about P2P streaming systems are conducted in [49, 50, 86]. A long tail characteristics for peer duration time is discovered. Based on the observation, the old peers are more likely to stay in the overlay for a longer time. Thus, it is safer to move such peers to the upper level of the distribution tree than a new peer[48].

Several other studies [87–90] have been done on modeling user behavior in P2P system in order to understand the impact on the overlay performance. In [87], the authors analyze the network behavior by examining the workload of the whole P2P system. The two phases are defined, transient phase and steady phase. Yang et al. [87] use branching processes to model data replication. Work in [89] uses stochastic fluid model to examine the performance of a churnless system. Tu et al. [88] analyzes the media streaming service in a dynamic environment. Queueing model is employed in [90] which considers user behavior as individual jobs in the queueing system.

The users are also likely to join the overlay in a crowd before some important event is scheduled to happen [86]. The modeling of user behavior [91] in bursty pattern such

CHAPTER 2. LITERATURE REVIEW

as peer churn helps to find the critical component to improve the overlay performance when flash crowd happens.

Another trend in P2P is the idea of exploiting natural connections between humans in large scale social networks. Social based networks possess several characteristics which can address issues in P2P system namely the decentralization of functionality, the availability of peer resources for operational availability of the P2P system and maintaining system integrity to achieve trust among peers. Social groups fit the decentralized P2P system well as it is by nature distributed as communication is mostly localized amongst group members. Moreover, social incentives such as awards and social recognition could stimulate peers to leave their P2P software running for longer periods and contribute resources, thus improving the overall availability of the P2P system. System integrity and achieving trust amongst peers can be solved with a social based network as users can actively help to clean polluted data and can select trustworthy representatives in their social proximity. To date, methods based on social clustering in P2P networks have been proposed for content distribution[92], user communities formation[93], collaborative service provisioning[94] and Tribler P2P file-sharing system[95]. [92] presents a system which uses knowledge discovery techniques for overlay network creation by automatically clustering users based on their preferences. The system enables content location and improves the performance of content sharing. In [93], a simple general purpose system is proposed. Both [92] and [93] group peers based on the similarity of their keyword searches. Authors give evidence on how their system can be used to form and maintain communities of users. An extensive experimental analysis of several collaborative filtering methods is given in [94]. Tribler exploits social phenomena, namely, friendship and trust to address the aforementioned P2P challenges, to automatically build a robust semantic and social overlay on top of BitTorrent.

2.6 Summary

This chapter surveys the milestone P2P overlay proposals documented in the various literatures and classifies them according to their topology designs. Tracking the data streaming path from source to receiver, the discussions first detail the pull versus push mechanisms, followed by the various data scheduling techniques and end with the common mechanisms adopted to protect against data loss as well as all the relative merits. The construction, maintenance, refinement and recovery from failure of the various P2P overlay systems are also discussed. A brief insight into the different peer behaviors and the issues involved are provided as the former can be exploited in the design of an efficient P2P data streaming system. The different techniques have been designed to cater to their different goals with their respective strengths and weaknesses.

Chapter 3

Content and Overlay-Aware Scheduling

Since every user desires short latency in order to enjoy better service quality in real-time streaming applications, minimizing the latency based on available network resources is very important. From the discussion in Chapter 1, the adaptation mechanisms [18, 22, 23, 30] of these overlays are somewhat limited in operation when there are persistent changes in the network.

To deal with the temporary changes in fluctuating network environment, we propose a data packet scheduling algorithm to improve transmission efficiency of the streaming system in this chapter. It is to be noted that data transmission over Internet on best effort basis will inevitably lead to jitter and delay. Hence data streaming via P2P systems also faces the same problem. Our proposal does not take jitter into consideration as the problem can be efficiently addressed with the use of a jitter buffer at the receiver. This is consistent with most of the current work in P2P streaming. The jitter buffer caches the received packets and reorders out-of-order packets for playback. However, the delay will be accumulated along the data path of overlay and if the delay is large enough such that the stream data arrives after the expiry time, the data will be discarded, leading to peers losing some of the data. Hence the motivation for a scheduling scheme to improve

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

streaming efficiency. The issue on re-transmission of loss packets will be addressed in Chapter 4.

Our scheduling algorithm distributedly prioritizes and sends the data packets which are the most important to the whole system first. By scheduling the sending order of data packets queued at each peer, it improves the overall data latency. Unlike traditional scheduling algorithms [14, 56, 57, 60, 61], our algorithm considers both content importance pertaining to a single stream and the overlay structure. It helps more peers receive important data than traditional methods and avoids wasting network resources while frequently switching peer connections. The algorithm serves to provide an optimal ordering solution to minimize overall latency based on overlay structure information. With the help of content-aware weighting scheme, the scheduling algorithm also improves the streaming quality at the peers under a dynamic and challenging network environment with negligible algorithm overhead. Even in the event that the weighting scheme does not exactly map to the actual stream codec, our scheduling algorithm still performs better due to its abilities to provide more data to the peers and to increase the possibility of data recovery.

3.1 Problem Formulation

3.1.1 Streaming Data Latency Modeling

From analyzing the procedure of data distribution through P2P overlay, the data latency experienced at each peer is from the time a packet is generated at the root peer to the time it is received by the destination peer, which is also equal to the delay along the data forwarding overlay path. The latency for any packet, p_k , is the sum of forwarding delays along the path of the multicast tree in the overlay:

$$\text{Latency for } p_k = \sum_{e_{s_{p_k}^{(i)}} \in \{\text{path of } p_k\}} P_{e_{s_{p_k}^{(i)}}}^{p_k} \quad (\text{Eq. 3.1})$$

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Here, $e_{S_{p_k}^{(i)}}$ is the link of the i th hop for packet p_k as it traverses the overlay from its source to its destination and belongs to the set of overlay links, E . $P_{e_{S_{p_k}^{(i)}}}^{p_k}$ is the delay of packet p_k from one peer to another peer through link $e_{S_{p_k}^{(i)}}$.

The overall latency is the sum of latencies for all data packets which have arrived at each peer as shown in Eq. 3.2:

$$\text{Total Latency} = \sum_{p_k \in \{\text{packets in the system}\}} \sum_{e_{S_{p_k}^{(i)}} \in \{\text{path of } p_k\}} P_{e_{S_{p_k}^{(i)}}}^{p_k} \quad (\text{Eq. 3.2})$$

$$= \sum_{e_i \in E} \sum_{p_k \in \{\text{packets pass through } e_i\}} P_{p_k}^{e_i} N_{p_k}^{e_i} \quad (\text{Eq. 3.3})$$

From the equation transformation, the overall latency of data packets is equal to the sum of delay generated through each peer connection link ($P_{p_k}^{e_i}$), times the number of peers requesting for the data packet p_k through the link e_i ($N_{p_k}^{e_i}$), as in Eq. 3.3. Note that the $N_{p_k}^{e_i}$ is not the number of times, the packet p_k is sent through link e_i . In P2P overlay, data is sent only once through peer connections unless the data is lost or corrupted. Then the data will be replicated and distributed to the descendants. As the delay of one data packet passing through one link will also be experienced by all other descendants requesting for the same data through that link, the resultant generated delay contributing to the overall latency is thus the delay through one peer connection, $P_{p_k}^{e_i}$, times the number of peers experiencing the delay along the data path, $N_{p_k}^{e_i}$.

Note that in the transformation equation Eq. 3.3, the overall latency is expressed as the sum of latency experienced by each data packet over each link traversed. It is difficult to design a scheduling algorithm for every packet traversing multiple links along its path. The transformation helps us to analyze the latency from the perspective of the links between the peers, which makes scheduling at each peer distributedly possible.

3.1.2 P2P Delay Components

When a data packet is being forwarded from one peer to another peer, delay is encountered at the following three stages:

Queueing Time (Q): When a data packet arrives at an intermediate peer, it will be queued for some time, if there are packets occupying the network interface for transmission, or there are packets queueing ahead of it.

Transmission Time (T): When the data packet gets the opportunity to be sent out, it takes some time slots to be transmitted.

End-to-End Delay (D): After the data packet is sent into the network, it incurs end-to-end delay from one peer to its destination peer in this hop.

The delay for every packet through a peer connection is the sum of the delay as mentioned above:

$$P_{p_k}^{e_i} = Q_{p_k}^{e_i} + T_{p_k}^{e_i} + D_{p_k}^{e_i} \quad (\text{Eq. 3.4})$$

Incorporating Eq. 3.4 into Eq. 3.3, the generated delay of packets passing through a single P2P link, e_i , contributing to the overall delay is expressed as:

$$\text{Generated Latency of } p_k \text{ on Peer Link } e_i = \sum_{p_k \in \{\text{packets pass through } e_i\}} (Q_{p_k}^{e_i} + T_{p_k}^{e_i} + D_{p_k}^{e_i}) N_{p_k}^{e_i} \quad (\text{Eq. 3.5})$$

As illustrated in Figure 3.1, packet p_k has to wait for all packets ahead of it to be sent out ($Q_{p_k}^{e_i}$), then incurs transmission time ($T_{p_k}^{e_i}$) followed by End-to-End delay ($D_{p_k}^{e_i}$). This delay through link e_i will be experienced not only by peer B but also by all the other $N_{p_k}^{e_i} - 1$ peers requesting data from B via that link.

The end-to-end delay between two peers is determined as the overlay is constructed and cannot be directly controlled by end hosts. Since we do not want to frequently switch

Figure 3.1: Delay of Packet p_k

the connections which introduce extra traffic to the network, it cannot be minimized by scheduling. The transmission time is determined by the physical profile of the peer accessing the network and the data packet size, which cannot be changed by scheduling either. For data packets queueing to be sent out, the change in the sending order will certainly impact the generated delay in the overall latency because of their different delays at the 3 stages and the different number of peers requesting for the packets. Thus, to minimize the overall latency, we can change the sending order of data packets by scheduling to manipulate the queueing delay of the data at each peer.

The queueing time for every packet at a peer is the sum of the time it has already waited and the time it will wait. Suppose M packets are queued in a peer with sending order of $\{p_1, p_2, p_3, \dots, p_M\}$, as illustrated in Figure 3.2. Each packet p_k has waited for $C_{p_k}^{e_i}$ from its arrival time to the current time at the peer. It has to wait for all the data packets in front of it to be sent out into the network, which is the sum of transmission time of all those packets. The packet transmission time can be estimated based on the link bandwidth and packet size. The queueing time for packet p_k at the peer connection

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.2: Example of Queueing Time with Scheduling

is:

$$Q_{p_k}^{e_i} = C_{p_k}^{e_i} + \sum_{j=1}^{k-1} T_{p_j}^{e_i} \quad (\text{Eq. 3.6})$$

Thus the total generated delay for data packets along one peer link with the sending order of $\{p_1, p_2, p_3, \dots, p_M\}$ is transformed to:

$$\text{Total Latency on } e_i = \sum_{k=1}^M (C_{p_k}^{e_i} + \sum_{j=1}^{k-1} T_{p_j}^{e_i} + T_{p_k}^{e_i} + D_{p_k}^{e_i}) N_{p_k}^{e_i} \quad (\text{Eq. 3.7})$$

3.1.3 Weighting Packets by Content Importance

Another important point to consider is that data packets deliver different content in the stream. For example, in video streaming services, the data in I frames is in charge of recovering the entire Group of Pictures (GOP) which is more important than the data in P frames and B frames. The data in the enhancement layer is dependant on the receipt of the data in the basement layer. Hence the content inside each data packet, p_k , should be weighted based on its importance to the data stream, denoted as W_{p_k} . The overall latency needs to take the importance of data packets into consideration. The generated delay for every packet passing through the peer connection is weighted by the weight factor. The expression in Eq. 3.7 is thus modified to the weighted delay by adding a

weight factor as follows:

$$\text{Weighted Total Latency on } e_i = \sum_{k=1}^M (C_{p_k}^{e_i} + \sum_{j=1}^{k-1} T_{p_j}^{e_i} + T_{p_k}^{e_i} + D_{p_k}^{e_i}) N_{p_k}^{e_i} W_{p_k} \quad (\text{Eq. 3.8})$$

Eq. 3.8 shows that every single packet may generate different delay during transmission over one hop in the overlay. In some general scheduling schemes, such as LIFO or NICE proposed by [13] and [25], data packets are treated equally or only based on their arrival or playback time. Important packets will be delayed for a long time until less important packets in front of them are sent out. Obviously, it is not an optimal way for data delivery. To provide service for large number of participants, it is even worse because the waiting time in a single link could also be experienced by all peers requesting data through that link. The deeper the multicast tree, the more is the delay experienced by the leaf peers. From our previous discussion, data packets which have different importance and are requested by different number of descendent peers will contribute different delays to the overall latency. Thus, they should not be treated equally. With the help of extra information to classify the data, such as the number of peers requesting for the data through a certain peer, the data generation time, time taken to send out the data, the importance of the data to the stream, etc., data can, therefore, be differentiated by how important they are and be treated differently. Such extra information could be propagated during overlay construction or update process with a small overhead to improve data delivery latency.

3.2 Algorithms

3.2.1 Scheduling Factor for Prioritization of Packets

Suppose at some point in time as shown in Figure 3.3(a), data A and B are queued at peer 1. They have the same timestamps and playback time. Peer 2 requests both of them from peer 1. Peers 3 and 4 request data A and B respectively from peer 2. For

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.3: Scheduling in Single Connection

peer 2, the order of receiving A and B makes no difference. However, peer 3 and peer 4 will experience different arrival times for the data they request. If there are 100 other peers requesting data A from peer 3 and 10 peers requesting data B from peer 4, peer 1 should send out data A before data B to benefit more peers in terms of shorter waiting time.

In another scenario as in Figure 3.3(b), if data A needs 100ms to be sent out and data B needs only 5ms to be sent out at peer 1, data B should be sent out first so that the 3 downstream peers incur minimal overall waiting time for their requests.

In another scenario, if data A is more important in improving the stream quality for the rest of the peers than data B, as in Figure 3.3(c), data A should be sent out first.

From these three observations, the important data packets to the entire P2P system is the data which is needed by more number of peers, more important to the quality of data stream and occupy less transmission time. Based on this, we propose a scheduling factor, which provides data possessing such features with a higher priority in the queue and so that they are sent out earlier. The scheduling factor is:

$$\frac{N_{p_k}^{e_i} W_{p_k}}{T_{p_k}^{e_i}} \quad (\text{Eq. 3.9})$$

Scheduling the sending order of data packets in non-increasing order based on this prioritization algorithm within every peer connection helps more peers receive better quality stream data at earlier time. Thus the total weighted delay which is added to the overall latency is minimized. The optimality of our scheduling algorithm is verified using a proof by contradiction as follows:

3.2.2 Optimization Proof

Theorem 3.1 *The packets queueing at each peer sorted in non-increasing order of Eq. 3.9 achieve minimum weighted delay.*

Proof: Suppose the proposition “sorting packets by non-increasing order of Eq. 3.9 could achieve minimum weighted delay” is not true. Then another sending order of M packets, $p_1, p_2, \dots, p_k, p_{k+1}, \dots, p_M$, which can achieve minimum weighted delay, will not follow the non-increasing order in Eq. 3.9. There must be two consecutive packets p_k and p_{k+1} to satisfy

$$\frac{N_{p_k}^e W_{p_k}}{T_{p_k}^e} < \frac{N_{p_{k+1}}^e W_{p_{k+1}}}{T_{p_{k+1}}^e}$$

Based on (Eq. 3.8), the total generated weighted delay of the optimum sequence is:

$$S_{optimum} = \sum_{i=1}^M (C_{p_i}^e + \sum_{j=1}^{i-1} T_{p_j}^e + T_{p_i}^e + D_{p_i}^e) N_{p_i}^e W_{p_i}$$

If the order of p_k, p_{k+1} is switched to $p_1, p_2, \dots, p_{k-1}, p_{k+1}, p_k, p_{k+2}, \dots, p_M$, the transposition only affects the generated delay of packets p_k and p_{k+1} . From Eq. 3.8,

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

the total generated weighted delay of this non-optimum sequence is:

$$\begin{aligned}
 S_{non-optimum} = & \sum_{\substack{1 \leq i \leq M \\ i \neq k, k+1}} (C_{p_i}^e + \sum_{j=1}^{i-1} T_{p_j}^e + T_{p_i}^e + D_{p_i}^e) N_{p_i}^e W_{p_i} \\
 & + (C_{p_{k+1}}^e + \sum_{j=1}^{k-1} T_{p_j}^e + T_{p_{k+1}}^e + D_{p_{k+1}}^e) N_{p_{k+1}}^e W_{p_{k+1}} \\
 & + [(C_{p_k}^e + \sum_{j=1}^{k-1} T_{p_j}^e + T_{p_{k+1}}^e) + T_{p_k}^e + D_{p_k}^e] N_{p_k}^e W_{p_k}
 \end{aligned}$$

The difference in weighted queueing delay between the optimum order and the changed non-optimum order is

$$\begin{aligned}
 S_{optimum} - S_{non-optimum} &= \sum_{i=k, k+1} (C_{p_i}^e + \sum_{j=1}^i T_{p_j}^e + D_{p_i}^e) N_{p_i}^e W_{p_i} \\
 &\quad - [(C_{p_{k+1}}^e + \sum_{j=1}^{k-1} T_{p_j}^e + T_{p_{k+1}}^e + D_{p_{k+1}}^e) N_{p_{k+1}}^e W_{p_{k+1}} \\
 &\quad + (C_{p_k}^e + \sum_{j=1}^{k+1} T_{p_j}^e + D_{p_k}^e) N_{p_k}^e W_{p_k}] \\
 &= T_{p_k}^e N_{p_{k+1}}^e W_{p_{k+1}} - T_{p_{k+1}}^e N_{p_k}^e W_{p_k} \\
 &= T_{p_k}^e T_{p_{k+1}}^e \left(\frac{N_{p_{k+1}}^e W_{p_{k+1}}}{T_{p_{k+1}}^e} - \frac{N_{p_k}^e W_{p_k}}{T_{p_k}^e} \right) > 0
 \end{aligned}$$

It shows that the weighted generated delay of optimal order is larger than the non-optimal one. The optimal order based on the assumption which does not obey the non-increasing order of Eq. 3.9 cannot generate the minimum weighted delay. Thus scheduling the data packets in non-increasing order of Eq. 3.9 achieves the minimum weighted delay.

3.2.3 Implementation of Algorithm

In the implementation, when the data packets arrive at each peer, they will be inserted into each forwarding connection. In each connection, the queueing packets are inserted

based on the priority calculated by the scheduling algorithm. When the peer connection is idle, the first packet at the head of queue will be sent out. The process is as follows:

Algorithm 1 Procedure Peer_Recevie_Packet(p_i)

```

if  $p_i$  miss the playback time then
    Discard  $p_i$ 
else
    Add  $p_i$  to peer's display buffer
    Extract layer information,  $layer_j$ , from  $p_i$ 
    for all child peer connection do
        if child peer requests for  $layer_j$  then
            Calculate the priority of  $p_i$ 
            Insert  $p_i$  to the peer connection queue based on  $p_i$ 's priority
            if the peer connection is idle then
                Get the first packet from peer connection's queue and send out
            end if
        end if
    end for
end if
end if

```

3.3 Content-Aware Weighting Scheme

In streaming applications, the stream data always introduces some compression schemes in order to support data intensive application under resource limited environment. Especially in video stream, consecutive moving pictures are encoded into GOPs to save storage and network resources. The importance of the information may not be evenly distributed in the stream data. Loss of important data such as I frames in video streaming affects the image quality of the entire GOP, while loss of less important data from B frames or enhancement layer only partially affects the quality of a single frame. Thus, it is necessary to prioritize the data packets according to their contribution to the quality of the stream and to send the important data earlier.

Besides the different information delivered inside I, P and B frames in a video stream, there is also dependency information between the consecutive frames within one GOP.

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.4: Content-Aware Weighting Scheme

Figure 3.5: Content-Aware Weighting Scheme suitable for H.264

For example as in Figure 3.4, I frames can be decoded independently. The decoding of P frames rely on the successful decoding of previous I frames or P frames within the GOP. The decoding of B frames is based on two adjacent I or P frames in the video sequence. The priority of each frame is based on the number of frames the decoder needs to rely on it. Since I frames take charge of successful decoding of frames within the whole GOP, the priority of data packets in I frames is the number of frames in that GOP. Packets from P frames at the early part of GOP, which guarantee the decoding of P and B frames in the later part of GOP, have higher priority. Such weighting scheme is used in [60, 61], where the streaming codec is explicit.

