Adaptation and Robustness in Peer-to-Peer Streaming

Anh Tuan Nguyen
Department of Informatics
University of Oslo [1]

January 25, 2011

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1075

ISSN 1501-7710

Cover: Inger Sandved Anfinsen.
Printed in Norway: AIT Oslo AS.

Produced in co-operation with Unipub.
The thesis is produced by Unipub merely in connection with the
thesis defence. Kindly direct all inquiries regarding the thesis to the copyright
holder or the unit which grants the doctorate.

Abstract

The rapid development of network communication infrastructure enables networked multimedia streaming applications ranging from on-demand video streaming to highly interactive video conferencing. Peer-to-Peer (P2P) technologies have emerged as a powerful and popular paradigm for bringing such emerging multimedia services to a large number of users. The essential advantage of P2P systems is that the system capacity scales up when more peers join, as peer upload capacity is utilized. However, providing satisfactory streaming services over P2P networks is challenging because of their inherent *instability* and *unreliability* and the limited adaptability of traditional video coding techniques. On one hand, different from dedicated servers, users may not have enough bandwidth to serve other users as most user connections are asymmetric in their upload and download capacity, and they are heterogeneous in terms of bandwidth and preferences. In addition, users can join and leave the system at any time as there are no guarantees on their contribution to the system. On the other hand, although traditional video coding techniques are efficient in terms of resource consumption, compression ratio, and coding and decoding speed, they do not support scalable modes *efficiently* as such modes come along with high computation cost. Consequently, in traditional P2P streaming systems, the bit rate (the video quality) of media streams is determined based on the capacities of the low-end users, *i.e.* the lowest common denominator, to make sure that most of their users can perceive acceptable quality. This causes two critical limitations of the current P2P streaming systems. First, users perceive the same quality regardless of their bandwidth capacity, *i.e.*, *no differentiated QoS*. Second, with the current best-effort Internet and peer dynamics, the streaming quality at each peer is easily impaired, *i.e.*, *no continuous playback*.

Recently, multiple layer codec research has become more refined, as SVC (the scalable extension of the H.264/AVC standard) has been standardized with a bit rate overhead of around 10% and an indistinguishable visual quality compared to the state of the art single layer codec. The hypothesis of this research work is that the adaptable coding technique can bring significant benefits to P2P streaming as it enables adaptability in P2P streaming. In addition, to improve the robustness of the system to network fluctuations and peer dynamics, network coding and social networking are also applied. The overall goal of this research is to achieve *adaptive* and *robust* P2P streaming services, which are believed to be the next generation of P2P streaming on the Internet. Several major contributions are presented in this dissertation. *First*, to use SVC in P2P streaming, a segmentation method to segment SVC streams into scalable units is proposed such that they can be delivered adaptively by the P2P paradigm. The method is demonstrated to be able to preserve the scalability features of a stream, *i.e.*, adaptation can be applied on

segments and the re-generated stream at each peer is a valid stream. *Second*, a novel and complete adaptive P2P streaming protocol, named *Chameleon*, is presented. Chameleon uses the segmentation method to use SVC and combine it with network coding in P2P streaming to achieve high performance streaming. The core of Chameleon is studied, including neighbor selection, quality adaptation, receiver-driven peer coordination, and sender selection, with different design options. Experiments on Chameleon reveal that overlay construction is important to system performance, and traditional gossip-based protocols are not good enough for layered P2P streaming. Therefore, *third*, a SCAMP-based neighbor selection protocol and a peer sampling-based membership management protocol for layered P2P streaming are proposed. These gossip-based protocols are *quality*- and *context-aware* as they form robust and adaptable overlays for layered P2P streaming so that high capacity peers have a higher priority to be located at good positions in the overlay, *e.g.* closer to the server, and peers with similar capacity are connected to each other to better utilize resources. *Fourth*, to better deal with peer dynamics, *Stir*, a social-based P2P streaming system, is suggested. In Stir, the novel idea of spontaneous social networking is introduced. Stir users who join the same streaming session can make friends and communicate with each other by cheap yet efficient communication means, *e.g.*, instant messaging and Twitter-like commenting. Such friendship networks are exploited directly by the underlying social-based P2P streaming protocol. The tight integration between the high level social networking of users and the low level overlay of peers is demonstrated to be beneficial in dealing with high churn rates and providing personalized streaming services. *Finally*, as the approaches are about different aspects of adaptive and robust P2P streaming, to complete the picture, *Chameleon++*, which combines Chameleon and Stir, is presented. The design and the evaluation of Chameleon++ demonstrate the feasibility and the benefits of the approaches, and the consistency of the study.