Likewise, in H.264 video coding scheme, the frames are encoded in I and P frames with dependent relation as in Figure 3.5. The success of P frame recovery between 2 consecutive I frames is dependent on the successful recovery of all I and P frames preceding it. Thus, the setting of weight for each frame depends on the number of frames whose recovery is dependent on the current frame. For example, since the successful recovery of P_{12} in the stream only affects its own quality, the weight of P_{12} is set to 1. Since the successful recovery of I_1 affects the quality of itself and all the following P frames from P_2 to P_{12} , the weight of I_1 is set to the number of frames it affects, which is 12.

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.6: An example of content and overlay-aware scheduling

In our scheduling algorithm, we provide the weighting factor W_{pk} , which helps to differentiate the importance of data to the stream. Only one byte overhead for each data packet is enough for marking the importance of data packet to the stream data. There are also some other rate-distortion optimization weighting schemes, like [96], which needs to calculate the optimal combination of data packets in order to minimize the quality distortion under channel error and inter-frame propagation. For simplicity, we use the weighting scheme set out in Figure 3.4 in our experiment, similar to [61]. The other optimization weighting schemes can also be incorporated into our scheduling algorithm since they all differentiate data importance by assigning weight to the data packets.

Unlike scheduling based only on the weight of data packets, our scheduling algorithm also considers the P2P network structure and the available resources. For example as in Figure 3.6, the video stream is divided into 3 substreams and distributed across 3 multicast trees. Peer P requests substreams 1 and 3 from peer S. Peer A and Peer B request substreams 1 and 3 from peer P respectively. At some point in time, one whole

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

	· · ·	P	B	I	B	P	B	P	B	P	B	P	B	P	B	I	B	P	· · ·
Correct	· · ·	3	1	13	1	11	1	9	1	7	1	5	1	3	1	13	1	11	· · ·
Mismatch-1	· · ·	2	1	4	1	2	1	2	1	2	1	2	1	2	1	4	1	2	· · ·
Mismatch-2	· · ·	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	· · ·

Figure 3.7: Mismatch of Data Weighting Scheme With Video Codec

GOP (suppose frame 1 to frame 12) is queued at peer S. For scheduling based only on the weight of each packet, the sending order will be $\{I_1, P_3, P_7, P_9, B_4, B_6, B_{10}, B_{12}\}$. Assume the bandwidth is congested and the available bandwidth is reduced to 75%, which could only support 1.5 substreams between peer connection S and P. Under such condition, packets B_{10} and B_{12} are likely to be dropped. Then peers A, B and their descendants in the small cloud and big cloud could only receive 0.75 substream of their requests, due to the loss of one packet out of four in each substream.

On the other hand, based on our scheduling algorithm, the sending order is not only based on the content within each packet, but also the contribution of each packet to the downstream overlay structure. According to Eq. 3.9, the sending order is $\{P_3, I_1, P_9, P_7, B_6, B_{12}, B_4, B_{10}\}$. When the same congestion happens as in the previous discussion, packets B_4 and B_{10} are dropped. The stream quality at peer P is unaffected by the loss of relatively unimportant B_4 and B_{10} . Moreover, peers in the big cloud is able to receive the full substream, while peers in the small cloud receive half substream. The average stream quality received by the descendants of peer P is 0.88 substream of their requests, which improves the stream quality by 13% compared to scheduling based only on weight.

In some cases, weighting scheme cannot exhaust all the combinations of data packets to calculate the exact distortion of each data packet under different conditions. In other cases, video codec does not exactly correlate the frame dependency to the weighting scheme. Normally, the weighting scheme assigns a general weight based on individual importance, such as Mismatch-1 in Figure 3.7. In some other cases, P2P overlay structure

is designed for universal usage, where different weights for different data is not supported during streaming, such as Mismatch-2 in Figure 3.7. In such case, the mismatch between the weighting scheme and the streaming data will highly degrade the stream quality. Thus, scheduling based only on weighting scheme is fragile under an inaccurate weighting scheme which does not correspond to the prevailing frame dependency of the video codec.

By considering the scheduling under the scope of the entire P2P overlay structure will compensate for the inaccuracy of the weighting scheme. Our scheduling algorithm sends the data which is requested by most number of peers first. Correspondingly, peers are likely to receive more data at an earlier time. It increases the probability of successful recovery of the data they received.

3.4 Evaluation

3.4.1 Simulation Setup

We test the performance of our scheduling algorithm in a comprehensive simulated environment. Since our scheduling algorithm is at each peer connection which can be implemented in application level, we randomly generate flat peer topologies by BRITE¹ to simulate the overlay structure with little knowledge of the substrate network. Every peer connects to at least 5 neighbor peers. To simulate the heterogeneous network environment, the bandwidth of peer links varies from 80kbps to 400kbps. In our experiment, we also simulate the dynamics in the network. The dynamic fluctuation and congestion is performed by introducing a "fluctuation ratio" between each peer connection to simulate the changing bandwidth. The bandwidth is randomly fluctuated within the following range:

$$[1 - \text{fluctuation ratio}, 1 + \text{fluctuation ratio}] \times \text{estimated bandwidth}$$

¹Brite: An approach to universal topology generation. <http://www.cs.bu.edu/brite/>

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

The larger *fluctuation ratio* indicates more drastic bandwidth fluctuation between peer connections through the overlay. Peers monitor their peer connections and periodically update the bandwidth, which in this case, is at every second.

Nowadays, there are different video codecs for media streaming application, such as H.263, H.264, MPEG-4, SVC, etc. They adopt different methods to recover from data loss and thus present different stream quality. Intuitively, in every codec, as long as more data is received, the stream quality will be better, and hence the perception quality from the users. Thus, in order to show the ability of our scheduling algorithm to be independent of specific codecs, we use synthetic stream data in the simulation. The data quality is defined as the ratio of recovered data before their expiration time over the original stream data. This data quality definition is commonly adopted in research work on P2P media streaming. It is used to show the quality improvement of our scheduling algorithm under different scenarios.

One video is streamed at a rate of 400kbps through the P2P overlay. The GOP structure of the video is the same as Figure 3.4 with a frame rate of 25fps. The stream is divided into 5 substreams and distributed over 5 multicast trees respectively. To examine the feasibility of our scheduling algorithm, we implement it under different overlay conditions, such as Minimum Spanning Tree (MST), Shortest Path Tree (SPT) and Maximum Bandwidth Sum Tree (MBST) [45]. The multicast trees are independently created under the same metrics. Thus, the multicast trees may overlap at some paths because of the heterogeneity in the topology. The overlapped paths in the multicast trees share the same peer connection for streaming data. In order to provide some insight into the physical data quality of the delivered data stream in terms of Peak Signal to Noise Ratio (PSNR) as well as to verify that the results from the real video data stream are consistent with those from the synthetic data stream, real video encoded in MPEG-4 simple profile and H.264 formats are also used in the simulation.

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

UDP is a light weight and efficient transport protocol. It also provides the option for further extension like Real-time Transport Protocol (RTP) which adds overhead to facilitate real-time stream delivery. To simulate a generic environment for the media streaming service through overlay network, UDP is employed to deliver the stream data.

In our simulation, we compare the performance of our scheduling algorithm (denoted as **Algo**) with 5 other commonly used scheduling algorithms:

LIFO The last received packet will be sent out first.

Numb Stream packets are ordered based only on the number of descendants requesting for the data. The most requested packet will be placed at the front of the queue.

NICE The last received packet will be added at the end of the queue. It is the same as first in first out ordering, which is commonly used in tree structure overlay, like NICE [13] and [25].

Rand Packets are sent out randomly.

Weig Stream packets are ordered based on their importance in the stream. The most important packets will be placed at the front of the queue. This scheduling mechanism is widely employed by video and other prioritized streaming applications through P2P overlay, such as CoDio by [63] and [60].

As for simulation of control overhead, it is detailed in the following subsection.

3.4.2 Overhead of Proposed Scheduling Algorithm

The control overhead in the simulation is set at one packet per 10 seconds between parent and child peers. It includes the control information, such as packet arrival time, packet loss ratio and stream quality which is mainly used to calculate the available bandwidth for

congestion control. These overheads are required by all the algorithms being compared in the experiments and is thus not unique to our proposed scheduling algorithm.

The only additional control information specific to our algorithm is the updates on the number of peers supported by each peer in each substream. A total of 5 substreams are simulated as mentioned in the previous subsection. For each substream, each peer sends out 4 Bytes of number-of-peers-supported information every 10 seconds. With 5 substreams, this control information from each peer amounts to (4 Bytes x 5 substreams), which is 20 Bytes per 10 seconds or 16 bps. Comparing with a 400kbps stream rate, this overhead attributed directly to our proposed scheduling algorithm is negligible.

3.4.3 Experiment Results

Experiments have been performed on overlays for 100, 200, 400, 800 and 1600 peers and simulated for 1000 seconds. The results show similar trend for different group sizes. The trend still stands after the 1000 seconds simulation time. For purpose of illustration, we only show the results of the 1600-peer overlay in 1000 seconds.

3.4.3.1 Latency Performance

As the fluctuation increases in the network, the data packets are more likely to meet congestion along the overlay path. Thus, they will experience more queueing delay at each peer and the average latency increases. Considering that the data packets deliver content at different importance level, we evaluate the average latency by weighting the content importance factor against the latency of each received packets.

In Figure 3.8(A), the curves show an upward trend as the fluctuation increases in all the scheduling algorithms in the MST overlay structure. However, by scheduling in our algorithm, the increase is much less than the other five. In our algorithm, the data which is requested by more number of descendant peers will be sent out earlier. Meanwhile, the more important data will also be received at an earlier time. Thus, the latency is

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.8: Average Latency under Different Fluctuation Conditions and Overlay Structures

Figure 3.9: Average Data Latency for Different Levels in the Multicast Trees

reduced compared to the other algorithms under the same condition. Figure 3.8(B) and (C) show similar results for average latency in both MBST and SPT overlay structures. It shows that our scheduling algorithm can perform better than the other commonly used scheduling algorithms in terms of average latency under different overlay structures and different network conditions.

Figure 3.9 shows the average latency of data arrival at the different levels of the multicast trees at a fluctuation ratio of 0.6 with the corresponding peer number distribution at each level shown in the histogram below. To eliminate the big variance caused by a

Figure 3.10: An Example of SPT Structure

small number of samples which cannot reflect the true trend, we only show the average level latency for levels with over 100 peers. Due to different overlay structure, the depth of each overlay is not the same. As shown in Figure 3.9, the latency increases as data traverses more levels in the multicast trees. However, since our scheduling algorithm minimizes latency at each peer, the increase is smaller at each level in the multicast tree than the other algorithms. Thus, the latency plot at each level in our scheduling algorithm is flatter than the others. In other words, since our algorithm optimizes the sending order at each level, the performance improvement will be more significant as more peers join the system resulting in more levels in the multicast trees.

The only exception is scheduling based on the number of peers ("Numb"). As the Numb scheduling only satisfies those with more number of peers' request first, peers having more descendant peers will be served first. Intuitively, this will imply that leaf peers will experience longer latency. One would thus expect that in Figure 3.9, all the curves will slope upwards towards higher latency as the level of tree increases. However, the Numb scheduling under SPT shown in Figure 3.9(C) deviates from this trend as the nodes in the higher levels actually have less average latency. This can be explained by the fact that nodes are clustered close to the roots in the SPT. For a given level in SPT, Numb scheduling will prioritize scheduling to those nodes with more descendants compared to the individual leaf nodes at the same level.

Take for example, in Figure 3.10, peer S will send requested data from peer A via peer P giving higher priority to Peer A's request since the data requested by A is also needed by peers E and F. Peers B, C and D have to experience some delay for their own needs. As soon as peer A receives the data, it will distribute to peers E and F immediately. At the same time, peers B, C and D begin to receive their requested data. The latency experienced by peers B, C and D may be larger than peers E and F. Hence when the average latency is computed for this given level n , it is larger than the average latency in the higher level $n + 1$. However, in terms of the overall average latency shown in Figure 3.8, ours still performs better than scheduling based only on peer numbers.

3.4.3.2 Stream Quality Performance

For some real-time streaming applications, it is important to provide timely information before an expiration time. The expiration time in current P2P streaming applications, like SopCast[97], PPStream and PPLive, varies from 10 seconds to several minutes. Since the scenario in our simulation is a stable overlay, the scenario is more ideal than the real world case. Thus, we set a more stringent expiration time at 10 seconds and investigate the ratio of data that arrives on time.

When more stream data is recovered at each peer, the more data a peer will see, and thus the better will be the peer's perception quality. To show the improvement of our scheduling algorithm in a general media streaming scenario, we use the ratio of data recovered before their expiration time over the total amount of stream data as the measurement of service quality. The data packets which miss the expiration time will be discarded.

Average stream quality evaluates the average portion of data that could be presented before the expiration time for all peers under different conditions and overlay structures. Since network fluctuation increases the data queueing delay at each peer along the overlay

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.11: Average Stream Quality under Different Fluctuation Conditions and Overlay Structures

path, data is more likely to miss the expiration time on its way. Similarly, experiments with different overlay group size show similar trends. Hence we choose the 1600-peer topology to show the variation in stream quality under different situations. Results in Figure 3.11 show that stream quality decreases as fluctuation increases. However, our scheduling algorithm shows the least reduction in stream quality compared to the others. The reason is that our scheduling algorithm sends the data which is requested by more number of peers first. Thus, peers will receive more data and increase the possibility of data recovery if any encryption or encoding scheme is used. The more important data packets will also be sent out at an earlier time which reduces their latency along the overlay path and increases the possibility of successful recovery.

The results in Figure 3.12 show the average PSNR of real video streams encoded in MPEG-4 and H.264 formats under different scheduling algorithms in SPT overlay structure. The average PSNR performance in Figure 3.12 and the average stream quality performance in Figure 3.11 show that the more stream data received at peers, the better the average PSNR. The improvement of our scheduling algorithm still holds when streaming real video data encoded under different coding schemes. Since the average PSNR results and the stream quality results are consistent, the rest of the results will

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.12: Average PSNR with Different Video Coding Schemes in SPT Overlay Structure

Figure 3.13: Stream Quality Distribution in Different Overlay

continue to be shown in terms of stream quality as per the many research work in P2P media streaming.

Figure 3.13 shows the distribution of peer stream quality which varies with network fluctuations. As the fluctuation increases, the peers will experience data loss because of increasing delay through the overlay and the breaching of expiration time. Thus, the stream quality variation spans a wider range when the fluctuation increases.

Since data will experience longer delay, it is more likely to miss the expiration deadline

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

and be discarded on its way. Thus peers close to the bottom of the multicast trees experience higher data loss. The minimum peer stream quality decreases with increasing fluctuations. In the case of scheduling based only on weight, under high fluctuations, it will delay a lot of less important packets requested by more number of peers. These packets are more likely to miss the expiration time. Thus, the minimum stream quality of weight scheduling is worse than our scheduling algorithm. Since the other algorithms do not make use of extra information to improve the stream quality, the performance is hence much worse than our algorithm in terms of the minimum stream quality.

From the results, the worst stream quality in our algorithm is more resilient to network changes than the worst stream quality provided by the others. Though our scheduling algorithm gives priority to fulfill the request from more number of peers, it does not mean we ignore the request from those with fewer descendant peers. Results in Figure 3.13 show that our algorithm does not just improve the average quality by sacrificing the minority's requests. On the contrary, it protects the minority's necessary needs when the network condition deteriorates. For application purposes, it also encourages the peers which cannot receive proper stream quality because of their positions in the overlay to relocate to a better position.

The maximum peer stream quality also decreases as the fluctuation becomes more drastic. It is because fluctuation also affects the stream quality at the upper levels of the multicast trees. The scheduling algorithms without the content dependency information, such as Numb, Rand, LIFO, NICE, are more likely to miss some important data and fail to recover some part of the stream.

3.4.3.3 Mismatched Weighting Scheme Performance

For some stream coding schemes, it is difficult to exactly specify the weight factor for the stream. For example, some coding schemes may have the Rate-Distortion value for

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.14: Stream Quality of Mismatched Weighting Scheme

each data packet in continuous range or presented in floating point value by complicated calculation. However, the transmission part only supports limited differentiated service level. Thus the accuracy of rate-distortion optimization will not be fully presented in the system. Thus, assigning weight to the stream data may not accurately reflect the importance of each data packet.

With reference to Mismatch-1 in Figure 3.7, it can be seen that 13 levels of precision has been reduced to 4 levels. For some general streaming applications, the system does not provide support for different codecs, or the stream does not explicitly provide the codec for weighting schemes. The system has to treat data equally such as Mismatch-2 in Figure 3.7. This kind of deviation will affect the performance of scheduling based only on weighting scheme. In terms of latency and data quality, only scheduling by weight is comparable to our algorithm, while the others are much worse than scheduling by weight. Moreover, the weight only scheduling is commonly used in video streaming and other prioritized streaming applications. Thus, the following results will only reflect the comparison between these two algorithms as it is not meaningful to compare the others any more, since they have worse performance.

Figure 3.14 shows the comparison of stream quality between weight scheduling and our scheduling algorithm under different weighting schemes in different overlay structures. A *fluctuation ratio* of 0.8 is used. Peers are classified into 4 groups based on their received

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

stream quality {below 70%, 70% ~ 80%, 80% ~ 90%, above 90%} compared to the original stream data. The total number of active peers is stable. Experiments are carried out based on the correct weighting scheme and the two mismatched weighting schemes shown in Figure 3.7 in order to study the variation in peer stream quality distribution under different scheduling algorithms and different degrees of weighting fidelity deviation. The results based on the mismatched weighting schemes are denoted as Mis-1 and Mis-2 in Figure 3.14.

With reference in Figure 3.14(A), Algo (with correct weighting scheme) has almost 68% of its peers enjoying recovered stream quality of more than 90% (black block), about 31% of its peers enjoying stream quality between 80% and 90% (dark gray block), with only 1% of the peers with stream quality between 70% to 80% (light gray block). Algo (Mis-1) has about 61% of peers enjoying above 90% stream quality, about 34% of peers with 80%-90% stream quality, almost 4% of peers with 70%-80% stream quality and about 1% of peers with below 70% stream quality. For Algo (Mis-2), the percentage of peers enjoying the respective stream quality is 12%, 27%, 33% and 28% respectively for above 90%, 80%-90%, 70%-80% and below 70% stream quality.

Results in Figure 3.14 show that the percentile of peers receiving high quality stream decreases when the weighting scheme could not truly reflect the stream data encoding scheme. It means that more and more peers have to bear with low quality stream as the weighting scheme fidelity decreases. However, as the weighting scheme fidelity decreases, our scheduling algorithm could still provide more peers with better stream quality compared to scheduling based only on weight. It is because our scheduling algorithm considers the overlay structure while sorting the data sending order. Data requested by more descendant peers is more likely to reach its destination because of higher priority through the overlay path even though the weight is deviated. Thus, it increases the possibility of successful recovery even when the weighting scheme fidelity decreases.

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

Figure 3.15: Data Recovery Ratio

Another metric worth mentioning is the data recovery ratio which shows the utilization of the bandwidth resource. The received packets may not be helpful to the stream presentation quality or cannot fully present their entire information unless other related parts are also received. The possibility of successful data recovery ratio not only depends on the precision of weighting scheme, but also the scheduling algorithm.

Figure 3.15 shows the average ratio of recovered stream packets to all received packets at peers under different weighting schemes and network conditions. For example, from Figure 3.15(A), Algo with the correct weighting scheme allows its peers almost full recovery of data from their received packets under the different fluctuation ratios. On the other hand, Algo (Mis-2) with mismatched weighting scheme only allows peers to recover 100% of the data under ideal condition. As the network fluctuation increases, Algo (Mis-2) peer recovery ratio drops to 0.99, 0.97, 0.91, 0.78 of all their received packets.