Acknowledgements

First and foremost, I would like to deeply thank my advisor, Professor Frank Eliassen, for his invaluable supports and encouragements thorough my research towards this thesis. He has offered me useful and helpful advice not only on research, but also on research ethic. Academic thinking and writing are two valuable things that I have learned a lot from him.

I would like to send my great appreciation to my supervisor, Dr. Michael Welzl, for his patience and willingness on discussions and fixing my English. Technical discussions with him always bring up new research ideas, better understandings, and improvement in interpersonal skills.

I would like to specially thank Professor Baochun Li, for his research guidances and collaboration. The 3-month research period at iQua, his research group, is the most important period of my research. Under his supervision during the period, I learned how to do practically experimental research, how to develop research ideas, and characteristics of high quality research. These experiences are always relevant for my research later on.

I am grateful for the supports and assistances from my colleagues and friends: Hai Ngoc Pham, Amirhosein Taherkordi, Lucas Luiz Provensi, and Narasimha Raghavan Veeraragavan in the Network and Distributed System research group at IFI; Di Niu, Chen Feng, Henry Hong Xu, and Zimu Liu in iQua research group at EECG; and Khai The Vuong, Khoa Huynh Dat Vu, and Tuan Vu Cao in VnOSLO. In addition to research discussions, pleasant and memorable time that we have shared is of great help for my stressful and not-always-easy research life. Thank you very much for being there when I need someone to talk to.

Last but not least, I reserve my heartfelt thanks to my family for their unconditional emotional support during the past 3 years. I am grateful to my parents and my younger sister for their encouragements; and to my wife, Nga, for her love, dedication and belief in my graduate studies, without which all that I have achieved was not possible. I love you all!

Contents

1 Introduction **1**

1.1 Motivation 1

1.1.1 Limitations of Traditional P2P Streaming Systems 1

1.1.2 The Case for Scalable Video Coding 3

1.1.3 The Case for Network Coding 4

1.1.4 The Case for Social Networking 4

1.2 Research Questions and Problem Statement 5

1.3 Research Methodology 7

1.3.1 Step 1: Formulating a Research Problem 8

1.3.2 Step 2: Approaching to Problems 8

1.3.3 Step 3: Evaluating Solutions 9

1.4 Contributions 10

1.5 Outline of the Dissertation 11

2 Background **13**

2.1 Peer-to-Peer 13

2.1.1 Definition 13

2.1.2 Research Challenges in P2P Systems and Applications 15

2.2 P2P Streaming 17

2.3 Network Coding 18

2.4 Scalable Video Coding 19

2.4.1 Scalability Modes 20

2.4.2 SVC Structure 21

2.4.3 Decoding Dependency 21
2.4.4 BitStream Switching 22
2.5 Social Networking 23

3 Literature Review 25
3.1 Taxonomy of P2P Streaming 25
3.2 Traditional P2P Streaming Systems 27
3.3 Adaptive P2P Streaming 28
3.4 Overlay Construction in P2P Streaming Systems 30
3.5 Network Coding in P2P Systems 33
3.6 Social Networking in P2P Systems 35

4 Coding Approaches: Network Coding Meets SVC in P2P Streaming 37
4.1 Motivation for Using Network Coding 37
4.2 Proposed Segmentation Method 39
4.3 SVC with Random Network Coding 40
4.4 Chameleon: Adaptive P2P Streaming with Network Coding 41
4.4.1 System Overview 41
4.4.2 Design Space of Key Components 42
4.5 Performance Evaluation 50
4.5.1 Scalability 51
4.5.2 Coping with Peer Dynamics 52
4.6 Summary 52