When the weighting scheme is correct ("Algo" and "Weig"), most of the packets can be successfully recovered under both scheduling algorithms in different overlay structures. If the fidelity of the weighting scheme degenerates to "Mis-1" and "Mis-2", the data recovery ratio decreases. The same explanation as that for Figure 3.14 applies. When fidelity of the weighting scheme decreases, peers are more likely to receive more unrecoverable packets because of the mismatched weighting. If network condition is worse, the effect of deviated weighting will be amplified and the packets which can fully present their

information is less. It is shown within each group of 5 different colored bars in Figure 3.15 that the recovery ratio drops when the *fluctuation ratio* increases. However, though the weight could not provide full fidelity information, by combining the partial overlay information and stream weight information, our scheduling algorithm helps to deliver the packets which are requested by more descendants with higher priority. It increases the number of packets received by peers which also increases the possibility that the received packets can be successfully recovered. Hence, though the recovery ratio decreases as the network condition deteriorates, the recovery ratio by our scheduling algorithm decreases slower than the scheduling only based on weight. The results of two pairs (Algo(Mis-1) vs. Weig(Mis-1) and Algo(Mis-2) vs. Weig(Mis-2)) show that our scheduling algorithm outperforms the weight only counterpart. Thus, adding overlay structure information into the scheduling scheme could not only highly improve the data recovery quality but also the data recovery quality is more resilient to bandwidth fluctuation.

3.5 Summary

In this chapter, we introduce a content and overlay-aware scheduling algorithm into the P2P system. It focuses on improving the data latency and received stream quality by scheduling the data sending order at each peer. The data which is requested by more number of peers and more important to the stream is placed at an earlier stage of the waiting queue. Accordingly, data packets experience differentiated treatment based on the overlay structure and content importance of the packets through the overlap paths. Both mathematical proof and simulation results show that the latency and stream quality could be improved by our scheduling algorithm. Comparing with other commonly used scheduling algorithms in different network conditions and overlay structures, our scheduling algorithm shows consistent performance and at a negligible algorithm overhead. Comparing with other commonly used scheduling algorithms in different network

CHAPTER 3. CONTENT AND OVERLAY-AWARE SCHEDULING

conditions and overlay structures, our scheduling algorithm shows consistent performance and at a negligible algorithm overhead with both simulated and real video data, namely, H.264 and MPEG4.

Chapter 4

Light Weight Prioritized Retransmission Mechanism

The work in Chapter 4 focuses on data exchange and protection. The pros and cons of push and pull mechanisms are discussed in Chapter 2. In order to incorporate the benefits from both the push and pull mechanisms, we propose the prioritized retransmission mechanism. It forwards the stream data as soon as the data arrives at peers like push mechanism. Meanwhile, it also publishes the buffered data availability to the neighbors as per the pull mechanism. When peers sense the mismatch between the received data and the available data from the parent peers, the prioritized retransmission task will be initiated. Since the stream data is mainly delivered by push mechanism, it maintains the efficiency of stream delivery. The prioritized retransmission allows the important data to be delivered in order. With a little overhead incurred in publishing data availability information, our bidirectional data exchange mechanism saves more bandwidth than adding redundancy into the stream data. A close examination shows that the retransmission mechanism better fits streaming applications than traditional push and pull mechanisms. The simulation results show the prioritized retransmission improves the stream quality compared to other commonly used methods.

4.1 Problem Formulation

In real networks, most stream data use unreliable data channel because of its light overhead and fast transmission. The drawback of unreliable data channel is that data may be lost in the network. Though some commercial applications use TCP to provide reliable data transmission and avoid data loss, the guaranteed transmission requires longer delay and buffer time which hinders users from enjoying timely streaming services. Thus, unreliable transmission is still preferable for delivering stream data.

However, stream data may lose more and more content when it traverses the tree overlay, level by level, without data protection. The negative impact of incomplete data will be exponentially accumulated through the data path. Such avalanche effect is detrimental to the P2P streaming application. Hence, the stream data passing through the lossy channel requires extra data protection mechanism. There are two ways to improve data quality through the lossy link: forward error correction and retransmission. By adding extra redundancy into the original stream data, forward error correction increases the probability of successful recovery of the original data. When small portion of stream data is lost, the missing part could be recovered from the redundant data. However, the redundancy added into the stream costs extra bandwidth resource. Moreover, the protection is usually effective only within the confine of a less dynamic environment. If the environment has a wide variation, the protection will degrade drastically. Retransmission request targets the missing data with a small overhead often limited to the index of the missing data. Though the overhead increases as the loss ratio of the peer connection increases, retransmission protection is still more effective in network environment with larger variation than forward error correction.

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

Figure 4.1: Example of Data Protection Comparison under Different Mechanisms

4.1.1 Merits of Retransmission - Quantitative Comparison

To provide an insight into the effectiveness of retransmission, we perform a quantitative comparison with Reed-Solomon (RS) like coding and unprotected data. To simplify the analysis, a simplified model of R-S like coding is used as an example of forward error correction. One redundant data block is added to every 3 data blocks. Every 3 out of 4 encoded data blocks received by the end peer can recover the full information of the original 3 data blocks. The metrics used for the comparison are data quality, bandwidth consumption and utilization under different lossy network environments represented by the loss ratios. Data quality measures the ratio of received data over the original data while bandwidth consumption measures the amount of bandwidth expended to receive the data. Bandwidth utilization is defined as the ratio of the data received over the bandwidth consumed in order to receive the data. The data rate through the peer connection is assumed to be 1 unit and the bandwidth between peers is taken as 1.5 units. For ease of comparison, the overhead of the retransmission requests is assumed to be 5% of the transmitted data which emulates a worst case for the retransmission mechanism. Based on the models set out above, we show an approximate trend of network resource usage and data quality under varying network conditions for the 3 different mechanisms.

The results in Figure 4.1 are calculated based on the different network loss ratios under the different data protection mechanisms. When the loss ratio is l , the ratio of

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

the received data in the unprotected scenario is $1 - l$. The used bandwidth for the unprotected scenario is 1 unit and the bandwidth utilization ratio is $(1 - l)/1$. Under the retransmission scenario, if the data is lost, retransmission requests will be issued and the traffic in the network is thus more than the original data. If the total traffic (including the original data, retransmission data and retransmission overhead) does not exceed the total bandwidth of 1.5 units, 100received yielding a data quality of 1. Given that the retransmission overhead soaks up 5the full data stream at a loss rate of l is $1 * (1 + 0.05)/(1 - l)$. If the maximum bandwidth of 1.5 units is used, we have $1 * (1 + 0.05)/(1 - l) = 1.5$ yielding a loss ratio of $l = 1 - 1/(1.5/(1 + 0.05)) = 0.3$. When the loss ratio exceeds this threshold, there will be a lot retransmission data but they will also experience loss and the received data is computed as $1.5/(1 + 0.05) * (1 - l)$ and the corresponding bandwidth utilization is $(1 - l)/(1 + 0.05)$. For R-S like coding scheme, every 3 of 4 data blocks received can recover the full stream. When the loss ratio is l , the probability of receiving 4 blocks is $(1 - l)^4$, and the probability of receiving any 3 of the 4 blocks is $4l(1 - l)^3$. The total probability of successfully recovering the full stream is the sum of two, which is $(1 + 3l) * (1 - l)^3$. The bandwidth used in the R-S like coding scheme is $4/3 = 1.33$. And the bandwidth utilization is $3/4 * (1 + 3l) * (1 - l)^3$.

The comparison is shown in Figure 4.1. We compare different data protection mechanisms: RS like coding (R-S), retransmission (Retrans) and stream without protection (Unprotected). The data quality versus data loss variation is shown in Figure 4.1(A). When the loss ratio is low (less than 0.2), the data quality of R-S coding is better than without any protection. However, when the loss ratio is high, the probability of receiving 3 out of 4 blocks drops drastically and little information can be extracted from the partially received data. Thus, in such condition, the data quality is even worse than without any protection where every received data block is valid. However, in the retransmission mechanism, the missing data will be requested and be sent again until it arrives. When

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

there are extra bandwidth resources, retransmission request will utilize the resources for retransmission. When the connection is fully utilized, the data quality will drop as loss ratio increases. Since it can utilize the extra bandwidth of the peer connection, the lost stream data has more possibility to be retransmitted. The data quality at the receiver is better than without any protection and RS coding.

Figure 4.1(B) shows the bandwidth consumed by the different protection mechanisms. Retransmission is flexible as its triggering will vary according to the loss ratio and will fully utilize the available bandwidth. The other two methods are fixed at a certain data rate. Though changing encoding or transcoding could adapt to different network conditions, such changes cost extra computational resources.

The utilization of network bandwidth is shown in Figure 4.1(C), which is defined as the received data quality over the bandwidth consumption. When there is no loss, the unprotected data will be transmitted at 1 unit of the channel bandwidth. RS like coding will consume 1.33 times of the actual data rate, since 1 redundant data block needs to be added to every 3 data blocks. The curve shows that retransmission incurs 5% overhead for retransmission request and utilizes bandwidth resources more efficiently than RS coding. Though it is not as efficient as the one without any protection, the better data quality achieved by retransmission can compensate for the lower bandwidth utilization efficiency.

The only drawback of retransmission is the extra delay generated while requesting for missing data. However, since current P2P streaming applications buffer the data until continuous playback is possible, this latency is not apparent to the users. Normally, the buffer time varies from tens of seconds to several minutes.

Comparing with data buffering time, the delay of retransmission request which is usually less than one second, is negligible. Its merits thus far outweigh this drawback.

4.2 Prioritized Retransmission Request

From the perspective of parent peer, the retransmission task shares the same connection as the original stream data distribution channel. The retransmission will cost extra bandwidth or contend for bandwidth with the existing stream data to fulfill the task. Thus, it is important to organize the retransmission data and existing queueing data fairly in the sending queue.

As discussed in chapter 3, the importance of the data is not equal according to the data content and overlay structure. The order of the sending sequence is based on the data importance. Since the retransmission data also have their importance based on their content and the descendent peers requesting for the data, the calculation of data importance in the previous chapter still stands for the retransmission data. Thus, the priority calculated for the retransmission data can also be used in scheduling the order with the existing queueing data.

4.2.1 Retransmission Scheme

Every parent peer connection keeps a DataMap as in Figure 4.2. The parent peer periodically publishes notifications of data index information it has sent out to the child peer (refer to the medium gray blocks as the sent but not acknowledged data in Figure 4.2). As child peer receives the data sent notification, it compares the notification with its received data from the parent. The mismatch will reflect the data lost in the delivery, and retransmission request is initiated. The missing data's priority is calculated at the child peer. The retransmission request is sent out with the priority. When parent peer receives the request, it will insert the requested data into the connection's sending queue based on the priority as per the existing queueing tasks. The child peers also periodically send the acknowledgement of received data to the parent peers. Parent peer marks the acknowledgement against its stream data (depicted as darker blocks for the acknowledged

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

data in Figure 4.2). This will prevent the parent peer from sending out the availability information of the received data to the child peer in the next round to reduce the overhead load. The expired data (depicted as black blocks for the expired data in Figure 4.2) are useless since they have either been played out before their expiration time or have not been received before their expiration time.

Figure 4.2: An example of DataMap for one connection

In proactive sending process, a source peer periodically publishes notifications of the sent data (the medium gray blocks in Figure 4.2) to the child peer and the child peer will send acknowledgement to the parent peer periodically. In passive sending process, when the connection is idle and there are data in the sending queue, a parent peer will pick the

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

first packet from the sending queue and send out the packet. The corresponding block of the packet in DataMap is thus changed from available but not sent status to sent but not acknowledged data status. (i.e. changing from the light gray blocks to medium gray blocks in Figure 4.2. The peers response to the arriving data is illustrated in Algorithm 2.

Algorithm 2 Peer Response to Received Data

INPUT: received data p_k from Connection e

if Data Type of p_k = Stream Data **then**

 Calculate the priority of p_k , Pr_{p_k}

 Pick the connections according to the substream forwarding rules,

 Add p_k into the connections' sending queue based on Pr_{p_k}

 Mark the block index of p_k in DataMap of these connections to Available but not sent status

 Add the index information of p_k into the acknowledgement for connection e

else if Data Type of p_k = Retransmission Request **then**

 Add the request data to connection e 's sending queue based on the priority

else if Data Type of p_k = Data Sent Notification **then**

 Compare the data sent from connection e with the received data from connection e , and find the missing data

 Calculate the missing data's priority

 Send retransmission request to connection e with the priority of missing data

else if Data Type of p_k = Child Peer Acknowledgement **then**

 Mark the block index of p_k in DataMap to acknowledged status

 Delete the availability information in the next round of sent data notification

end if

A point to note is in the event that the path between a parent and a child peer is bad, retransmission alone will certainly not improve the data quality at the child peer. It will then have to rely on the prevailing overlay refinement algorithm to allow the affected peers to switch to a better parent peer.

4.3 Performance Evaluation of Scheduling Algorithm with Retransmission

4.3.1 Simulation Setup

In our simulation, we generate a 1000-node network topology by BRITE[98] and randomly generate a 400-peer overlay on top of it. The data streaming rate is 2Mbps and is divided into 5 substreams. The 5 substreams are distributed through 5 independent multicast channels. Each peer connects to at least 5 peers and requests the stream data from the peer who can provide the shortest latency substream through each channel. Thus, some parts of the multicast structures may overlap. The bandwidth of each peer connection varies from 400Kbps to 2Mbps based on a uniform distribution. The maximum peer connections for each peer is 30.

In a dynamic environment, bandwidth cannot be kept stable at a certain level. In order to simulate the fluctuating environment, we introduce *fluctuation ratio* at each peer connection which changes the connection bandwidth. Hence, the available bandwidth between peers varies uniformly within a range around the estimated bandwidth, as in:

$$[1 - \text{fluctuation ratio}, 1 + \text{fluctuation ratio}] \times \text{estimated bandwidth}$$

A 0.2 fluctuation ratio implies that the connection bandwidth between two peers varies randomly from 0.8 times to 1.2 times of the estimated bandwidth based on uniform distribution. As the *fluctuation ratio* increases, the bandwidth will vary across a wider range. The larger the *fluctuation ratio*, the more drastic is the fluctuation in the P2P system. Peers monitor the bandwidth variation and update the sending data rate to fit the available bandwidth.

We compare our algorithm with 3 other commonly used scheduling schemes:

- **Algo:** Schedule as per our proposed scheduling algorithm in chapter 3.

- **NICE**[13]: Schedule the first received stream data to be sent out first.
- **LIFO**: Schedule the last received stream data to be sent out first.
- **Rand**: Randomly schedule the sending order of the received stream data.

The stream has expiration time of 10 seconds. Stream data which exceeds the expiration time will be discarded at the peers. Besides *fluctuation ratio*, we also introduce a *loss ratio* at each peer connection to simulate the lossy connection in a dynamic network environment. We have simulated the streaming for 10000 seconds with 25 random generated topologies. Since the result after 500 seconds shows the same trend, we only collect the simulation result for the duration of 500 seconds.

Since our proposed scheduling algorithm can readily adapt to temporary changes in the network, peers in our simulation remain stable in the overlay and do not change the substream request during the streaming services. Even with dynamic peer behavior like join and leave, our scheduling algorithm will still perform better. This is because it sends out the most wanted data first, and hence less such data will be lost when the peer leaves the streaming service. We also assume no data dependency between different parts of the stream.

4.3.2 Results

The request for retransmission will certainly compromise the latency for every scheduling algorithm as the missing data will have to be retransmitted and the retransmitted data will contend for bandwidth with the fresh data. Hence latency is not measured here directly as it will not be meaningful. Instead, the effect of latency is being indirectly reflected through the data quality. As the expiry time is set, so long as the data arrives before its playback time, it can be displayed and the data quality will correspondingly improve. Moreover, the earlier the stream data arrives at a peer, the earlier it can be

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

Figure 4.3: Data Quality in Different Fluctuating Environments

forwarded to the peer's descendants which therefore increases the possibility that the descendant peers can get the data on time thus improving the data quality received. The improvement of data quality with the incorporation of the retransmission mechanism in the presence of loss data indirectly reflects the improvement in latency under such dynamic scenario. Although retransmissions will take up bandwidth and may lead to more delay being incurred for both the retransmitted data and fresh data, the ability of our scheduling mechanism to schedule the retransmissions together with the fresh data has helped to alleviate the effects of the compromised latency, resulting in better data quality in a lossy environment compared to the other methods.

Figure 4.3 shows the data quality variation in different fluctuating environment. The *loss ratio* is 0.2 and the retransmission request interval is 1 second. The overhead is around 2% for every case. The data quality is defined as the average data rate received at each peer over the original data stream rate. As shown in the results, the data quality degrades when there are more drastic fluctuations in the network. Since the bandwidth is sometimes incompatible with the sending data rate, peers have to adapt the data rate to the available bandwidth. When the bandwidth is insufficient, peers will delay part of the stream data and send it out at later time. If the queueing data blocks cannot be

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

Figure 4.4: Data Quality in Different Lossy Environments

sent out before the expiration time, it will be discarded. Thus, as the *fluctuation ratio* increases, more data blocks will be delayed at each peer and will be waiting for sending opportunity. They are more likely to be discarded. By scheduling in our algorithm, the data distributed through the channel which is requested by more number of peers, will be sent out first. Thus, more number of peers will receive the data before the expiration time. The commonly used scheduling algorithms, like LIFO and Random, is 5% worse than our algorithm. NICE performs the worst because the data blocks peers sent are more likely to be close to the expiration time since its scheduling is based on sending packets with the earliest arrival times first. Hence more packets would already have expired during propagation.

In Figure 4.4, the data quality varies under different lossy environment. The *fluctuation ratio* is 0.4. When the *loss ratio* increases, more data blocks will be lost on their way and the data quality drops drastically. Figure 4.4(A) shows the data quality without retransmission request. Without retransmission, the effect of loss is accumulated level by level. As the tree structure grows, peers at the bottom of the tree may receive unacceptable quality stream. The data quality curves overlap in such conditions because the simulated loss is in addition to the fluctuating network environment which weeds out all the advantages of the different scheduling algorithms. This is actually one worst case

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

Figure 4.5: Average PSNR under Different Video Coding Schemes in Different Lossy Environments

scenario, meaning the P2P system is operating under extreme condition. Under more favorable conditions, our proposed scheduling algorithm performs very well as shown in Figure 3.11.

Figure 4.4(B) shows the data quality with retransmission request interval at 1 second. The overhead of retransmission is around 1.5% to 2.5% under different lossy conditions. With the help of retransmission request for missing data, data quality is improved by 30% to 40%. The retransmission increases the probability of successfully receiving stream data by utilizing the extra available bandwidth. However, in some of the peer connection where the bandwidth is insufficient, continued retransmission will further clog the link and contribute to network congestion. As shown in Figure 4.4(B), our algorithm puts the data block which is in the channel requested by most peers in the front of the queue and sends it earlier. Our retransmission scheduling provides 5% to 10% better data quality than the other algorithms.

To provide an insight into the physical data quality received in terms of Peak Signal to Noise Ratio (PSNR), we perform simulations using real video data coded in MPEG-4 and H.264 formats. The same videos coded in both formats are used. Figure 4.5 shows

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

Figure 4.6: Data Quality at Different Peer Levels

the average PSNR of the received real video stream at the peers in the overlay under MPEG-4 and H.264 video coding schemes under the same environment as Figure 4.4(B). It shows that the display quality under different video coding schemes is still improved by our scheduling algorithm with retransmission request in lossy environment. The most important packets are scheduled before the less important ones and the important frames are more likely to be reconstructed under our data delivery mechanism. Since 0.5dB difference in PSNR is considered as identifiable in video coding research, the improvement from our mechanism is significant compared to the other mechanisms in Figure 4.5. From the results in Figure 4.4(B) and Figure 4.5, the more stream data successfully recovered at each peer, the better PSNR the peer can get. The PSNR results are consistent with the data quality results yielded from the simulated data. The rest of the results will continue to be shown in terms of received data ratio as per many research work in P2P media streaming.

We also evaluate the peer data quality by their position in the multicast channels. The average data quality at each level of the multicast tree is shown in Figure 4.6. The *loss ratio* is 0.2, the *fluctuation ratio* is 0.4, and the retransmission request interval is 1 second. The figure shows that the nearer the peer is to the bottom of the multicast

CHAPTER 4. LIGHT WEIGHT PRIORITIZED RETRANSMISSION MECHANISM

Figure 4.7: Data Quality by Different Retransmission Request Frequency

trees, the more the data quality drops. The lossy effect accumulates at each level of the tree. Without retransmission protection ("No-Retrans" in the figure), the data quality declines exponentially. By requesting the missing data periodically, the data quality does not decline as fast as the one without protection. Our retransmission scheduling algorithm helps to provide more peers with the requested data first and performs better at compensating the cumulative effect of loss than other comparisons.

The data quality is also dependent on the retransmission request frequency. Figure 4.7 shows that when the retransmission request interval is bigger which means less frequent request for retransmission, the data quality declines under the *loss ratio* at 0.2 and *fluctuation ratio* at 0.4. On the other hand, frequent retransmission request helps to improve data quality. However, there are limitations to frequently using the retransmission request as they contend for bandwidth with the stream traffic or even exhaust network bandwidth. The overhead of retransmission request, in this case, varies within a reasonable range between 1.5% to 3.5% of stream traffic load depending on the request frequency. Since our algorithm requests the most wanted data by most peers in the first place, it provides data blocks which are useful to more peers. Thus, our scheduling algorithm performs better than the other comparisons.

In summary, the simulation results show that our scheduling algorithm with retransmission incorporated performs better than the others even under extreme operating conditions where there are losses over and above that cause by normal network fluctuations.

4.4 Summary

In this chapter, we propose a light weight bidirectional data exchange mechanism. It pushes the stream data across a data distribution tree while publishing the data availability to the neighbors periodically. The mismatch between received data of the child peer and the notifications published by the parent peer reflects the lost data. The retransmission request is issued for prioritized missing data. It provides the transmission efficiency from push and the responsive protection for data integrity from pull mechanisms with little incremental overhead. The qualitative analysis and simulation results show the improvement of our data exchange mechanism in a lossy dynamic network environment.

Chapter 5

Resilient Streaming Overlay Scheme

In this chapter, a new criterion to evaluate robustness is proposed. The robustness of a peer's stream is only as strong as the weakest link serving the peer among its connected peers. Based on this criterion, the quality of connection can be defined as how complementary the connection is to the other existing connections in order to improve the number of source providers supplying similar copies of the substream being supplied via the weakest link. Peers can thus find better connection combination and experience the best stream quality and backup robustness from the existing connections. The maximization of robustness among peers rather than self resource maximization prevents the contention for resources, and helps the peers to reasonably allocate and utilize the limited network resources.

5.1 Problem Formulation

The problem for tree based overlay is the fragility due to each peer having only one parent. To solve the problem, the whole stream data is partitioned to multiple substreams and distributed through independent tree structures as in SplitStream[9] and CoopNet[10]. The organization of the channels is classified as a multiple tree overlay as discussed in Chapter 2. It insulates peers from loss arising from connection failure. However, the independent multiple tree structure sacrifices delivery efficiency for robustness, since

Figure 5.1: Comparison of independent and shared tree structure

different substreams have to be delivered through different trees. In shared multiple tree structure, multiple substreams are allowed to be delivered through the same channel. However, the improvement in network utilization is at the expense of robustness, since the impact of a single connection failure cannot be insulated within one substream.

Figure 5.1 shows an example where the data stream is split into 2 substreams to be delivered via non-overlapped and overlapped trees. 4 peers are connected to peer A where each link can support 2 substreams, and peers C and F are connected to their neighbors with bandwidth of each link supporting only 1 substream. Peer A uses 4 connections to deliver substreams to its connected neighbors, in both the independent and shared trees. The delay of each link is set to 1. The latency experienced is 1.5 at peers B, D, E, G and 2 at peers C and F in the independent tree structure and 1 at peers B, D, E, G and 2 at peers C and F in the shared trees. From the example, the shared multiple trees could reduce the latency of the stream data by fully utilizing the available connection bandwidth in a heterogeneous environment. Moreover, a single connection between two peers handling multiple substreams in the shared tree saves more bandwidth

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

and reduces communication overheads compared to separate connections in independent trees. However, the connection failure between peers A and B affects 2 substreams in peer B and consequently affects 1 substream in peer C under shared tree overlay. The same failure in independent tree structure only affects 1 substream on both peers B and C. It shows that the independent tree overlay can be more robust than the shared tree overlay.

Though the pull mechanism, like CoolStream[11], can promptly respond to the connection failure, the three-step information exchange does not fit the tight temporal requirement in live streaming or near live streaming applications as discussed in Chapter 4.

Another solution for improving the robustness is to connect to more peers as backups for substreams as in ALMA[99]. The connections with other peers build a source pool for each peer, where the peer can have multiple source providers for each substream. It provides peers with alternatives to choose the best substream provider¹. These connections also provide the necessary backups when connection fails or when certain substream quality has degenerated.

From the above discussions, the shared multiple tree structure overlay with backup mechanisms is a promising solution for streaming where both stream quality and robustness in tight temporal requirement environment can be addressed.

Figure 5.2: Three Cases of Peer A's Source Pool

¹The peers should not choose the same substream from multiple source providers at the same time to avoid redundant traffic wasting network bandwidth

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

The problem is how to evaluate the quality of drawing a certain combination of substream from a source pool given that peers face multiple connections with different substream sources available under different download bandwidth constraints. For example, as in Figure 5.2, three different cases of peer A's source pool are shown. The whole stream is divided into 5 substreams with equal importance and same data rate, $\{Str_1, Str_2, Str_3, Str_4, Str_5\}$. There are 4 peers connected to peer A with the substream availability shown in the boxes of each connection. The blank box with no value indicates the substream is not available from that connection. The box with value Str_i indicates that substream i is available from that connection. The value set against each link to the peer A indicates the number of substreams peer A can download from that connection based on the download bandwidth. From the view of substream availability of the source pool, case (a) in Figure 5.2 is the best, since the number of alternatives for every substream is the most abundant among the three cases. From the perspective of download bandwidth, case (b) in Figure 5.2 is the best since the total download bandwidth from the 4 connections are the most among the three cases. Case (c) is the best in terms of the ability to supply Peer A with all the substreams comprising the entire stream data. The results of evaluating the source pool under different criteria are thus inconsistent with one another. Actually, the maximal number of substreams peer A can download from the source pool is 4, 3 and 5 for case (a), (b) and (c) respectively. One of the best solutions is indicated by the gray boxes in Figure 5.2. Although case (c) is not the best in terms of having more copies of available substreams or download bandwidth, it provides the best stream quality in the three cases in terms of completeness of the entire stream data.

Thus, it is important to design an effective criterion for evaluating the different combination of substream download from the source pool which can unify the aims of effective bandwidth allocation and stream data availability.

5.2 Source Pool Modeling and Stream Quality Maximization

Before proposing the solution for the stream quality measurement, the characteristics of peers, peer connection and peer's source pool are defined as follows:

Table 5.1: Notations

notation	Description
Str_i	The stream is divided into T substreams with equal importance. Each substream is labeled as Str_i , $i = 1 \dots T$
n_k	There are N peers in the overlay. Peers are denoted as n_k , $k = 1 \dots N$
$Conn_{k[j]}$	The j th connection to peer n_k with M being the upper bound of the total number of permitted connections for each peer, $j = 1 \dots M, k = 1 \dots N$
$n_{k[j]}$	Peers who are connected to peer n_k via connection $Conn_{k[j]}$, $j = 1 \dots M, k = 1 \dots N$.
$Bw_{k[j]}$	The download bandwidth of $Conn_{k[j]}$. (Here, we consider all substreams have equal stream rate. The bandwidth is illustrated as the number of substreams it can support via the connection), $j = 1 \dots M, k = 1 \dots N$
$A_{k[j]}^i$	A value of 1 indicates the availability of substream Str_i from the node $n_{k[j]}$ and 0 otherwise, $j = 1 \dots M, k = 1 \dots N, i = 1 \dots T$.
$E_{k[j]}^i$	A value of 1 means the substream Str_i is selected as a copy of Str_i from peer $n_{k[j]}$. 0 means no selection, $j = 1 \dots M, k = 1 \dots N, i = 1 \dots T$.

The set of the connection feature is defined as follows:

$$\begin{aligned}
 Bw_k &= \{Bw_{k[1]}, Bw_{k[2]}, \dots, Bw_{k[M]}\} \\
 A_{k[j]} &= \{A_{k[j]}^1, A_{k[j]}^2, \dots, A_{k[j]}^T\} \\
 A_k &= \{A_{k[1]}, A_{k[2]}, \dots, A_{k[M]}\} \\
 Conn_{k[j]} &= \langle Bw_{k[j]}, A_{k[j]} \rangle \\
 Conn_k &= \{Conn_{k[1]}, Conn_{k[2]}, \dots, Conn_{k[M]}\}
 \end{aligned}$$

The source pool can be modeled as a graph in Figure 5.3. S node is a virtual node representing all of peer n_k 's connections in the source pool. The edges between S and peer $n_{k[j]}$ indicates the j th connection for peer n_k , $Conn_{k[j]}$, which is between peer $n_{k[j]}$

Figure 5.3: The Max Flow Model for Peer n_k 's Source Pool

and n_k . The value above the edge indicates the upper bound of download bandwidth from $Conn_{k[j]}$. The edge between the node $n_{k[j]}$ and Str_i in Figure 5.3 indicates the availability of substream Str_i from peer $n_{k[j]}$. If $A_{k[j]}^i$ is 1, there is an edge between node $n_{k[j]}$ to Str_i , which means peer n_k can download the substream Str_i , from peer $n_{k[j]}$, which satisfies peer n_k 's stream temporal requirement. The edge between Str_i to n_k indicates peer n_k wants to download substream Str_i . The value 1 above the edge indicates a maximum of 1 substream is needed for Str_i .

The process to compute the best stream quality of a source pool can provide with the existing connections in terms of the number of download substreams can be modeled as a maximum flow problem. The best ability for peer n_k to download substream from the source pool can be transformed to the value of the maximum flow from node S to node n_k in Figure 5.3. The optimal download solution for peer n_k is indicated by the flow through the edge from node $n_{k[j]}$ and Str_i . If there is a flow through the edge, the value of $E_{k[j]}^i$ is 1, and substream Str_i is being downloaded from peer $n_{k[j]}$. If there is no flow through the edge, the value of $E_{k[j]}^i$ is 0, and the downloading of substream Str_i from peer $n_{k[j]}$ will not take place.

From Figure 5.3, the objective function and constraint function can be expressed as

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

follows:

$$C_{p_0} = \max \sum_{i=1}^T \sum_{j=1}^M E_{k[j]}^i \quad s.t. \quad (\text{Eq. 5.1})$$

$$E_{k[j]}^i \leq A_{k[j]}^i \quad k = 1 \cdots N, j = 1 \cdots M, i = 1 \cdots T \quad (\text{Eq. 5.2})$$

$$\sum_{i=1}^T E_{k[j]}^i \leq Bw_{k[j]} \quad k = 1 \cdots N, j = 1 \cdots M, \quad (\text{Eq. 5.3})$$

$$\sum_{j=1}^M E_{k[j]}^i \leq 1 \quad k = 1 \cdots N, i = 1 \cdots T \quad (\text{Eq. 5.4})$$

The goal of the maximization function is to obtain the maximum number of downloadable substreams, C_{p_0} , as in Eq. 5.1. Eq. 5.2 means that the selection of downloadable substream could only be chosen from the connections which provide the substream. Eq. 5.3 indicates that the number of substreams selected for downloading from each peer connection should not exceed the download bandwidth constraint. Eq. 5.4 indicates that at maximum, only one substream should be selected as the downloadable substream.

Thus the maximization of stream quality from the source pool is modeled into a classic Binary Integer Programming (BIP) problem. It can be solved in polynomial time. Some other classic algorithms like Dinitz' algorithm¹ [100] has a runtime complexity bounded by $O(MT(M+T)^2)$ and Ford-Fulkerson algorithm² [101] has runtime complexity bounded by $O(M * T * f)$, where the f is maximum flow of the graph.

The solution for the case in Figure 5.2(c) is shown as a graph in Figure 5.4. The bandwidth is expressed as x/y to indicate that x unit of bandwidth is used for downloading given a downloading capacity of y for the connection. The bold arrow indicates

¹The common expression for runtime complexity of Dinitz' algorithm is $O(P * V^2)$, where P is the number of edges in the graph and V is the number of vertices in the graph. Since the number of edges in our graph is $P = M * T + M + T$ and the number of vertices is $M + T + 2$, the runtime complexity of Dinitz' algorithm is equal to $O(MT(M + T)^2)$

²The common expression for runtime complexity of Ford-Fulkerson algorithm is $O(P * f)$, where P is the number of edges in the graph and f is the maximum flow in the graph. Since the number of edges in our graph is $P = M * T + M + T$, the runtime complexity of Ford-Fulkerson algorithm is equal to $O(M * T * f)$

Figure 5.4: An Example of Max Flow Solution in Graph

the respective substream being downloaded from the peer connection in the source pool. The maximum flow for the case in Figure 5.2(c) is $Cp_0 = 5$.

5.3 Robustness of Peer's Source Pool

Network dynamics and peer churns are unpredictable. Connection failure and changes in the availability of substreams affect the quality of the source pool at any time. The robustness of the source pool is thus also a measure of how well prepared is the source pool in adapting to unpredictable changes.

For illustration, Figure 5.2(c) is replicated in Figure 5.5(c1) while Figure 5.5(d1) is a new case for comparison. The two cases have the same bandwidth constraint in every connection. The total number of available substreams is also the same. Peer A can download all 5 substreams in both cases. However, when the 3rd connection fails in both cases as blocked out by the line in Figure 5.5, the best stream quality for the comparison case is 4 substreams as illustrated in Figure 5.5(d2) as opposed to 5 substreams in the original example as illustrated in Figure 5.5(c2). This example in Figure 5.5 shows that substream which has less alternative sources is more vulnerable to unpredictable changes in the source pool.

Figure 5.5: Impact of Connection Failure to the Source Pool

Thus, the criterion for evaluating the source pool quality should not only focus on the ability to allow for maximum number of downloadable substreams needed to form the complete stream, but should also aim at evaluating the weakest connection supplying the respective substreams while considering both stream data availability and bandwidth constraint of each connection in the source pool to handle the unpredictable changes.

5.3.1 Iterative Robustness Calculation

In order to measure the weakest link of the substreams, we use the same mechanism as finding the maximum number of downloadable substreams to determine the best quality the source pool can provide when all the downloadable substreams and the resources for downloading are no longer available. The bandwidth reserved for the downloadable substreams as well as the availability of these substreams are deleted from the graph to represent their absence from the source pool. The new round of substream selection thus starts in the residue graph. The value of the maximum flow in the residue graph indicates the number of substreams that can be downloaded in the absence of the current

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

substreams and resources. This iteration thus measures the availability of the 1st set of backup copies and the completeness of this 1st set of backup copies for the whole stream. Moreover, this set of backup copies does not contend for resources with the current downloadable substreams within the source pool, defined as Cp_1 .

Figure 5.6: Selection of the 1st Set of Backup Substream Copies

Figure 5.6 shows the solution for the maximum flow solution in the residue graph based on Figure 5.4. Besides the downloadable substream selection for Cp_0 , the goal of another round of substream backup copy selection for Cp_1 is to try to have the stream data as complete as possible, in order to measure the current source pool's ability to recover from the connection failure. The more number of substreams for Cp_1 , the more robust is the source pool to cope with connection failure. However, since Cp_0 determines the real stream quality and Cp_1 only kicks in when Cp_0 experiences failure, Cp_0 is always more important than Cp_1 .

The computation of Cp_0 allows all the different substreams in the source pool to be selected without bandwidth contention. Cp_1 is calculated from the residue graph after computation of Cp_0 ; the value of Cp_0 and Cp_1 should thus obey relation $Cp_0 \geq Cp_1$. The physical meaning of the relation $Cp_0 \geq Cp_1$ is that the 1st set of backup copies cannot have substreams which do not exist in the downloadable substreams in Cp_0 . It

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

thus makes the source pool reserve resources for downloading first, and use the remaining resources for backup purposes.

If more fine-scale assessment of the quality of source pool is needed, the calculation for Cp_2 continues as per that of Cp_1 using the residue graph of Figure 5.6. See Figure 5.7 for the complete process. The relation between Cp_0 and Cp_1 also applies to Cp_1 and Cp_2 where $Cp_1 \geq Cp_2$, because if a particular substream is not found in the 1st set of backup copy, it should not be found in the 2nd set of backup copy either. Since the possibility of 1 connection failure is larger than 2 connections failing at the same time, Cp_1 is thus more important than Cp_2 .

The calculation can continue for Cp_3 , Cp_4 , etc., until the download bandwidth is used up or there is no substream available from any connection that has download bandwidth. The measurement of quality of peer n_k 's source pool is defined as *Robustness* in the form of a tuple:

$$Robustness(n_k) = (Cp_0^{n_k}, Cp_1^{n_k}, Cp_2^{n_k}, Cp_3^{n_k}, \dots)$$

The relation between different source pools' *Robustness* is defined as:

$$Robustness(n_{k1}) = Robustness(n_{k2}), \quad \text{for } k1, k2 \in \{1, 2, \dots, N\}$$

$$\text{If } \forall x \in \mathbf{Z}, x \geq 0 : Cp_x^{n_{k1}} = Cp_x^{n_{k2}}$$

$$Robustness(n_{k1}) > Robustness(n_{k2}), \quad \text{for } k1, k2 \in \{1, 2, \dots, N\}$$

$$\text{If } \exists x \in \mathbf{Z}, x \geq 0 : Cp_x^{n_{k1}} > Cp_x^{n_{k2}}$$

$$\forall y \in \mathbf{Z}, 0 \leq y < x : Cp_y^{n_{k1}} \geq Cp_y^{n_{k2}}$$

$$\text{Otherwise, } Robustness(n_{k1}) < Robustness(n_{k2})$$

The source pool which has better quality in terms of *Robustness* is the one which has a more complete stream data at a lower index of backup set ¹ of substream copies. It implies that the source pool can provide more number of downloadable substreams as well as recover more quickly in the event of connection failure.

¹Here, lower index means the x of Cp_x is closer to 0

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Figure 5.7: Example of robustness computation

Continuing with the example of Cp_2 calculation in the residue graph of Figure 5.6, Figure 5.7 shows all the three steps in the calculation process. The table below the graph shows the bandwidth constraint and substream availability of each connection. The circled number indicates the substream selection from the respective connections during each iteration to compute the original downloadable substreams and the subsequent sets of backup copies.

With reference to Figure 5.7(a), a maximum of 5 substreams can be selected for the original set (i.e. 0th copy) of downloadable substreams. One possible solution is shown (refer to the circled values). The selected substreams are then deleted from the availability table as well as the respective bandwidth consumed by downloading them for the next iteration. The remaining bandwidth and substream availability after selection of the 0th copy is shown in Figure 5.7(b). The same process is repeated to obtain the maximum number of substreams for the 1st set of backup copy, which in this case is 2 as shown in Figure 5.7(b). After choosing one possible solution, the process iterates until there is no substream from the bandwidth sufficient connections that can be taken as a copy or there is no available bandwidth left. The value of *Robustness* for peer n_k 's source pool is thus (5, 2, 2).

5.3.2 Best *Robustness* Quality of Source Pool

Since the calculation for maximum Cp_x value for each x th set of substream backup copy may generate multiple optimal solutions, the result of Cp_{x+1} is highly dependent on the solution chosen from the previous iteration. To derive the maximum *Robustness* a source pool can provide, it is necessary to try every possible optimal solution in each of the iteration. The best *Robustness* the source pool can achieve reflects the actual ability of the source pool to recover from connection failure and peer churn.

The depth-first search algorithm is used for the iterative computation of the sets of substream copies as summarized in Algorithm 3.