5 Overlay Construction Approaches 55
5.1 SCAMP-based Overlay Construction 55
5.1.1 Problem Identification 55
5.1.2 A Quality- and Context-Aware Neighbor Selection Method 57
5.1.3 Evaluation 61
5.2 Peer Sampling Based Overlay Construction 63

5.2.1 Protocol Design 63
5.2.2 Evaluation 65
5.3 Summary 69

6 Social Networking Approaches 71
6.1 The Case for Spontaneous Social Networking 71
6.2 Stir: Schematic and Architectural Design 73
6.3 A Social-based P2P Streaming Protocol 74
6.3.1 Relying on Friendship or Bandwidth: a Tradeoff 75
6.3.2 Partner Manager 76
6.3.3 Packet Scheduler 76
6.4 Stir: Experimental Results 77
6.4.1 Data Preparation and Assumptions 79
6.4.2 Comparison with Existing Work 81
6.4.3 Insights of Stir 83
6.5 Social Traffic Costs 87
6.6 Summary 88

7 *Chameleon*++: Putting It All Together 89
7.1 The Design of Chameleon++ 89
7.2 Evaluation 91
7.2.1 Quality-aware and Social-based Partner Selection 92
7.2.2 Chameleon++ vs. Chameleon 93
7.2.3 Skip Rate - Quality Satisfaction Tradeoff: Is it worth to combine Chameleon and Stir? 95
7.3 Summary 96

8 Concluding Remarks 97
8.1 Self-Assessment 97
8.2 Technical Contributions 98
8.2.1 Chameleon: Adaptive P2P Streaming with NC and SVC 98

8.2.2 Quality-aware Membership Management for Layered P2P Streaming . 99
8.2.3 Stir: Spontaneous Social P2P Streaming 99
8.3 Future Directions 100

A The Simulator **103**

A.1 Available Simulators 103
A.2 Packet-level Discrete-event Simulator 105
A.3 P2P Protocols 106
A.4 Max-min Fair Rate Allocation in Simulation 108
A.4.1 Traditional Implementation 108
A.4.2 A More Efficient and Practical Implementation 110
A.4.3 Performance Evaluation 112
A.5 Summary 114

List of Figures

1.1 Friendships among members of the “Macbook Pro” group. 7

2.1 Architectural Comparison among Client-Server, Hybrid, and P2P Model. . . . 14

2.2 Reasons against P2P [1]. 16

2.3 Research Focus on P2P (2003-2007) [1]. 16

2.4 Research Focus on P2P (2007-2010) [1]. 16

2.5 Video Chunk Delivery in P2P Streaming. 17

2.6 Traditional Relay Vs. Network Coding. 19

2.7 An example of the SVC structure. (a) An AU consisting of four LRs. (b) A GOP consisting of eight pictures (AUs) and coded with hierarchical B-pictures. The symbols *Tk* specify the temporal layers with *k* representing the corresponding T_ID. The numbers below specify the coding order. (c) A coded video sequence. 21

2.8 The graph representation of the example social network. 24

3.1 Overlay-based taxonomy of P2P streaming systems. 26

3.2 Adaptability-based taxonomy of P2P streaming systems. 27

4.1 The store order of the entities in the video file 39

4.2 An example of the segmentation method where the stream has three quality levels and is divided into segments of two GOPs. The symbols *QL k* specify $Q\_ID = k$. 40

4.3 An example of the combination of network coding and SVC. Packet 1, 2, and 3 are divided into n, m, and k blocks, respectively. Network coding with different numbers of unknowns (n, m, and k) is used for different quality levels. 41

4.4 Architecture of Chameleon with key components. 42

4.5 The effect of the neighbor selection methods on Chameleon. 44

4.6 The playback buffer in Chameleon: The dark shade indicates the receiving status of each segment. 45
4.7 The effect of the quality adaptation parameters on Chameleon. 46
4.8 An example of the playback graph of a typical peer. 47
4.9 The performance of Chameleon with different sender selection methods. 49
4.10 The performance of Chameleon and FABALAM in different network sizes. . . 51
4.11 The effects of peer dynamics on Chameleon. 52

5.1 The performance of Chameleon in the two cases. 57
5.2 The system performance with the proposed protocol in Case B 61
5.3 Performance of Chameleon with different network sizes 62
5.4 The arrangement of neighbors in the neighbor list of peer P. C_P is the class identifier of P 65
5.5 Overlay evolves from a random topology. 67
5.6 Overlay evolves from a ring topology. 68
5.7 Overlay evolves with peers join. 68
5.8 Number of clusters generated when peers leave the system 69

6.1 Initial steps in the Stir schematic design. After a user logs in, a list of friends may be established spontaneously. 73
6.2 The architectural design of Stir, with an emphasis on the tight integration between streaming quality and spontaneous social network relationships. 74
6.3 Friendships among peers 80
6.4 Peer Dynamic Scenarios 80
6.5 Stir minimizes the impact of peer churn 82
6.6 Skip rates with different sizes of the neighbor list. The numbers in parentheses denote the size of the neighbor list in CoolStreaming and NCStream. 83
6.7 The effect of α and β in partner selection. 84
6.8 In Stir: The more famous you are, the higher the quality you receive. 86

7.1 The architecture of Chameleon++. Components in light grey color are from Stir. The dark grey color of Partner Manager is to mark that this component is the integration 'point' of Chameleon and Stir 90

7.2 The role of the quality-aware selection in Chameleon++ 93

7.3 Chameleon++ Vs. Chameleon on different neighbor list sizes. The numbers in parentheses denote the size of the neighbor list in Chameleon 94

7.4 Chameleon++ Vs. Chameleon on coping with peer dynamics. 95

A.1 The components of the simulator with their main functions 105

A.2 An example of modeling the network to undirected graph. The directed edges on the left graph represent for the connections from the upload nodes to the download nodes. A physical node *i* is modeled as two distinct nodes *iU* and *iD* for its uplinks and downlinks in the right graph. The number above a node is its bandwidth (displayed as upload/download for the nodes in the left graph). . . . 110

A.3 An example of the computation of the proposed algorithm. Dashed links are saturated connections. 113

A.4 The performance of the algorithms with different network sizes 114

A.5 The performance of the algorithms with different values of AC 115

List of Tables

2.1 Differences between live P2P streaming and on-demand P2P streaming. 18
2.2 An example about the scalability structure of an SVC stream. 22
2.3 The matrix representation of the example social network. 23

4.1 Meanings of the two bits used in buffer maps. 42
4.2 Peer clustering in Chameleon. 45
4.3 Main configuration parameters used in the simulation. 50

5.1 Topology 62

A.1 Events used in our simulation 108

Chapter 1

Introduction

This first chapter discusses motivations of the work: *why is adaptive and robust P2P streaming necessary to be studied and achieved?* Next, technical problems to be tackled are described at a high level. Then, the research methodology that is used is presented. This chapter also lists main contributions and outlines the remainder of the dissertation.

1.1 Motivation

1.1.1 Limitations of Traditional P2P Streaming Systems

The rapid development of computer and network technologies has changed the way in which people learn, communicate, and entertain themselves, as it enables a wide range of multimedia streaming applications from video-on-demand streaming to highly interactive video conferencing. The key attractions of such emerging multimedia services are space-decoupling, and rich and diversified services. For example, tele-education allows people to study without the need of participating in physical classes. Thanks to video conferencing technologies, virtual classes in which teachers and students can communicate with each other in multiple directional ways are possible even with critical timing constraints for real time interaction. Another example is that nowadays users can enjoy a large number of video and TV channels on the Internet using IPTV (Internet Protocol Television) services. Compared to traditional TV, IPTV provides more interactive programming, *e.g.*, viewers can customize their channel lists. The "Semiannual IPTV Global Forcast Report – IPTV Global Forecast – 2010-2014", published by MRG (Multimedia Research Group), Inc. in June 2010, indicates that the number of global IPTV subscribers will grow from 41.2 million at the end of 2010 to 101.7 million in 2014, a compound annual growth rate of 25.3% [2].