Algorithm 3 Get the best robustness value for the n_k 's source pool

INPUT: Peer connection set $Conn_k$, and starts from the x th set of copies

OUTPUT: The best *Robustness* of n_k 's source pool, *Robustness*

```

function Robustness = Best_Robustness( $Conn_k$ ,  $x$ )
     $Max\_Robustness = (0, 0, 0, \dots)$ 
    if No ( $A_{k[j]}^i = 1$  AND  $Bw_{k[j]} > 0$ ) for  $Conn_{k[j]} \in Conn_k$ ,  $i \in 1, 2, \dots, T$  then
        return  $Max\_Robustness$ 
    end if
    Calculate for maximum value of  $Cp_x$  from  $Conn_k$ ,
    Put all possible optimal solutions into set  $S$ 
    for all  $s$  in  $S$  do
         $Tmp\_Conn = Conn_k$ 
        Apply solution  $s$  on  $Tmp\_Conn$  to build residue graph for the  $x + 1$  iteration
         $Tmp\_Robustness = \text{Best\_Robustness}(Tmp\_Conn, x + 1)$ 
        if  $Tmp\_Robustness > Max\_Robustness$  then
             $Max\_Robustness = Tmp\_Robustness$ 
        end if
    end for
    Replace the  $x$ th value of  $Max\_Robustness$  to  $Cp_x$ 
    return  $Max\_Robustness$ 
end function

```

The algorithm returns the best *Robustness* starting from the x th set of substream copy selection based on the given source pool information $Conn_k$. It initializes

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

$Max_Robustness$ with an empty $Robustness$ ¹ first. If no substream can be selected from the connections with available download bandwidth, an empty $Robustness$ will be returned. Otherwise, the maximum value of Cp_x is calculated using either BIP, Dinitz' algorithm or Ford-Fulkerson algorithm. The maximum number of different substreams without bandwidth contention is selected. All the possible solutions which can achieve Cp_x form an optimal solution set, S . A temporary variable, Tmp_Conn , is used to store the snapshot of the current source pool, $Conn_k$. Every optimal solution s is applied to the current source pool. The residue graph is built by removing the downloadable substream and its respective allocated bandwidth from Tmp_Conn based on solution s . The residue graph is then used to compute the best $Robustness$ starting from the next iteration for the next set of backup copy. The value is stored in $Tmp_Robustness$. If the value is larger than the best of all the previous tested optimal solutions, $Max_Robustness$ is replaced by $Tmp_Robustness$. Finally, the x th value of the $Max_Robustness$ is replaced by Cp_x . The value of $Max_Robustness$ is returned from **Best_Robustness**($Conn_k, x$) as the best robustness value starting from the x th backup copies of substreams until all bandwidth sustainable backup copies are selected from the given source pool, $Conn_k$. To get the best $Robustness$ of the source pool starting from the original downloadable substreams, the search begins by calling **Best_Robustness**($Conn_k, 0$) in Algorithm 3.

	Bw	A						Bw	A				
		Str ₁	Str ₂	Str ₃	Str ₄	Str ₅			Str ₁	Str ₂	Str ₃	Str ₄	Str ₅
Conn _k [1]	3	1	1	0	1	1	Conn _k [1]	3	1	1	0	1	1
Conn _k [2]	3	0	0	1	1	0	Conn _k [2]	3	0	0	1	1	0
Conn _k [3]	2	1	1	1	0	1	Conn _k [3]	2	1	1	1	0	1
Conn _k [4]	3	0	0	1	1	0	Conn _k [4]	3	0	0	1	1	0

(a)
(b)

Figure 5.8: The Difference in $Robustness$ between Computation of One Possible Solution versus the Best Solution after Investigating All Possible Solutions

¹The empty $Robustness$ is to assign 0 value to each Cp_x in the tuple

Continuing with the example in Figure 5.7, Figure 5.8 shows the difference in *Robustness* between a calculation based on one possible combination of downloadable substreams and the optimal one derived from the investigation of all possible solutions. The darkest shaded boxes are Set 0 (i.e. original downloadable substreams) of the substreams; medium-shaded boxes are Set 1 backup copy of substreams while the lightly-shaded boxes are the 2nd set of substream copies. After all the possible solutions are investigated for every set of backup copies, the best *Robustness* the source pool can achieve is $(5, 4, 0, 0, \dots)$, as shown in Figure 5.8(b). This robustness value is much larger than that of $(5, 2, 2, 0, 0, \dots)$ for Figure 5.8(a).

Since the calculation is an iterative process in solving maximum flow problem, the runtime complexity of the problem also increases. The iteration for the worst case is M rounds of calculation. Thus, the runtime complexity of the problem is increased to $O((MT(M+T)^2)^M)$ if Edmonds-Karp algorithm is employed, or $O((M * T * f)^M)$ if Ford-Fulkerson algorithm is employed. The problem cannot be solved in a polynomial time. However, since the number of connections, M , and the number of substreams, T , are usually limited and relatively small, the problem is usually manageable in the real world. We also propose a heuristic algorithm which can find near optimal solution in polynomial time in section 5.3.5.

5.3.3 Less Fine Scale *Robustness* Calculation

In order to reduce the runtime complexity, the fine scale calculation of *Robustness* can be loosen, when the number of connections, M , and the number of substreams, T , are large. Only several sets of backup copies are necessary, beyond which, they will assume to share the same number of backup copies. The expression of *Robustness* is simplified

to the following:

$$\begin{aligned} Robustness &= (Cp_0, Cp_1, \dots, Cp_{x-1}, Cp_{x+}) \\ Cp_{x+} &= \sum_{i=x}^{\infty} Cp_i \end{aligned}$$

Since Cp_{x+} counts all copies starting from Set x onwards, the backup copies of Set x is taken as the same copies for Set $x+1$ and so on. To calculate Cp_{x+} which is the total number of substreams available from all connections without bandwidth contention from Set x onwards, the condition in Eq. 5.4 can be released for the last round calculation. It is equivalent to changing the capacity of edges between Str_i and n_k to infinity in Figure 5.3. The maximum flow of the graph is thus the value of Cp_{x+} .

The problem can be solved in polynomial time, since there are $x+1$ iterations of the maximum flow calculation. By controlling the fine scale of *Robustness* calculation, the runtime complexity of Algorithm 3 can be managed at a solvable level when the number of connections, M , and the number of substreams, T , are large.

5.3.4 Selection of Substream with Bandwidth Conflicts

The residue graph after all iterations of maximum flow calculation leaves the edges of available substreams with no download bandwidth from the connection to support the download as well as connections with available download bandwidth but with no available substream for downloading. An example is shown in Figure 5.9 where (b) shows the optimal solutions for downloadable substreams (refer to the darkly shaded boxes) as well as Set 1 of the backup copies (refer to the lightly shaded boxes) for the example in Figure 5.7. After the two rounds of iterations, the residue graph is shown in Figure 5.9(a). The rest of the available substreams, like Str_1 from $Conn_{k[3]}$, are not selected because of the download bandwidth constraint.

Another example is shown in Figure 5.10. The darkly shaded boxes show the original downloadable streams of Set 0 while the lightly shaded boxes show the Set 1

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Figure 5.9: An Example of a Final Residue Graph

backup copies. The source pool condition is presented in Figure 5.10(b) with the same *Robustness* value as the case in Figure 5.9(b), which is $(5, 4, 0, \dots)$. The difference in the source pool between Figures 5.9 and 5.10 is that for the latter, in $Conn_{k[3]}$, there is no edge left in the final residue graph as illustrated in Figure 5.10(a)). Although the two source pools have the same *Robustness*, it does not mean that the remaining edges, like Str_1 from $Conn_{k[3]}$ in the residue graph of Figure 5.9(a) is useless. The difference becomes apparent when substream Str_1 becomes unavailable in both cases due to changes in the source pool, as in Figure 5.9(c) and Figure 5.10(c). In the case of Figure 5.9(c), Str_1 from $Conn_{k[3]}$ can be taken as the original downloadable copy in set 0 in the recalculation of *Robustness* upon changes in the source pool. In the case of Figure 5.10(c), Str_1 cannot be recovered as it is no longer available at $Conn_{k[3]}$.

Figure 5.10: Another Example of a Final Residue Graph

Thus, another tuple, $Robustness'$, for evaluating those unselected substreams due to bandwidth constraint in the source pool is necessary and is defined as follows:

$$Robustness' = (Cp'_0, Cp'_1, Cp'_2, \dots)$$

The value of Cp'_x in $Robustness'$ counts the number of different substreams in the x th set of possible backup copies which cannot be supported by the bandwidth in the

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Figure 5.11: Process for non-bandwidth sustainable substream selection

final residue graph upon computation of *Robustness*. Since the runtime complexity for counting the number of residue edges in each iteration is $O(M * T)$, the complexity of Algorithm 3 is not affected by the additional computation of *Robustness'*.

The counting of bandwidth unsustainable substream in *Robustness'* continues after the count in *Robustness* for each substream. Using the same example as Figure 5.9, the residual graph is shown in Figure 5.11(a). Figures 5.11(b) and (c) show the progressive selection of non-bandwidth sustainable substream backup copies while Figure 5.11 (d) summarizes the selected substreams in Set 0 and the subsequent sets of backup copies after *Robustness* and *Robustness'* computation. Bandwidth unsustainable substreams identified in *Robustness'* computation are indicated as crossed boxes. As per the terminology used earlier, the darkness of the shaded boxes indicates the set index of the backup copies which means that the darkest boxes refer to the original downloadable substreams (Set 0); the medium shaded boxes refer to Set 1 backup copies while the lightly shaded boxes refer to the Set 2 backup copies.

Starting from Str_1 , since there is already 1 bandwidth sustainable copy for Str_1 identified as Set 0 copy, the count of non-bandwidth sustainable substream starts from

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Figure 5.12: Optimal Solution for Source Pool with Both Bandwidth Supported and Non-Bandwidth Supported Copies

Set 1 backup copy. Str_1 from $Conn_{k[3]}$ is counted as a non-bandwidth sustainable copy in Set 1 backup copy of $Robustness'$ with the corresponding Cp'_1 for Set 1 backup copy being incremented by 1. See crossed medium shaded box in Figure 5.11(b). If there are other connections which can provide Str_1 as non-bandwidth sustainable backup copies, each of them will be considered as a backup copy in Sets 2, 3, $\dots$ accordingly in $Robustness'$. Since there is no other non-bandwidth sustainable copies for Str_1 , the algorithm proceeds to examine Str_2 . As there is no non-bandwidth sustainable copy for Str_2 , no backup copy for Str_2 will be counted in $Robustness'$. From Figure 5.11 (d), Str_3 already has 2 bandwidth sustainable copies in Sets 0 and 1. Hence the non-bandwidth sustainable Str_3 from $Conn_{k[3]}$ is identified as the backup copy of Set 2 in $Robustness'$. Cp'_2 will thus be incremented by 1 and this is denoted as the crossed medium shaded box in Figures 5.11(b) and (d). Similarly, Str_4 from $Conn_{k[1]}$ is taken as Set 2 in $Robustness'$ as reflected in Figures 5.11(c) and (d). Thus, $Robustness'$ of the example in Figure 5.11 is $(0, 1, 2, \dots)$.

Crossed boxes in Figures 5.12(a) and (b) show the selection of bandwidth unsustainable substreams under the source pool condition of Figure 5.9 and Figure 5.10, respectively. From the above examples, the combined usage of the two measurements, $Robustness$ and $Robustness'$, is helpful to recognize the actual robustness quality and downloading stream quality of the source pool. The $Robustness_Pair$ combines the two tuples as follows:

$$Robustness_Pair = ([Cp_0, Cp'_0], [Cp_1, Cp'_1], [Cp_2, Cp'_2], \dots)$$

The comparison of elements in the two *Robustness_Pair* tuples for source pools of peer n_{k1} and n_{k2} is defined as follows:

$$[Cp_x^{n_{k1}}, Cp_x'^{n_{k1}}] = [Cp_x^{n_{k2}}, Cp_x'^{n_{k2}}]$$

$$\text{If } Cp_x^{n_{k1}} = Cp_x^{n_{k2}} \text{ and } Cp_x'^{n_{k1}} = Cp_x'^{n_{k2}}$$

$$[Cp_x^{n_{k1}}, Cp_x'^{n_{k1}}] > [Cp_x^{n_{k2}}, Cp_x'^{n_{k2}}]$$

$$\text{If } Cp_x^{n_{k1}} > Cp_x^{n_{k2}} \text{ or}$$

$$Cp_x^{n_{k1}} = Cp_x^{n_{k2}} \text{ and } Cp_x'^{n_{k1}} > Cp_x'^{n_{k2}}$$

$$\text{Otherwise, } [Cp_x^{n_{k1}}, Cp_x'^{n_{k1}}] < [Cp_x^{n_{k2}}, Cp_x'^{n_{k2}}]$$

The comparison relation for two *Robustness_Pair* tuple is the same as tuple comparison of *Robustness*. The *Robustness_Pair* of the case in Figure 5.9 is $([5, 0], [4, 1], [0, 2], [0, 0], \dots)$ which is larger than the *Robustness_Pair* of the case in Figure 5.10, $([5, 0], [4, 1], [0, 1], [0, 0], \dots)$.

5.3.5 Heuristic Algorithm to Improve Calculation Efficiency

Since the optimal *Robustness_Pair* cannot be calculated within polynomial time by Algorithm 3, a heuristic algorithm is proposed which can achieve near optimal outcome as the multiple-step maximum flow computation for *Robustness_Pair* of the source pool within polynomial time. The maximum fine scale for the iteration is controlled by setting *Max_Backup_Level* in the algorithm. The process is stated as follows:

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Algorithm 4 Obtain the robustness value for the existing source pool

INPUT: Peer connection set $Conn_k$ **OUTPUT:** Maximal value for the source pool, $Robustness_Pair$

```

1: function  $Robustness\_Pair = \text{Get\_Robustness}(Conn_k)$ 
2:   SubStream\_Counter[j] = 0 for  $j \in 1, 2, \dots, T$ 
3:   for all  $Conn_{k[i]}$  in  $Conn_k$  do
4:     Counter[i] = the number of available substreams in  $Conn_{k[i]}$ 
5:     if Counter[i] = 0 then
6:       Conn\_Quality[i] = 0
7:     else
8:       Conn\_Quality[i] =  $Bw_{k[i]} / \text{Counter}[i]$ 
9:     end if
10:    if Conn\_Quality[i]  $\geq 1$  then
11:      for all  $A_{k[i]}^j = 1$  in  $Conn_{k[i]}$  do
12:        SubStream\_Counter[j] = SubStream\_Counter[j] + 1
13:         $Bw_{k[i]} = Bw_{k[i]} - 1$ 
14:         $A_{k[i]}^j = 0$ 
15:      end for
16:      Conn\_Quality[i] = 0
17:    end if
18:  end for
19:  repeat
20:    changed\_marker = false
21:    min\_counter = Minimum(SubStream\_Counter)
22:    Search for  $Conn_{k[i]}$  such that for SubStream\_Counter[j]=min\_counter,  $A_{k[i]}^j=1$ 
    with maximal Conn\_Quality[i] and  $Bw_{k[i]} > 0$ 
23:    if  $i$  and  $j$  exist then
24:      SubStream\_Counter[j] = SubStream\_Counter[j] + 1
25:       $Bw_{k[i]} = Bw_{k[i]} - 1$ 
26:       $A_{k[i]}^j = 0$ 
27:      changed\_marker = true
28:    end if
29:    for all  $Conn_{k[i]}$  in  $Conn_k$  do
30:      Counter[i] = the number of available substreams in  $Conn_{k[i]}$ 
31:      if Counter[i] = 0 then
32:        Conn\_Quality[i] = 0
33:      else
34:        Conn\_Quality[i] =  $Bw_{k[i]} / \text{Counter}[i]$ 
35:      end if
36:    end for
37:  until changed\_marker = false
38:
  
```

▷ For space reason we need to break here!

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

```

39:  Initilize  $Cp_i = 0$  in Robustness, where  $i = 0$  to Max_Backup_Level
40:  for  $i = 0$  to Max_Backup_Level -1 do
41:    for all SubStream  $j$  do
42:      if SubStream_Counter[ $j$ ] >  $i$  then
43:         $Cp_i = Cp_i + 1$ 
44:      end if
45:    end for
46:  end for
47:  for all SubStream  $j$  do
48:    if SubStream_Counter[ $j$ ] > Max_Backup_Level then
49:       $Cp_{Max\_Backup\_Level} = Cp_{Max\_Backup\_Level} + \text{SubStream\_Counter}[j] -$ 
Max_Backup_Level
50:    end if
51:  end for
52:  Initilize  $Cp'_i = 0$  in Robustness', where  $i = 0$  to Max_Backup_Level
53:  for all  $Conn_i$  and Substream  $j$  do
54:    if  $A_{k[i]}^j = 1$  then
55:      SubStream_Counter[ $j$ ] = SubStream_Counter[ $j$ ] +1
56:       $A_{k[i]}^j = 0$ 
57:      if SubStream_Counter[ $j$ ] ≤ Max_Backup_Level then
58:         $Cp'_{SubStream\_Counter[j]-1} = Cp'_{SubStream\_Counter[j]-1} +1$ 
59:      else
60:         $Cp'_{Max\_Backup\_Level} = Cp'_{Max\_Backup\_Level} + 1$ 
61:      end if
62:    end if
63:  end for
64:  Combine Robustness and Robustness' to make Robustness_Pair
65:  return Robustnes_Pair
66: end function

```

In our proposed heuristic algorithm, the counters for each substream copy are first set to 0. The ability to provide the substreams from each connection is recorded in Conn_Quality. A high value of Conn_Quality indicates that the connection has abundant bandwidth relative to the number of available substreams. If Conn_Quality is greater or equal to 1, the selection of substream from this connection will not affect the selection of other substreams from the same connection, or other connections. Thus, the substream selection starts from those connections whose Conn_Quality is greater or equal

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

to 1. The selected substream is considered as bandwidth supported substream and the bandwidth allocated for downloading the substream and the availability of the substream are deducted from that connection, since the substream from one connection can only be selected once. The `changed_marker` is set in order to detect whether a new substream has been selected. If no new substream is selected, the loop will stop. The **Minimum** function returns the minimum value of the `SubStream_Counter` array, an array which captures the various available substreams and bandwidth. The selection process of the substreams comprising the different sets of backup copies starts from searching among the minimum `SubStream_Counter` substreams and selecting the substream from the connection which can provide maximum `Conn_Quality` and where the bandwidth is sufficient. Likewise, the bandwidth and availability of the substreams are deducted from the connection accordingly. The `changed_marker` is set to **true** when a new substream is selected. `Conn_Quality` is updated. When no new substream is taken as copies, the *Robustness* of the connection combination is calculated based on how many bandwidth sustainable copies can be selected for each substream. The *Max_Backup_Level* is the maximum scale for backup copy computation and determines the maximum number of sets of backup copies to compute. After the bandwidth sustainable copies are selected, the substream which does not have bandwidth allocated is determined for *Robustness'*. The combination of *Robustness* and *Robustness'* constitutes the *Robustness_Pair* of the source pool, which is returned by the algorithm.

In the example of Figure 5.12(a), Counter for each connection is 4, 2, 4, 2 respectively. The corresponding `Conn_Quality` (i.e. bandwidth divided by the number of available substreams where the available bandwidth is 3, 3, 2, 3) is 0.75, 1.5, 0.5, 1.5 for the 4 connections. Since $Conn_{k[2]}$ and $Conn_{k[4]}$ have `Conn_Quality` larger than 1, the two copies of Str_3 and Str_4 from the two connections are counted in `SubStream_Counter[3]` and `SubStream_Counter[4]` as 2 in each counter and the bandwidth and substream availability

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

are removed (see lines 10-17) accordingly in both connections and Conn.Quality array becomes 0.75, 0, 0.5, 0 (see line 16). In the loop (see lines 19-37), bandwidth sustainable substreams will be selected one by one. The min_counter returns 0, since the counted copies in SubStream.Counter array is 0, 0, 2, 2, 0. From the example, it can be clearly seen that there are substreams and bandwidth available for at least one of the minimum SubStream.Counter elements. For example Str_1 can be provided by $Conn_{k[1]}$ with available bandwidth. At the beginning of each round of loop, it searches for connection j that has maximum value in the Conn.Quality array (0.75, 0, 0.5, 0) and can provide the bandwidth sustainable substream i for which SubStream.Counter[i] equals the min_counter. Substream Str_1 from $Conn_{k[1]}$ is selected since SubStream.Counter[1]=0 and Conn.Quality[1] is maximum. Thus, SubStream.Counter[1] is incremented by 1 and the availability of Str_1 from $Conn_{k[1]}$ and the bandwidth are removed and Conn.Quality array is updated to (0.67, 0, 0.5, 0) since $Conn_{k[1]}$ now has only 2 units of bandwidth and 3 available substreams. The changed_mark is set to true. The new round of substream selection picks Str_2 from $Conn_{k[1]}$ and the Conn.Quality is changed to (0.5, 0, 0.5, 0) where $Conn_{k[1]}$ now has only 1 unit of bandwidth and 2 available substreams. The loop continues and the substream selections are Str_5 from $Conn_{k[1]}$, Str_2 from $Conn_{k[3]}$, Str_5 from $Conn_{k[3]}$. Thereafter, there is no bandwidth sustainable substream copy to be selected. The SubStream.Counter array stands at (1, 2, 2, 2, 2), which is the selected copies for each substream. Cp_i is calculated up to $Cp_{Max_Backup_Level}$. For Max_Backup_Level equals 3, the *Robustness* is (5, 4, 0, 0). Since there is no selected substream backup copies which is more than Set Max_Backup_Level , the rest of algorithm will not be executed (lines 47-51).