Peer-to-Peer (P2P) technologies have emerged as a powerful and popular paradigm for bringing multimedia services to a large number of users as they overcome the scalability problem of the traditional client-server architectures. The essential advantage of P2P systems is that the system capacity scales up when more peers join, as peer upload capacity is utilized. There are nowadays many popular video streaming systems, *e.g.*, PPLive [3], CoolStreaming [4], TVAnts [5], UUSee [6]. They are serving hundred thousands of users simultaneously, thanks to their P2P-based architecture. However, providing satisfactory streaming services over P2P networks is challenging because of their inherent instability and unreliability and the limited adaptability of traditional video coding techniques. On one hand, different from dedicated servers, users may not have enough bandwidth to serve other users, as most user connections are asymmetric in their upload and download capacity, and they are heterogeneous in terms of bandwidth and preferences. In addition, users can join and leave the system at any time, as there are no guarantees on their contribution to the system. On the other hand, although the traditional video coding techniques are efficient in terms of resource consumption, compression ratio, and coding/decoding speed, they do not support scalable modes efficiently as their encoded streams are united and only playable if the whole stream is received correctly. Consequently, in traditional P2P streaming systems, the bit rate (the video quality) of media streams is determined based on the capacities of the low-end users, the lowest common denominator, to make sure that most of their users can perceive acceptable quality. This causes two critical limitations of the current P2P streaming systems:

- ▷ Users have to receive the same stream regardless of their bandwidth, *i.e.*, high capacity users perceive the same low quality as average users (no differentiated QoS).
- ▷ With the current best-effort Internet and the peer dynamics, the streaming quality at each peer is easily impaired, *i.e.*, when the available bandwidth at a peer drops below the streaming rate, it may suffer playback skips (no continuous playback).

From the above discussion, it is very much required for a large-scale P2P streaming system to improve user satisfaction by achieving adaptive and robust streaming services. In this context, adaptation means twofold. *First*, the video quality a user receives depends on her bandwidth, *i.e.*, high capacity users enjoy high quality video while low capacity users watch low quality video in the same streaming session. *Second*, when the available bandwidth of a user changes her video quality may change accordingly, *e.g.*, when the bandwidth drops the image quality can be reduced to maintain continuous playbacks. Meanwhile, robustness is defined in this work as the ability of maintaining the current quality under a certain level of peer dynamics, *i.e.*, the current video quality of peers is maintained even when a certain number of other peers join or leave the session. In a nutshell, this research work is motivated by the fact that there currently is

a big gap between the streaming quality that is expected by users and the streaming quality that is provided by existing P2P streaming systems. It investigates the feasibility of offering such adaptive and robust streaming services with the current technologies.

1.1.2 The Case for Scalable Video Coding

There have been many video coding standards for different kinds of video transmission and storage. Traditional digital video transmission and storage systems are based on H.222.0|MPGE-2 systems for broadcasting services over satellite, cable, and terrestrial transmission channels, and for DVD storage, or on H.320 for conversational video conferencing services [7]. These (and past) coding schemes define a fixed spatio-temporal format for the video signal (SDTV or HDTV or CIF for H.320 video telephone). Consequently, their application behavior in such systems typically falls into one of the two categories: it works or it does not work. If the available bandwidth is greater than the video bit rate, good quality is perceived, otherwise playback skips (in live streaming) or playback stops (in video-on-demand streaming) occur. Such behavior is very inappropriate when videos are transferred over the current best-effort Internet, as no bandwidth guarantees can be made.

Scalable video coding (SVC) is a highly attractive solution to the adaptability problem. The term "scalability" refers to the removal of parts of the video bit stream in order to adapt it to the various needs or preferences of end users as well as to varying terminal capabilities or network conditions. SVC has been an active research and standardization area for at least 20 years. The prior international video coding standards H.262 MPEG-2 Video, H.263, and MPEG-4 Visual already include several tools by which the most important scalability modes can be supported. However, the scalable profiles of those standards have rarely been used because the spatial and quality scalability features came with a significant loss in coding efficiency as well as a large increase in decoder complexity as compared to the corresponding non-scalable profiles. Only until the scalable extension of the H.264/AVC standard [8], different scalability modes can be supported with high coding efficiency. It should be noted that the term SVC is used interchangeably for both the concept of scalable video coding in general and for the particular new extension of the H.264/AVC standard. The following advantages of the scalable extension leads to its standardization in July 2007 [9].