The next part is the *Robustness'* calculation where the non-bandwidth sustainable substream backup copies are selected. Str_4 from $Conn_{k[1]}$ is selected, and SubStream.Counter[4] is incremented to 3. Cp'_2 is incremented to 1. Str_3 from $Conn_{k[3]}$

is selected and `SubStream_Counter[3]` is also incremented to 3 with Cp'_2 incremented to 2. Str_1 from $Conn_k[3]$ is selected and `SubStream_Counter[1]` increases to 2 with Cp'_1 increasing to 1. $Robustness'$ is thus (0, 1, 2, 0). The combination of $Robustness$ and $Robustness'$ results in $Robustness_Pair$, ([5,0], [4,1], [0,2], [0,0]), which is same as the previous calculation.

The runtime complexity for the heuristic algorithm is $O((M*T)^2)$, which is acceptable when the search space is large. We randomly generated 1 million cases under the scenario of 5 substreams and 10 peer connections source pool with $Max_Backup_Level = 3$. There are only 167 cases (about 0.017%) where there is a difference in the computation of $Robustness_Pair$ of the source pool between the multiple-step BIP and our heuristic algorithm. Thus, the heuristic algorithm is a feasible solution to provide the robustness measurement of the connection combination source pool when the search space is large.

5.3.6 Meaning of Backup Substream Copy and Its Application

The calculation of the backup substream copies provides an insight into how to receive a high stream quality with robust recovery ability from the existing source pool. It gives the maximum bandwidth sustainable backup copies each substream can get based on the current download bandwidth and substream availability of the connections. Since the bandwidths are reserved for the substreams, the downloading request of any such substream will not have any bandwidth contention with other substreams using the same connection. Thus, after the calculation, the concept of the original downloadable substreams and the order of the sets of backup substream copies is no longer necessary. The ordering is to facilitate the marking of all possible sources of each substream from each connection without bandwidth conflict during the computation.

Thus, the downloadable substreams can be selected from all the bandwidth sustainable copies of that substream. For the example in Figure 5.12, the downloading request

for a particular substream can be made to any of its shaded boxes except those with the cross.

In order to improve the data delivery efficiency and latency, the source which can provide short latency is preferred. In the application, when a new peer joins the overlay, it will first contact the server for several peer candidates to connect to. Then the new peer attempts to contact the candidate peers. The new join peer will find the optimal substream selection combination which can provide the maximum *Robustness_Pair* among all the contactable candidates. The new peer can download each substream from the bandwidth sustainable copies and will issue download request to the connection which can provide the substream at the shortest latency.

5.4 Best Source Pool Construction Scheme under Constrained Environment

However, the redundant connections for backup do not come at no cost. It takes time for peers to search for a proper backup connection. Resources have to be allocated to maintain the backup connections between two peers. Control overhead has to be generated periodically to monitor the availability of the backup connections. The resource used for improving one peer's robustness may hinder other peers from requesting stream data through that path, especially when data distribution is a more effective and meaningful use of limited resources rather than providing standby backup connections. Thus, it is important to prudently balance the resources used for streaming and for backup.

In order to keep the resource allocation balanced throughout the overlay, peers should control their desire to exploit and exhaust the network resources for the overall overlay efficiency. A cap on the number of connections for each peer is necessary and realistic to maintain the peers' stream quality and robustness at a proper level. Since the number of peer connections is limited, a peer should check out any new connection against existing

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

connections especially when the source pool hits the connection number limit. The best combination of connections should be the one which achieves the best *Robustness_Pair* in the source pool. The least useful connection which is not in the combination will be replaced by the new connection. The deletion and addition of connections to the source pool require the definition of subtraction and addition relations in the *Robustness_Pair* to maintain its consistency. The definitions are as follows:

$$\begin{aligned}
 & \text{Robustness_Pair}(A) + \text{Robustness_Pair}(B) \\
 = & ([Cp_0^A + Cp_0^B, Cp_0'^A + Cp_0'^B], [Cp_1^A + Cp_1^B, Cp_1'^A + Cp_1'^B], \dots) \\
 & \text{Robustness_Pair}(A) - \text{Robustness_Pair}(B) \\
 = & ([Cp_0^A - Cp_0^B, Cp_0'^A - Cp_0'^B], [Cp_1^A - Cp_1^B, Cp_1'^A - Cp_1'^B], \dots)
 \end{aligned}$$

The stability of the overlay is also important to the stream quality, since any disconnection during data streaming will lead to all descendent peers having to change their substream source provider and cause overlay oscillation. To keep the overlay structure stable, it is necessary to protect the existing connections which are downloading substreams in the computation for the optimal connection combination. In our overlay construction scheme, we give the connection which is downloading stream data an extra priority, by reducing the *Robustness_Pair* by 1 in Cp_0 if the new connection wants to preempt the existing connection which is currently downloading while finding the maximum *Robustness_Pair* connection combination. The procedure of selecting the weakest connection to be replaced by a new connection is one which leads to achieving the maximum increase in *Robustness_Pair*, denoted as *Rob_Change* when the new link replaces an existing link and is shown as follows:

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Algorithm 5 Calculate the maximum incremental change when new connection replaces one connection in the source pool.

INPUT: The connection set $Conn_k$ in peer n_k and the new connection $Conn_{k[new]}$

OUTPUT: The connection $Conn_{k[repl]}$ which can be replaced by the $Conn_{k[new]}$ with maximum Rob_Change

```

1: function  $[Conn_{k[repl]}, Rob\_Change] = \text{Max\_Replace\_Change}(Conn_k, Conn_{k[new]})$ 
2:    $Rob\_Change = ([-\infty, -\infty], [-\infty, -\infty], \dots)$ 
3:    $Curr\_Rob\_Pair_{n_k} = \text{Get\_Robustness}(Conn_k)$ 
4:   for all  $Conn_{k[i]}$  in  $Conn_k$  do
5:      $n_p =$  the other end of connection  $Conn_{k[i]}$ 
6:      $Conn_{p[j]} =$  the connection  $Conn_{k[i]}$  from the view of peer  $n_p$ 
7:      $Curr\_Rob\_Pair_{n_p} = \text{Get\_Robustness}(Conn_p)$ 
8:      $Repl\_Rob\_Pair_{n_k} = \text{Get\_Robustness}(Conn_k - Conn_{k[i]} + Conn_{k[repl]})$ 
9:      $Repl\_Rob\_Pair_{n_p} = \text{Get\_Robustness}(Conn_p - Conn_{p[j]})$ 
10:    if any  $Str_x$  is downloading through  $Conn_{k[i]}$  or  $Conn_{p[j]}$  then
11:       $Adjustment = [-1, 0], [0, 0], \dots$ 
12:    else
13:       $Adjustment = [0, 0], [0, 0], \dots$ 
14:    end if
15:    if  $Rob\_Change < Repl\_Rob\_Pair_{n_k} + Repl\_Rob\_Pair_{n_p} - Curr\_Rob\_Pair_{n_k} -$ 
       $Curr\_Rob\_Pair_{n_p} + Adjustment$  then
16:       $Conn_{k[repl]} = Conn_{k[i]}$ 
17:       $Rob\_Change = Repl\_Rob_{n_k} + Repl\_Rob_{n_p} - Curr\_Rob_{n_k} - Curr\_Rob_{n_p} +$ 
       $Adjustment$ 
18:    end if
19:  end for
20:  return  $[Conn_{k[repl]}, Rob\_Change]$ 
21: end function

```

The process of Algorithm 5 is started with the initialization of the Rob_Change in the format of $Robustness_Pair$ with the value of each element set to $-\infty$. The current $Robustness_Pair$ of peer n_k is calculated and stored in $Curr_Rob_Pair_{n_k}$. Every connection in the source pool of n_k will be replaced by the new connection $Conn_{k[new]}$ and be tested. n_p is the peer at the other end of connection $Conn_{k[i]}$. The connection $Conn_{p[j]}$ from the view of peer n_p is the same as the connection of $Conn_{k[i]}$ from the view of n_k . The current $Robustness_Pair$ of peer n_p 's source pool is stored in $Curr_Rob_Pair_{n_p}$. The $Robustness_Pair$ of n_k after $Conn_{k[new]}$ replaces $Conn_{k[i]}$ is stored in $Repl_Rob_Pair_{n_k}$.

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

The *Robustness_Pair* of n_p after $Conn_p[j]$ is replaced and disconnected is stored in $Repl_Rob_Pair_{n_p}$. If there is stream data exchange through the replaced connection, an extra priority, *Adjustment* in the format of *Robustness_Pair*, is used. If there is no stream data exchange, the *Adjustment* is set to empty. The total changes of the *Robustness_Pair* at both sides of the connection after replacement is calculated as in line 15. If it is larger than all previous tests, the current testing connection will be recorded in $Conn_{k[repl]}$, and the changes is recorded in *Rob_Change*. Finally, the weakest connection in source pool $Conn_{k[repl]}$ is identified as the possible candidate to make way for new connection which can improve the source pool robustness. The maximum incremental change *Rob_Change* is returned as a pair.

If the new connection wants to replace the existing connections in the source pool, it has to provide the affected peers at both sides of the replaced connection with higher *Robustness_Pair* values in total, especially when the replaced connection is streaming data.

During the stay in the overlay, the connected peers periodically exchange information about their connected neighbors in a random gossip manner. When new peers are contactable, the peer will check whether the new connection improves the total *Robustness_Pair*. Meanwhile, if the number of the connection exceeds the upper bound for each peer, the replacement of the existing connection may also affect the *Robustness_Pair* of other peers. Thus, the improvement of the *Robustness_Pair* should take into consideration all peers affected by the new connection. The decision procedure for the new connection is summarized in Algorithm 6.

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Algorithm 6 Decision on Whether a New Connection should be Admitted into the Source Pool

INPUT: The connection set $Conn_{n_a}$ and $Conn_{n_b}$ in peer n_a and n_b , and new connection $Conn_{n_a[new]}$, $Conn_{n_b[new]}$ from the views of peer n_a and n_b .

```

1: procedure Connection_Replacement( $Conn_{n_a}$ ,  $Conn_{n_b}$ ,  $Conn_{n_a[new]}$ ,  $Conn_{n_b[new]}$ )
2:   if connection in  $Conn_{n_a}$  is full then
3:     [ $Conn_{n_a[repl]}$ ,  $Repl\_Rob\_Pair_{n_a}$ ] = Max_Replace_Change( $Conn_{n_a}$ ,  $Conn_{n_a[new]}$ )
4:   else
5:      $Conn_{n_a[repl]}$  = empty slot in  $Conn_{n_a}$ 
6:      $Repl\_Rob\_Pair_{n_a}$  =  $\text{Get\_Robustness}(Conn_{n_a} + Conn_{n_a[new]}) -$ 
        $\text{Get\_Robustness}(Conn_{n_a})$ 
7:   end if
8:   if connection in  $Conn_{n_b}$  is full then
9:     [ $Conn_{n_b[repl]}$ ,  $Repl\_Rob\_Pair_{n_b}$ ] = Max_Replace_Change( $Conn_{n_b}$ ,  $Conn_{n_b[new]}$ )
10:  else
11:     $Conn_{n_b[repl]}$  = empty slot in  $Conn_{n_b}$ 
12:     $Repl\_Rob\_Pair_{n_b}$  =  $\text{Get\_Robustness}(Conn_{n_b} + Conn_{n_b[new]}) -$ 
       $\text{Get\_Robustness}(Conn_{n_b})$ 
13:  end if
14:  if  $Repl\_Rob\_Pair_{n_a} + Repl\_Rob\_Pair_{n_b} \geq ([0,0], [0,0], \dots)$  then
15:    Replace the connection  $Conn_{n_a[repl]}$  and  $Conn_{n_b[repl]}$  by  $Conn_{n_a[new]}$ ,  $Conn_{n_b[new]}$ 
16:  end if
17: end procedure

```

Algorithm 6 finds the maximum *Robustness_Pair* combination of the source pool considering all existing connections and the new connection. The calculation is conducted at both sides of the new connection, peer n_a and n_b . If the connection at any peer is not full, find an empty connection slot, $Conn_{n_a[repl]}$ and $Conn_{n_b[repl]}$ at each peer for the new connection and set replacement changes value at each peer, *Repl_Rob_Pair*, to be the increment of *Robustness_Pair* by adding the new connection into the source pool. Otherwise, the *Repl_Rob_Pair* value is calculated from the function *Max_Replace_Change* with the maximal incremental change of the two affected peers stored in the same format as *Robustness_Pair*. If the total incremental change of the two peers is non-negative as in line 14, the permanent connection between n_a and n_b is built. It will be added to the source pool of both peers. If the peer's connection number is not at the maximum,

the new connection will be added directly. If the peer's connection number is at the maximum, replacement is carried out at the peer by replacing the weakest connection with the new connection.

When a new peer first joins the network, it contacts the server for candidate peers to connect to. After receiving the information about the other peers, it tries to contact these peers to seek connection and requests for substreams based on the neighbors availability and its expiry time. When a peer receives a request for new connection, Algorithm 6 is called to decide whether the new connection is accepted or rejected, and if any existing connection is to be replaced. If a change results in the source pool of the peer, Algorithm 4 will be called to calculate the *Robustness_Pair* and substream will be requested from the bandwidth supported substream copies through the connection which can provide the shortest data latency.

5.5 Deployment Issue

The condition under which a new connection replaces an existing connection is based on how much better the new connection can be complementary to the existing source pool than the existing connection. Since the more number of backup copies provided in source pool will have larger *Robustness_Pair* value, the connections which can provide more number of backup copies, will preempt the one which provides the least in the source pool. Since the calculation of the optimal combination gives preference to the low order set of backup copies¹, the connection which can increase the value at the low order elements in *Robustness_Pair* will preempt the connection which provides the substream copies in the higher order sets.

Another implicit criterion of connection preemption from the uploader point of view is that, since it is harder for the short-latency tolerant peers to find proper source provider

¹The low order set of substream backup copies means the substream is taken from sets which are close to Cp_0 or Cp'_0 in *Robustness_Pair*.

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

than the long-latency tolerant peers, the parent peer may preempt the connection of long-latency tolerant peer by the connection of a short-latency peer which can thus enjoy the original set of downloadable substreams straightaway while the replaced long-latency tolerant peer can have more time to seek out the substreams in the subsequent sets of backup copies. Thus, more number of the short-latency peers are placed in positions close to the root of the tree where they can receive proper QoS from the overlay.

When disconnection happens, the lost substream will be immediately requested from the connection which is taken as backup copies thus providing the shortest latency. If there is no backup for the substream, the request for more candidate peers is issued to its connected peers in order to find the lost substream in the candidate peers. Peers also periodically exchange information on their connected peers and increase the peers' contact with other unknown peers. It helps peers find better source providers so that resource can be more efficiently tapped and allows the resources to be allocated to these resource-barren peers.

When a new connection is added to the source pool or the availability of the substream changes in the source pool, the optimal substream downloading solution from the network may not be consistent with the existing substream downloading request. If the parent peer changes its substream downloading combination, the new latency may be longer than the old one. Though the new latency still satisfies the peer's own temporal requirement, the longer latency may not satisfy its descendent peers' temporal requirements, since they may be less latency tolerant than the parent peer. For example, the parent peer has expiration time at 20 seconds and the child peer has expiration time at 10 seconds. The stream arrives at the parent at a latency of 5 seconds and arrives at the child peer at a latency of 7 seconds. If a new connection is added into the parent peer's source pool and changes the optimal allocation of substream copies whereby the old stream is no longer a bandwidth sustainable copy, the backup copy which has a latency of 15 seconds is

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

taken as the downloading substream. Though the latency still satisfies the parent peer's temporal requirement, it does not satisfy the child peer's temporal requirement. Thus, the child peer may disconnect from the parent and search for new parent peers.

Under such condition, the whole branch of distribution tree for the substream is subjected to change. Its descendent peers have to disconnect from the parent, search for the new substream providers and experience unacceptable stream quality for a while. Since network dynamics is unavoidable in P2P environment, if every peer strictly complies with its own constantly changing optimal downloading combination, the stream quality cannot be stable. Oscillation in the network traffic caused by overlay churning will degrade the network service. Peers will desperately search for stable source provider but will find few and the overlay will be no longer stable. Thus, the changing of substream request should be conservative.

With reference to the previous discussion that the existing downloading request should be protected, if the new request combination has the same number of Set 0 downloadable substreams as the current one which means the current number of downloadable substreams is the same as the downloadable substreams in the new *Robustness_Pair* calculation of the changed source pool, the existing request will not be changed. If the new request combination has more number of Set 0 substreams than the current downloading request, which means the number of current downloadable substreams is smaller than the best the peer can download within the current source pool, the optimal substream download solution will be employed as the substream downloading request.

Using the example in Figure 5.12, consider the case when the 2nd connection of peer n_k failed as blotted out in Figure 5.13, the Set 1 backup copies will comprise Str_3 and Str_4 from $Conn_{k[4]}$, and the downloading requests for these two substreams are issued to $Conn_{k[4]}$. The downloading requests for all the substreams are shown as darkly shaded boxes in Figure 5.13(a). After a while, another new connection, named $Conn_{k[5]}$, joins

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Figure 5.13: Example of keeping the current request instead of adopting the optimal one. Both have the same number of original (Set 0) downloadable substreams.

Figure 5.14: Example of adopting the optimal one when there is different number of original (Set 0) downloadable substreams.

the source pool. Without changing the Set 0 combination, the substreams $Conn_{k[5]}$ can provide are: Str_1 and Str_4 as Set 1 backup copies (refer to medium shaded boxes of $Conn_{k[5]}$ in (a)) and Str_2 as Set 2 backup copies (reflected as lightly shaded box of $Conn_{k[5]}$ in (a)). The combination which can maximize the total *Robustness_Pair* is shown in Figure 5.13(b). Since Set 0 (original downloadable substream) has the same number of substreams, to protect the existing downloading connections and to avoid any further increase in latency, the current downloading request in Figure 5.13(a) is preserved and the optimal request combination in (Figure 5.13(b)) will not be adopted.

Again, using the example in Figure 5.12, consider the case when both $Conn_{k[1]}$ and $Conn_{k[2]}$ of peer n_k fail as blotted out in Figure 5.14. Str_3 and Str_4 from $Conn_{k[4]}$ and Str_5 from $Conn_{k[3]}$ are taken as Set 0, the original downloadable substreams (refer to the darkly shaded boxes) and the download request for these substreams are issued. After that, if another connection $Conn_{k[5]}$ joins the source pool, the available substreams from

$Conn_{k[5]}$ are taken as Set 1 backup copies if there is no change in downloading requests as in Figure 5.14(a). However, based on *Robustness_Pair* computation, the maximum substreams that could be downloaded from the new source pool is 5 which is larger than the existing downloadable substreams. Thus the downloading request should be changed to the optimal request in Figure 5.14(b). This example also illustrates how a bandwidth-unsustainable substream (refer to Str_1 of $Conn_{k[3]}$ in Figure 5.14(a)) becomes a downloadable substream in Figure 5.14(b) as a result of changes in the source pool.

5.6 Performance Evaluation

In order to evaluate the performance of the proposed resilient tree structure, extensive simulation is conducted in NS2 [102].