- ▷ Similar coding efficiency compared to single-layer coding for each subset of the scalable bit stream.
- ▷ Little increase in decoding complexity compared to single-layer decoding that scales with the decoded spatio-temporal resolution and bit rate. A bit rate overhead of around 10%

and an indistinguishable visual quality, compared to the state of the art H.264/AVC, have been demonstrated [9, 10].

- ▷ Support of different scalability modes: temporal, spatial, quality scalability, and their combinations.
- ▷ Support of a backward compatible base layer (H.264/AVC).
- ▷ Support of simple bit stream adaptations after encoding.

There are also other alternatives to provide adaptability when videos are transferred over the Internet, *e.g.*, multiple description coding and multiple version streaming. However, experimental studies have demonstrated that these approaches come along with the cost of a significant increase in bit rate [11, 9], and references therein. Altogether, there is reason to believe that SVC will become the dominant coding technique used for transmitting videos over the Internet in the near future.

1.1.3 The Case for Network Coding

Network coding (NC) has been shown to be beneficial in P2P streaming as it offers better throughput and improves system performance [12, 13]. The key feature of network coding is that it makes all pieces of data equally important, and every coded packet is innovative to receivers with high probability [14]. This feature maximizes the potential of peer collaboration (referred to as *perfect collaboration* in [13]) as a receiver does not need to coordinate senders to avoid duplicated packets. As long as the receiver receives enough linearly independent coded blocks, she can decode to the original data. As being presented in the "Research Questions and Problem Statement", the scalability features of SVC limit the collaboration potential of peers when it is used in P2P streaming. Therefore, the objective of using NC in the combination with SVC is to exploit the perfect collaboration feature of NC to mitigate the limitation caused by the layering features of SVC to make the use of SVC feasible and beneficial.

1.1.4 The Case for Social Networking

In traditional P2P systems, users are often anonymous as the systems do not keep any user profiles for operation. On one hand, such anonymity simplifies the use of services as users only need to run a client program and will be served. On the other hand, the anonymity causes one of the most challenging problems in P2P systems: the free-riding problem [15]. Free-riders are those who join a P2P session and receive services without contributing their resource to the

community. This selfish behavior violates the P2P principle and reduces the overall resource which is based on the contribution of every peers.

Recently, the popularization of new social network sites that allow individuals to construct personal profiles, connect to people, and keep up with friends has shown that users are indeed interested in sharing their common interests on networked applications. For example, at the time of writing, Facebook [16] is the second most-trafficked website in the world [17]. It currently has 400 million active users who spend 500 billion minutes per month to interact with over 160 million objects (pages, groups, and events) [18].

Applications of social networking to P2P networking have emerged with the objective of improving P2P system performance. The idea is that connections among friends are more reliable than those among strangers. Some studies have shown advantages of social-based P2P systems in file sharing in which the systems import social contacts from existing social networking sites and establish connections among peers based on their social relationships [19, 20, 21]. Such connections are more reliable and durable. Since users now have to be responsible for their behavior as their identity is revealed, the above free-riding problem is minimized.

In addition to mitigating the free-riding problem, the use of social networking in P2P streaming has another important and unique motivation. One very challenging and inherent problem in P2P streaming is high churn rates, *i.e.*, peers stay in a streaming session for a short period of time. When a peer leaves, others who are receiving packets from it are negatively affected. There has been a substantial amount of research on dealing with peer churn [22, 13, 23]. Although the works have shown improvements in system performance, the impact of peer churn can not be minimized as its origin has not been considered. The arrivals and departures of users depend on their interests in the session, *e.g.*, users leave quickly because they do not like the content. A hypothesis here is that knowledge about user interests is critical in dealing with peer churn, and such knowledge can be collected if users are supported with social activities, *e.g.* making friends and communicating with each others, *inside* of a P2P streaming system. Therefore, the use of social networking in P2P streaming, if possible, will bring significant benefits in dealing with the free-riding and the high churn rate problems, and so improve the robustness of the system.

1.2 Research Questions and Problem Statement

The main research question is: *How to make a large-scale P2P streaming system adaptable and robust so that it can offer the best possible experience to heterogeneous users under highly dynamic network conditions while maintaining its efficiency and scalability?* To address this question, many problems need to be considered.