5.6.1 Simulation Setup

The stream data rate is 1Mbps and the stream is divided into five substreams to be distributed. Five 2-tier network topologies are generated by GT-ITM [103] with 1020 nodes. The backbone bandwidth varies from 2Mb to 10Mb in transit domain and the bandwidth varies from 0.25Mb to 0.75Mb in the stub domain. Three recent proposals based on tree structure overlay which are efficient in constrained temporal requirement environment, namely, CoolStream[11], LagOver[23] and ID-Host [46], are used as comparison. LagOver is a latency oriented overlay as described in Chapter 2. ID-Host is a bandwidth oriented overlay. Unlike the previous two, CoolStream is a representative pull based data exchange overlay which is used to compare the performance with push based mechanisms in tight temporal requirement environment. We analyze the peer dynamic feature from findings in [49, 50] and design two dynamic scenarios to represent the typical scenarios in P2P streaming applications.

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

In scenario A, the flash-crowd join and leave are simulated. During 30000 s of simulation time, 200 peers join the overlay in the first 100 s. Another 200 peers join the overlay in 100-s intervals starting from $t = 10000$ s. Finally, 200 peers (half of the total peers) leave the overlay in 100-s intervals starting from $t = 20000$ s. There are no peer joins and departures other than the periods mentioned above.

In scenario B, the peers' lifetime is found to be a long tail distribution from studies in [48–50] which can be modeled as Pareto distribution. The join process during the stable period follows Poisson distribution. We approximate the peer join and departure by the two distributions with real data extracted from the popular TV show documented in [49]. Five different peer join and departure cases are studied in five different network topologies respectively to avoid randomness bias. In each of these cases, 200 peers join the overlay at the beginning. Around 4500 peers join the overlay following Poisson distribution and stay on the overlay with lifetime following Pareto distribution. Some peers may leave and join again in the overlay. We compare the simulation performance for 10000 s with the other three proposals.

Each peer can have at maximum 10 connections for both uploading and downloading at the same time, and 20 temporary connections for contacting the potential unknown peers. The server can have at maximum 30 connections and 20 temporary connections.

Five levels of peer expiration times are used to provide differentiated services. The expiration times are 5s, 10s, 15s, 20s and 25s respectively. Five different peer expiration time distributions are simulated in both scenarios in order to present different degree of temporal constraints imposed on the streaming services. The percentage of expiration time distribution at each level is shown in Table 5.2.

Table 5.2: Peer Expiration Time Distribution Cases

Distribution Cases	Expiration Time				
	5s	10s	15s	20s	25s
Dist-0	0%	10%	20%	30%	40%
Dist-1	10%	15%	20%	25%	30%
Dist-2	20%	20%	20%	20%	20%
Dist-3	30%	25%	20%	15%	10%
Dist-4	40%	30%	20%	10%	0%

5.6.2 Simulation Results

Since the simulation results in the five different topologies present consistent trends in the comparison, we only show a set of results from one topology to illustrate the performances of our overlay as compared to the other proposals.

5.6.2.1 Stream Quality

Stream quality is defined as the ratio of the amount of stream data that can be played back without violating the expiration time of a peer over the total amount of stream data within a specified time interval. The average stream quality refers to the average of all peers' stream quality in the overlay.

The performance of the four different overlay schemes (ID-Host (blue), LagOver (light green), CoolStream (yellow) and ours (red)) in scenario A with Dist-2 expiration time are presented in Figure 5.15. During the first 100 s, 200 peers join the overlay. Figure 5.15(b) shows the detail snapshot during the initial 1000s of simulation. Most of the four schemes become stable around 500 s, with the highest average stream quality registered by ours at 0.74 and the lowest quality registered for CoolStream at 0.15. ID-Host provides a stream quality of 0.67 at that time and an average stream quality of around 0.7 in the first 10000 s after the overlay is stable. Since both LagOver and our scheme consider the different temporal requirement in the overlay construction, the two scheme provides similar stream quality in the first 10000 s at stream quality of around 0.78. The reason CoolStream

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

performs much worse than the other three is mainly because the receiver-driven pull-based data request mechanism does not fit the stringent temporal requirement of Dist-2. In the experiment, three additional simulations for CoolStream with loose temporal requirements are conducted where all peers' expiration time are set to 25 s (CoolStream-25s), 50 s (CoolStream-50s) and 75 s (CoolStream-75s). Note that the black plot is the only one close to our green one and this represents CoolStream-75s. This shows that our overlay is much more adaptive and can manage to meet tight temporal constraints. The average received quality in pull-based data exchange mechanism like CoolStream is compatible to the quality received in push-based data exchange mechanism with Dist-2 peer expiration time distribution only when the temporal requirement is loosen to 75 s for all peers. With such loose temporal requirements, all push based overlays can easily obtain an average quality near to 1.

Starting from 10000s, another 200 new peers join the network in 100-s interval with snapshot details shown in Figure 5.15(c). Our scheme is the fastest to achieve stability after 300 s. ID-Host takes around 1000 s to stabilize but at more than 5% lower average quality than ours. It takes LagOver 1000 s to be stable and the oscillation after time 11000 s indicates that LagOver is not good at adapting quickly to the changes in the network and overlay. CoolStream performs the worst when new peers join. It is mainly because most of the existing peers do not have the proper quality to support the new peers. Meanwhile, the more number of peers will add to the delivery latency of the stream data. Since the delivery time in pull-based mechanism is much longer than the push mechanism, it takes a much longer time for the stream data to arrive at each peer and hence the stream data violating the expiration time at the peers is more common for the pull-based mechanism.

At time 20000 s, 200 peers leave the overlay within 100 s as shown in the detail snapshot of Figure 5.15(d). Most of the remaining peers experience connection failures

Figure 5.15: Average Stream Quality for all proposals under Dist-2 expiration time distribution and loosened temporal constraint for Coolstream

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

and fluctuating network dynamics. However, due to the resilience of our proposed source pool connection selection scheme, the average quality drops to 0.62 and it recovers after 100 s. In bandwidth oriented ID-Host, the average quality drops to 0.48 and it recovers after 400 s. In LagOver, the average quality drops to 0.47 and it recovers to 0.7 after 150 s while the overlay continues to slowly adapt to the changes as indicated by the oscillation after time 20300 s. For CoolStream, the average quality recovers to the same level as the first 10000 s at around 0.15 since fewer peers mean less contention and the stream delivery path from server to peer is also shortened. The quality of CoolStream-75s drops less than our scheme when 50% of the peers leave the overlay. It is mainly because the long expiration time mitigates the negative effects of connection failures. When the expiration time is extended to 75 s for push-based overlays (i.e. ID-Host, LagOver and ours), the drop in quality is much less than CoolStream-75s. Thus, the pull-based overlay (i.e. CoolStream) is not suitable for stringent temporal requirement in live or near-live streaming applications.

Figure 5.16: Average Stream Quality under 5 Different Expiration Time Distributions

The average stream quality in Figure 5.16 is determined by averaging all the peers' received stream quality during the entire simulation duration in scenario B. The three push-based overlay schemes are compared under different latency distribution cases. When

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

the expiration time is more stringent, the average stream quality for peers in the overlay drops. Our proposed overlay scheme achieves the best quality under different temporal requirements; ID-Host achieves the second best and LagOver performs the worst.

Our overlay construction scheme provides peers with better substream downloading combination and more balanced network resource allocation. In the peer churning scenario, the impact of disconnection and congestion could be quickly mitigated by changing the substream downloading to the connections which can provide the backups in the source pool. It tries to alleviate the impact on the tree branch due to peer and network dynamics and reduces the negative impact on the descendent peers. Since it is more urgent for short-latency tolerant peers to find proper substream providers, when a suitable source who can provide the missing substream is found, the substream provider will give preference to the short-latency tolerant peer. If a long-latency tolerant peer has other backup copies from other connections, it is likely to be preempted by the short-latency tolerant peer at its current parent. For the long-latency tolerant peers, it is easier for them to find other backup sources under our schemes given its longer latency.

In ID-Host, since it is a bandwidth oriented overlay, resourceful peers are given preference in the overlay. It leads to contention for resources in the overlay. The resource-abundant peers are more likely to connect to other resource-abundant peers. The network resource is not fairly distributed. The disconnection at the resource-barren peer will cause more serious degradation to the peer's stream quality. Meanwhile, since the structure is less concern about the latency, short-latency tolerant peers may fail to gain preference for their substream download requests. Thus, it is hard for short-latency tolerant peers to get proper stream quality while contending with long-latency tolerant peers.

In LagOver, though its structure focuses on placing the resourceful peers close to the root of the tree as well as to satisfy the short-latency tolerant peers, the aim of supporting peers as much as possible makes LagOver place peers at the positions in the tree where

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

the stream data arrives just on time. The tree structure built in LagOver is somewhat fragile. A small congestion through the connection may render a stream to be overdue, and leads to a switch at the downloading connection. It is the reason that LagOver takes a relatively long time to stabilize in scenario A as in Figure 5.15, and it also achieves the lowest stream quality in scenario B as in Figure 5.16.

5.6.2.2 Start-Up Latency

Since scenario B is closer to the real operating scenario, the subsequent results will just focus on the performance of the different overlays in scenario B.

The start-up latency of peers is defined as the duration from the point a peer joins the overlay until it can receive x substreams at a stable stream quality set at above 80%. In other words, more than 80% of a requested substream is received by the peers within the expiration time. The start-up latency for three substreams and five substreams are presented in Figures 5.17(a) and (b). Over 82% of peers can receive 3 stable substreams under our proposed overlay scheme after they join the overlay for 150 seconds. ID-Host and LagOver could only allow 74% and 45% of peers to receive 3 stable substreams at $t = 150$ s respectively. The start-up latency for receiving 5 substreams is longer as expected. Our scheme allows over 80% of peers to receive all 5 stable substreams in 270 seconds compared to near 70% of peers in ID-Host and 41% of peers in LagOver.

Figure 5.17: CDF for Start-up Latency Dist-2 Time Expiration in Scenario B

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

Our overlay prioritizes Set 0 downloadable substreams as the most important in the *Robustness_Pair* computation. Thus, for the new peers, it is given more priority to reserve the resources from its parent peer. If the existing connection in the parent peer is preempted, it is more likely that the connection to the new child peer is more important than the connection to the preempted peer. The lost substream can be recovered from the displaced peers' backup sets of substreams in the source pool. LagOver performs the worst among the three because of its relatively fragile tree structure. The unstable stream quality at the existing parent peer causes the newly joined peer to take a longer time to find a stable substream provider. ID-Host performs worse than our overlay due mainly to the fact that the newly joined peer needs to contend with the existing peers based on its resources contribution. Since it takes time for newly joined peers to evaluate the maximum resources it can contribute to the overlay, it is thus hard for the new peer to compete with the existing peers which have already discovered their maximum contribution ability. Thus, it takes a longer time to obtain three stable substreams than our overlay.

5.6.2.3 Recovery Time

The recovery time is a measure of the robustness of the overlay. It is defined as the time from the detection of a quality degradation event to the time it recovers to its maximum stream quality before the occurrence of the event. Quality degradation events include any conditions that cause the stream quality to drop below its maximum quality. Examples are bandwidth congestion, peer failure, quality degradation at parent peer, etc.

We collect the time taken for all peers to recover to their maximum stream quality achieved before the events occur in Scenario B. We then compute the cumulative distribution for the percentage of events (in y axis) that are recovered before the recovery time (in x axis) as shown in Figure 5.18. Near 80% of the stream quality degradation

CHAPTER 5. RESILIENT STREAMING OVERLAY SCHEME

events are recovered within 10 seconds in our overlay. There is a little more than 60% of the quality degradation events that can be recovered within 10 seconds in LagOver and ID-Host.

Figure 5.18: CDF for Recovery Time under Dist-2 Expiration Distribution in Scenario B

The *Robustness_Pair* calculation in our overlay helps peers manage their source pool of connections efficiently which can provide the most downloading substreams as well as giving the most backup options to the stream without bandwidth conflicts. For LagOver and ID-Host, the degraded substream could only be recovered from the existing connections which are selected without special consideration for backup resource reservation. If the existing connection cannot provide certain substreams, peers have to refer to the server for more peer candidates and start the rejoin process to graft themselves to the substream distribution tree again. It is for this reason that the two overlay structures exhibit similar characteristics in terms of recovery time and accounts for why the time for recovery from substream degradation in our overlay is shorter than the other two.

5.6.2.4 Overhead Measurement

The overhead for peer connection testing, feedback, data availability publication and retransmission request as in Chapter 4 are collected for the five different peer expiration

time distribution cases. Overhead load is calculated as a percentage of the stream data rate. The average overhead load of all peers is presented in Table 5.3.

Table 5.3: Overhead Load of Different Overlays under Different Expiration Time

Distribution cases	ID-Host	LagOver	Ours
Dist-0	3.03%	2.77%	3.05%
Dist-1	2.99%	2.71%	3.01%
Dist-2	2.93%	2.64%	2.97%
Dist-3	2.91%	2.62%	2.96%
Dist-4	2.86%	2.57%	2.95%

The overhead load for different overlays are almost similar, hovering around 3% under the different peer expiration time scenarios. Our proposed scheme incurs a slightly higher overhead relative to the others. This is more than acceptable given its improvements in stream quality, short join time and quick recovery compared to the other proposals. All three schemes show a trend of decreasing overhead load as the temporal requirement becomes tighter. This is due to the degradation in stream quality as there are fewer proper source providers to connect to which in turn reduces the connection monitoring and testing overheads.

5.6.3 Discussion

As stability and convergence time are highly reliant on peer dynamics and network conditions, comprehensive simulation has been carried out under realistic scenarios with peer churn and different expiry time requirements at the peers to investigate the performance of the algorithms as compared to other proposals. Simulation results show that our algorithms achieve significant improvements in terms of stability and convergence time compared to the other proposals. These are reflected in the improvements in stream quality, peer join time and recovery from degradation in stream quality which can be a result of peer join, peer departure, peer failure, network congestion, degradation of stream quality from parents etc.

The performance improvement can be explained due to the distributed nature of the algorithm. Information required for robustness calculation is all available locally. The search space for the most robust connection is thus relatively small and the computation complexity is manageable. Moreover, a heuristic has been incorporated to reduce the convergence time to facilitate fast convergence given the iterative nature of the algorithm.

Given that the performance improvement is achieved under simulation environments which are modeled upon realistic situations with peer churn and different expiry time requirements, the feasibility of real deployment and stability of the algorithm are thus vindicated.

5.7 Summary

In this chapter, we propose a new evaluation criterion to measure the quality of a peer's connections. Unlike other common criteria, our evaluation for the quality of peer connections is based on how it can be complementary to the other connections rather than based on a certain characteristic of the individual connections. The evaluation method helps peers to discover the best and robust stream downloading connection as well as the most complete backup copies for the whole stream within the source pool of their connections. In order to fairly allocate network resources throughout the overlay, the number of connections at each peer should be limited. Our evaluation criterion also helps to find the best combination of connections within the limited connection number. The decision of admitting a new connection to a source pool is based on whether the new connection can improve the stream quality and robustness of all affected peers in total. The optimization mechanism allows resource barren peers to more efficiently utilize network resource. Simulation results show that both the stream quality and the overlay resiliency are improved in our overlay construction scheme compared to others.

Chapter 6

Conclusion

This chapter concludes the dissertation by reviewing the contributions of the research work and presenting the directions for future research in this area.

6.1 Contributions

In this dissertation, we have studied the functions of several important components of streaming service through P2P overlay in a dynamic network environment. Since the dynamics in the network and peer behavior are uncontrollable and unpredictable, the quality of streaming service on the P2P overlay is mainly determined by the efficiency of stream data delivery at each peer and the organization of peers in the overlay. Based on this analysis, we proposed three solutions for these important components in P2P streaming applications, i.e., scheduling, data exchange mechanism, and overlay construction. The proposed solutions make the following contributions:

- Our proposed scheduling algorithm aims at improving the efficiency of queued data delivery in a fluctuating network environment. The algorithm considers both the content of the stream data and the overlay structure. It prioritizes the sending of stream data which are critical to the entire stream and have a large number of descendent peers. The scheduling algorithm minimizes the data queuing time for

CHAPTER 6. CONCLUSION

the whole system and thereby improves the overall stream latency. The stream data arrives at peers at an earlier time than commonly used scheduling algorithms, and, thus, further improves the stream quality by allowing more stream data to arrive before their expiration time. In fluctuating network, peers can adapt the data rate to the variable bandwidth and forward the most important data first. For systems where the streaming encoder is unknown, the algorithm can also mitigate the negative impact of a mismatch in the actual importance of the various encoded data streams vis-a-vis the weighting importance used in our scheduling algorithms. This is due to the increased probability of successful recovery of the data streams at the peers as a result of the scheduling algorithm's ability to send more data at a lower latency to all the peers.

- In Chapter 4, we propose a light weight retransmission mechanism to improve the stream quality in a lossy network environment. It incorporates immediate data forwarding as in push mechanism and periodic data availability notification with responsive data retrieval as in pull mechanism. The benefits from both push and pull mechanisms help our proposed technique to keep the data exchange as efficient as push mechanism and yet enjoy the same data recovery ability as pull mechanism. By prioritizing the retransmissions according to the proposed scheduling algorithm and exchanging only data index information, overhead is maintained at a low level.
- A new robustness measurement criterion for every peer is proposed based on a peer's existing connections in Chapter 5. Based on the robustness measurement, the criterion helps peers compare the connection quality among all their existing connections in a source pool rather than just comparing connections individually. The connection which can provide the bandwidth to backup the scarcest substream source is given preference. An optimization heuristic strategy is proposed for peers

CHAPTER 6. CONCLUSION

to find the best combination of connections which can provide better and resilient streaming services without incurring too much latency. The cooperative operation in peer membership management helps peers avoid contention for the limited network resources and utilize the available resources prudently in dynamic peer churning environment.

The improvement in stream quality, peer join time and recovery from degradation in stream quality stems from the better utilization of the stream data itself, the efficient exploitation of additional information from the neighbors' buffer as well as own resource optimization. Thus, it introduces additional communication overhead and consumes computational resource for optimization calculation. The overhead is actually negligible compared to the amount of stream data and the improvements, albeit it being slightly higher than proposals such as ID-Host and LagOver. However, since the overhead is generated among neighbors, its global impact is thus confined. As the computation is done locally and heuristically, extra resource needed for the improvement is thus not significant compared to the improvements achieved.

6.2 Future Work

6.2.1 Merging of Multiple-Stream Overlays

Currently, most P2P streaming schemes proposed are for a single server providing service for one media stream. To support multiple streams, the server has to create multiple independent streaming sessions and stream data through separate channels. In real applications, it often happens that the popular media streams are more likely to provide better quality since there are more participants in the service session and peers are more likely to find many good source suppliers. Users, who join the less popular media stream, cannot receive the data stream at the desirable quality. The low quality degrades user

CHAPTER 6. CONCLUSION

experience and results in frequent join and departure. The high peer dynamics will impact the overlay efficiency and further degrades the service quality. This vicious cycle eventually leads to unbearable service quality in that session. Even though there is an increasing number of users interested in the session, it will take a long time for the overlay to evolve and stabilize from the initial stage of a few participants to many participants. The same applies when the number of peers in one media streaming session is decreasing. The variable number of users in a service session affects the content quality it is serving. Users will quit the session when the content quality does not match their expectation and search in the other sessions. Though the number of users in one session may change drastically, the total number of users participating in all sessions will not fluctuate so much.

Hence, it may be better to build one overlay for all the users in the network to enjoy different media streaming services. To improve the performance of a service session where there are few users, it is necessary to break the isolation between the different sessions. Every participant should not only contribute to their own session, but also to other sessions in order to help their neighbors enjoy better service quality. Although it requires participants to contribute more than they receive at times, the limited increase will not affect their enjoyment of the services. An analogy of a successful application is Skype [104], where users will not notice their extra contribution while having audio and video services. Of course for media streaming, serving multiple streams to a large group of participants in one overlay is different from Skype where most communication is confined to two participants and only requires the help from peers along the path of the two participants. The issue is how to decide which peer to help when multiple requests from different sessions are received. The P2P overlay construction and data placement at each peer need to take the surrounding peers' capacity and current status into consideration and decide on how to cooperate with the neighbor peers without

incurring high overhead of redundancy. Our proposed scheduling algorithm could also help to decide how to create the overlay based on evaluating the requests.

6.2.2 Unification of Live Streaming and Video on Demand Streaming Overlays

The video on demand (VoD) streaming service is also a very hot topic in both the research and the business world. Users of VoD stream service, like YouTube [105] and Hulu [106], grow astronomically. Internet users in USA viewed 14.8 billion online videos in January 2009, according to research by comScore [107]. YouTube currently has generated more traffic than the entire Internet bandwidth of year 2000. Moving the web-based VoD service into a P2P system could better utilize the network resources and save the bandwidth expense of the service providers. It will also allow the efficiency and scalability of P2P system to be exploited by VoD services.

However, although both live streaming and VoD streaming applications provide streaming services, there is a big gap between the two applications which hinder the provision of a unified service by service providers. VoD users join the streaming session at different times and start from different parts of the stream data. Such peer behavior is not compatible with live streaming where every user in the session starts from almost the same point of the data stream. Moreover, live streaming users usually discard data after the expiration time and hence it is unlikely that VoD users are able to fetch the expired data from live streaming peers. Hence, it is hard for users to issue VoD request in live streaming application. Since the VoD stream data could only be available after they are fully uploaded to the server, the timeliness of live streaming is compromised as live streaming users cannot enjoy the immediate services while the data is being uploaded. Moreover, VoD systems have to support user-interactivity such as playback, fast forward, pause etc. which are not supported in live streaming and contradict the definition of live

streaming. For a start, such user interactivity will have to be suppressed or restricted to realize this unification.

The gap between live and VoD streaming can be mitigated by increasing the buffer size and retaining the cached data at live streaming peers. But since the live streaming peer cannot hold the entire stream data and performs the same function as the streaming server, it is important to design a data assignment algorithm in order to maintain the robustness and data integrity at peers. Bandwidth resource allocation between live streaming request and VoD streaming request is also important. The proportion of users in the two different groups will change over time. An adaptation mechanism is necessary to allow the overlay to adjust to such changes in user access patterns quickly and smoothly without affecting the users' viewing quality.

6.3 Summary

In summary, we proposed a scheduling algorithm which is both content and overlay-aware for the P2P streaming system. Data which is requested by more number of peers and more important to the overall stream is given priority to be sent out thus minimizing the average latency of the streaming data to all users. A data exchange mechanism which combines the merits of data-push mechanism and retransmission-pull mechanisms is proposed to improve data integrity in a lossy and dynamic environment. Retransmissions are prioritized to maintain good stream quality and low overhead. Finally, a cooperative yet distributed overlay construction scheme is proposed for optimal peer connection selection. The usefulness of a new connection is computed based on an existing source pool rather than evaluating the individual connections. The three contributions result in a P2P streaming system that is more efficient and resilient.

References

- [1] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended lans," *ACM Trans. Comput. Syst.*, vol. 8, no. 2, pp. 85–110, May 1990.
- [2] "BitTorrent homepage," <http://www.bittorrent.com/>.
- [3] "eMule Homepage," <http://www.emule.com/>.
- [4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 31, no. 4. New York, NY, USA: ACM, October 2001, pp. 161–172.
- [5] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. London, UK: Springer-Verlag, 2001, pp. 329–350.
- [6] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, pp. 41–53, 2004.
- [7] "PPlive homepage," <http://www.pplive.com>.
- [8] "PPStream homepage," <http://www.ppstream.com>.
- [9] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: high-bandwidth multicast in cooperative environments," in *SOSP*

REFERENCES

- '03: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 298–313.
- [10] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, “Resilient peer-to-peer streaming,” in *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, 2003, pp. 16–27.
- [11] X. Zhang, J. Liu, B. Li, and Y. S. P. Yum, “Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming,” in *24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, 2005, pp. 2102–2111.
- [12] C. Wu and B. Li, “Strategies of conflict in coexisting streaming overlays,” in *26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, 2007, pp. 481–489.
- [13] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, “Scalable application layer multicast,” in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 32, no. 4. New York, NY, USA: ACM Press, October 2002, pp. 205–217.
- [14] R. Rejaie and A. Ortega, “Pals: peer-to-peer adaptive layered streaming,” in *NOSS-DAV '03: Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. New York, NY, USA: ACM Press, 2003, pp. 153–161.
- [15] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, “Almi: An application level multicast infrastructure,” in *Proceedings of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, San Francisco, CA, USA, March 2001, pp. 49–60.
- [16] Z. Xu, C. Tang, S. Banerjee, and S. J. Lee, “Rita: receiver initiated just-in-time tree adaptation for rich media distribution,” in *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2003, pp. 50–59.

REFERENCES

- [17] J. Aspnes and G. Shah, "Distributed data structures for peer-to-peer systems," in *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*. CRC Press, 2005, pp. 685–700.
- [18] J. Vogel, J. Widmer, D. Farin, M. Mauve, and W. Effelsberg, "Priority-based distribution trees for application-level multicast," in *NETGAMES '03: Proceedings of the 2nd workshop on Network and system support for games*. New York, NY, USA: ACM Press, 2003, pp. 148–157.
- [19] C. K. Yeo, B. S. Lee, and M. H. Er, "A framework for multicast video streaming over ip networks," *J. Netw. Comput. Appl.*, vol. 26, no. 3, pp. 273–289, August 2003.
- [20] C. K. Yeo, B.-S. Lee, and M. H. Er, "An overlay for ubiquitous streaming over internet," in *NETWORKING '02: Proceedings of the Second International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; and Mobile and Wireless Communications*. London, UK: Springer-Verlag, 2002, pp. 1239–1244.
- [21] C. K. Yeo, B. S. Lee, and M. H. Er, "A peering architecture for ubiquitous ip multicast streaming," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. 3, pp. 82–95, July 2002.
- [22] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 2, 2003, pp. 1521–1531.
- [23] A. Datta, I. Stoica, and M. Franklin, "Lagover: Latency gradated overlays," in *27th International Conference on Distributed Computing Systems (ICDCS '07)*, 2007, pp. 1–13.
- [24] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient multicast using overlays," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 102–113, June 2003.

REFERENCES

- [25] S. Banerjee, S. Lee, R. Braud, B. Bhattacharjee, and A. Srinivasan, "Scalable resilient media streaming," in *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM Press, 2004, pp. 4–9.
- [26] S. Lee, R. Sherwood, and B. Bhattacharjee, "Cooperative peer groups in nice," *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 2, pp. 1272–1282, 2003.
- [27] D. A. Tran, K. A. Hua, and T. Do, "Zigzag: an efficient peer-to-peer scheme for media streaming," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 2, 2003, pp. 1283–1292.
- [28] J. Liang and K. Nahrstedt, "Dagstream: locality aware and failure resilient peer-to-peer streaming," in *Multimedia Computing and Networking 2006*, S. Chandra and C. Griwodz, Eds., vol. 6071, no. 1. SPIE, 2006, pp. 1–15.
- [29] J. Liang and K. Nahrstedt, "Randpeer: Membership management for qos sensitive peer-to-peer applications," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006, pp. 1–10.
- [30] Y. H. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *Measurement and Modeling of Computer Systems*, 2000, pp. 1–12.
- [31] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2001, pp. 55–67.
- [32] S. Jain, R. Mahajan, D. Wetherall, G. Borriello, and S. D. Gribble, "A comparison of large-scale overlay management techniques," *Technical Report UV-CSE 02-02-02*, February 2002.

REFERENCES

-
- [33] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," *Lecture Notes in Computer Science*, vol. 2233, pp. 1–14, 2001.
 - [34] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. T. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1489–1499, 2002.
 - [35] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," *Peer-to-Peer Systems IV*, pp. 127–140, 2005.
 - [36] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 1415–1423.
 - [37] N. Magharei and R. Rejaie, "Understanding mesh-based peer-to-peer streaming," in *NOSSDAV '06: Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*. ACM, 2006, pp. 1–6.
 - [38] Q. Sun and D. C. Sturman, "A gossip-based reliable multicast for large-scale high-throughput applications," in *Dependable Systems and Networks, 2000. Proceedings International Conference on*, 2000, pp. 347–358.
 - [39] C. Tang, R. N. Chang, and C. Ward, "Gocast: gossip-enhanced overlay multicast for fast and dependable group communication," in *Dependable Systems and Networks, 2005. Proceedings. International Conference on*, 2005, pp. 140–149.
 - [40] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: peer-to-peer media streaming using collectcast," in *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*. New York, NY, USA: ACM Press, 2003, pp. 45–54.
 - [41] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast," in *Network Protocols, 2006. ICNP*

REFERENCES

- '06. *Proceedings of the 2006 14th IEEE International Conference on*, 2006, pp. 2–11.
- [42] H. M. Radha, M. van der Schaar, and Y. Chen, "The mpeg-4 fine-grained scalable video coding method for multimedia streaming over ip," *Multimedia, IEEE Transactions on*, vol. 3, no. 1, pp. 53–68, 2001.
- [43] Y. Wang, A. R. Reibman, and S. Lin, "Multiple description coding for video delivery," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 57–70, 2005.
- [44] H. Guo, K. T. Lo, and C. T. Cheng, "Overlay networks construction for multi-layered live media streaming," in *International Symposium on Multimedia*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 427–436.
- [45] S. Ganguly, A. Saxena, S. Bhatnagar, R. Izmailov, and S. Banerjee, "Fast replication in content distribution overlays," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4, 2005, pp. 2246–2256.
- [46] M. Bishop, S. Rao, and K. Sripanidkulchai, "Considering priority in overlay multicast protocols under heterogeneous environments," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*, 2006, pp. 1–13.
- [47] L. Dai, Y. Cui, and Y. Xue, "Maximizing throughput in layered peer-to-peer streaming," in *Communications, 2007. ICC '07. IEEE International Conference on*, 2007, pp. 1734–1739.
- [48] F. Wang, J. Liu, and Y. Xiong, "Stable peers: Existence, importance, and application in peer-to-peer live video streaming," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008, pp. 1364–1372.
- [49] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale p2p iptv system," *Multimedia, IEEE Transactions on*, vol. 9, no. 8, pp. 1672–1687, 2007.

REFERENCES

- [50] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM Press, 2004, pp. 41–54.
- [51] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006, pp. 1–10.
- [52] Y. Liu, L. Xiao, X. Liu, L. M. Ni, and X. Zhang, "Location awareness in unstructured peer-to-peer systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 2, pp. 163–174, 2005.
- [53] T. Silverston and O. Fourmaux, "Source vs data-driven approach for live p2p streaming," in *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/I-CONS/MCL 2006. International Conference on*, 2006, pp. 99–99.
- [54] M. Zhang, Y. Xiong, Q. Zhang, L. Sun, and S. Yang, "Optimizing the throughput of data-driven peer-to-peer streaming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 1, pp. 97–110, 2009.
- [55] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang, "On the optimal scheduling for media streaming in data-driven overlay networks," in *IEEE Global Telecommunications Conference*, 2006, pp. 1–5.
- [56] N. Magharei and R. Rejaie, "Adaptive receiver-driven streaming from multiple senders," *Multimedia Systems*, vol. 11, no. 6, pp. 550–567, June 2006.
- [57] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," in *Distributed Computing Systems, the 22nd International Conference on*, 2002, pp. 363–371.
- [58] J. B. Kwon and H. Y. Yeom, "Distributed multimedia streaming over peer-to-peer networks," in *Euro-Par 2003 Parallel Processing*, 2003, pp. 851–858.

REFERENCES

-
- [59] F. Chen and V. Kalogeraki, "Ruben: a technique for scheduling multimedia applications in overlay networks," in *GLOBECOM '04: IEEE Global Telecommunications Conference*, vol. 2, 2004, pp. 1254–1258.
 - [60] E. Setton, J. Noh, and B. Girod, "Rate-distortion optimized video peer-to-peer multicast streaming," in *P2PMMS'05: Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming*. New York, NY, USA: ACM Press, 2005, pp. 39–48.
 - [61] P. Baccichet, J. Noh, E. Setton, and B. Girod, "Content-aware p2p video streaming with low latency," in *Multimedia and Expo, 2007 IEEE International Conference on*, 2007, pp. 400–403.
 - [62] E. Setton and B. Girod, "Rate-distortion analysis and streaming of sp and si frames," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, no. 6, pp. 733–743, 2006.
 - [63] E. Setton and B. Girod, "Congestion-distortion optimized scheduling of video over a bottleneck link," in *Multimedia Signal Processing, IEEE 6th Workshop on*, 2004, pp. 179–182.
 - [64] J. Li, C. K. Yeo, and B. S. Lee, "Peer-to-peer streaming scheduling to improve real-time latency," in *Multimedia and Expo, 2007 IEEE International Conference on*, 2007, pp. 36–39.
 - [65] J. Li and C. K. Yeo, "Content and overlay-aware scheduling for peer-to-peer streaming in fluctuating networks," *J. Netw. Comput. Appl.*, vol. 32, no. 4, pp. 901–912, July 2009.
 - [66] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 31, no. 4. New York, NY, USA: ACM, October 2001, pp. 149–160.

REFERENCES

- [67] D. A. Helder and S. Jamin, "Banana tree protocol, an end-host multicast protocol," *Technical Report CSE-TR-429-00*, July 2000.
- [68] P. Francis, "Yoid: Extending the internet multicast architecture," <http://www.isi.edu/div7/yoid/>, April 2000.
- [69] W. T. Ooi, "Dagster: contributor-aware end-host multicast for media streaming in heterogeneous environment," *Proceeding of SPIE*, vol. SPIE-5680, pp. 77–90, 2005.
- [70] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: a framework for delivering multicast to end users," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1366–1375.
- [71] J. Jannotti, D. K. Gifford, K. L. Johnson, F. M. Kaashoek, and J. W. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, October 2000, pp. 197–212.
- [72] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1190–1199.
- [73] V. Vishnumurthy and P. Francis, "On heterogeneous overlay construction and random node selection in unstructured p2p networks," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*, 2006, pp. 1–12.
- [74] Y. D. Chawathe, "Scattercast: an architecture for internet broadcast distribution as an infrastructure service," *Ph.D Thesis*, 2000.
- [75] J. Postel, "Internet control message protocol," *RFC 792*, September 1981.
- [76] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications," *RFC 3550*, July 2003.

REFERENCES

- [77] L. Mathy, R. Canonico, and D. Hutchison, "An overlay tree building control protocol," in *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*. London, UK: Springer-Verlag, 2001, pp. 76–87.
- [78] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and white-washing in peer-to-peer systems," in *PINS '04: Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*. New York, NY, USA: ACM, 2004, pp. 228–236.
- [79] M. Afergan and R. Sami, "Repeated-game modeling of multicast overlays," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*, 2006, pp. 1–13.
- [80] M. Feldman and J. Chuang, "Overcoming free-riding behavior in peer-to-peer systems," *SIGecom Exch.*, vol. 5, no. 4, pp. 41–50, July 2005.
- [81] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and white-washing in peer-to-peer systems," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 5, pp. 1010–1019, 2006.
- [82] V. Vishnumurthy, S. Chandrakumar, and E. Sirer, "Karma: A secure economic framework for peer-to-peer resource sharing," in *Workshop on the Economics of Peer-to-Peer Systems*, 2003.
- [83] E. P. Lazear and S. Rosen, "Rank-order tournaments as optimum labor contracts," *Journal of Political Economy*, vol. 89, no. 5, pp. 841–864, October 1981.
- [84] A. Habib and J. Chuang, "Service differentiated peer selection: an incentive mechanism for peer-to-peer media streaming," *Multimedia, IEEE Transactions on*, vol. 8, no. 3, pp. 610–621, 2006.
- [85] A. Habib, D. Xu, M. Atallah, B. Bhargava, and J. Chuang, "Verifying data integrity in peer-to-peer media streaming," *Proc. SPIE*, vol. SPIE-5680, pp. 1–12, 2005.

REFERENCES

-
- [86] B. Li, G. Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin, "An empirical study of flash crowd dynamics in a p2p-based live video streaming system," in *IEEE Global Telecommunications Conference*, 2008, pp. 1–5.
 - [87] X. Yang and G. de Veciana, "Service capacity of peer to peer networks," in *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 4, 2004, pp. 2242–2252.
 - [88] Y.-C. Tu, J. Sun, M. Hefeeda, and S. Prabhakar, "An analytical study of peer-to-peer media streaming systems," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 4, pp. 354–376, November 2005.
 - [89] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications*, 2007, pp. 919–927.
 - [90] T. Li, M. Chen, D.-M. Chiu, and M. Chen, "Queuing models for peer-to-peer systems," in *8th International Workshop on Peer-to-Peer Systems (IPTPS '09)*, April 2009.
 - [91] F. Liu, B. Li, L. Zhong, B. Li, and D. Niu, "How p2p streaming systems scale over time under a flash crowd?" in *The 8th International Workshop on Peer-to-Peer Systems (IPTPS'09)*, April 2009.
 - [92] A. Fast, D. Jensen, and B. N. Levine, "Creating social networks to improve peer-to-peer networking," in *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. New York, NY, USA: ACM, 2005, pp. 568–573.
 - [93] N. Borch, "Social peer-to-peer for social people," in *International Conference on Internet Technologies and Applications*, September 2005.
 - [94] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, July 1998, pp. 43–52.

REFERENCES

- [95] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips, "Tribler: a social-based peer-to-peer system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 2, pp. 127–138, February 2008.
- [96] Z. He, J. Cai, and C. W. Chen, "Joint source channel rate-distortion analysis for adaptive mode selection and rate control in wireless video coding," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 12, no. 6, pp. 511–523, 2002.
- [97] "SopCast homepage," <http://www.sopcast.com>.
- [98] "Brite: An approach to universal topology generation," <http://www.cs.bu.edu/brite/>.
- [99] C. K. Yeo, B. S. Lee, and M. H. Er, "Application layer multicast architecture for media streaming," in *Seventh IASTED International Conference on Internet and Multimedia Systems and Applications*, 2003, pp. 455–460.
- [100] Y. Dinitz, "Dinitz' algorithm: The original version and even's version," *Theoretical Computer Science*, pp. 218–240, 2006.
- [101] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network." *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.
- [102] "NS2 Homepage," <http://www.isi.edu/nsnam/ns/>.
- [103] "Gt-itm: Modeling topology of large internetworks," <http://www.cc.gatech.edu/projects/gtitm/>.
- [104] "Skype Homepage," <http://www.skype.com>.
- [105] "YouTube Homepage," <http://www.youtube.com>.
- [106] "Hulu Homepage," <http://www.hulu.com>.
- [107] "Youtube surpasses 100 million u.s. viewers for the first time." [Online]. Available: <http://www.comscore.com/press/release.asp?press=2741>

List of Publications

Journal

1. Jiaming Li, and Chai Kiat Yeo, Content and overlay-aware scheduling for peer-to-peer streaming in fluctuating networks, *Journal of Network and Computer Applications*, Vol. 32, Issue 4, Pages 901-912, 2009
2. Jiaming Li, and Chai Kiat Yeo, Dynamic Transmission Scheduling for Multimedia Streaming via P2P Overlay, *Journal of Signal Processing Systems*, Special Issue, 2009
3. Jiaming Li, and Chai Kiat Yeo, A Resilient Overlay for Low Latency Tolerant Streaming Applications, *IEEE Transaction on Multimedia*, submitted

Conference

1. Jiaming Li, Chai Kiat Yeo, and Bu Sung Lee, Peer-to-peer streaming scheduling to improve real-time latency, in the proceeding of the *IEEE International Conference on Multimedia and Expo 2007 (ICME 2007)*, July 2007, 36-39
2. Jiaming Li, Chai Kiat Yeo, and Bu Sung Lee, Fast Scheduling on P2P Streaming Overlay. in proceeding of the *Second International Conference on Ubiquitous Information Management and Communication 2008 (ICUIMC 2008)*, Invited paper, January, 2008, pp. 242-246
3. Jiaming Li, Chai Kiat Yeo, and Bu Sung Lee, Content and Overlay-Aware Transmission Scheduling in Peer-to-Peer Streaming, in the proceeding of *IEEE Global Communication Conference 2008 (GlobeCom 2008)*, November, 2008, pp. 1-5