The background of the problem is in using SVC with P2P streaming due to the P2P-based data delivery mechanism and the layering features of SVC. In P2P systems, a peer not only receives data from other peers but also sends received data to others. Consequently, the bandwidth and data availability of each receiving peer are constrained and heterogeneous, which further limits the bandwidth and data availability of its downstream peers when it acts as a sender. When SVC is in use, the heterogeneity among peers is even higher because peers do not receive the same content, *i.e.*, different peers may receive different numbers of layers based on their bandwidth. For example, a peer which can only receive two layers due to its bandwidth or preferences cannot provide the third (or higher) layers to others which might want and be able to receive more than two. If with SVC, the collaboration potential of peers is so limited that the system cannot utilize the upload bandwidth of peers, the benefit of SVC is minimal. Therefore, it is necessary to first study: *Can SVC be used with the P2P paradigm to provide the best possible experience to heterogeneous users?* If the answer is 'Yes', the next step is to investigate further with the following questions:

- ▷ *What is the cost of using SVC?* It could be straightforward to understand that the layering features of SVC reduces the collaboration potential among peers as explained above. However, the cost of using SVC needs to be measured in terms of system performance to answer if the cost is affordable.

- ▷ If the effect of using SVC on the collaboration among peers is seriously negative, *Can the perfect collaboration feature of network coding help?* If 'Yes', *How can SVC efficiently be combined with NC?* and *What are benefits of the combination in terms of system performance?*

With respect to the application of social networking in P2P streaming, there are two problems to be considered: the *formation* of social networks and the *exploitation* of the social networks. In existing social-based P2P file sharing systems, the establishment of connections among peers is based on social relationships among users, which are not formed in the context of a peer-to-peer session but, *e.g.*, imported from other social networks. Because friends in such a separate social network do not always have similar interests, they may not necessarily join the same P2P session. Consequently, a user may not acquire enough qualified connections and suffers degraded quality as a result. As an example, in Flickr [24], each user has a friend (contact) list. Some user can create a group on a particular topic, and others can join the group to share their interests in photos of group members. Friendship among members of Flickr groups is investigated: *How many friends of a member also join the group?* or *Are members of a group also friends of each other?* For 20 groups with different sizes and topics having been considered, the answer is that in most of the groups, members have *no*, or very few, friends in their group.

Figure 1.1 demonstrates the observation with the friendship network of members of "Macbook Pro" group.

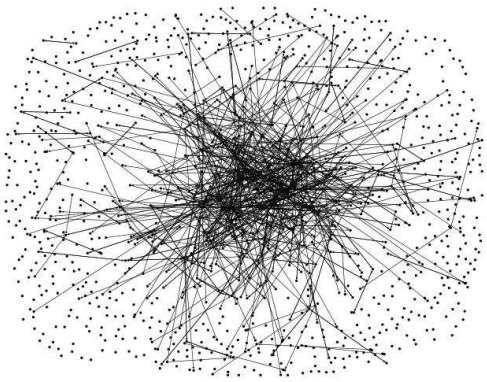


Figure 1.1: Friendships among members of the "Macbook Pro" group.

Therefore, the first problem to be tackled is: *How to form social networks which have a very tight integration with the P2P overlay,* i.e., *the social relationships should be closely related to the context of the P2P application?* If such social networks can be formed efficiently, the next question is: *How can such social networks be exploited by the underlying P2P streaming protocol?*

Once all the above questions are answered, it would be convincing that an adaptive and robust P2P streaming system with the current cutting edge technologies is feasible. Seeking solutions for the questions is the main task of this research work, and what has been studied is presented in this dissertation.

1.3 Research Methodology

The research methodology that was used in this research is tailored from the general research methodology, presented in [25]. To tackle the research questions, mentioned in Section 1.2, different research methods for different tasks were identified and justified during the research period. The research has been carried out with the following steps. In each step, suitable methods were selected and justified. It should be noted that some steps were repeated, as results were not as expected or new problems were identified.

1.3.1 Step 1: Formulating a Research Problem

The purpose of formulating a research problem is to understand it deeply: *Is it worth to spend effort on?* and *How can we evaluate the solutions?* The following methods are applied:

- ▷ Literature study: related work in P2P streaming, layered P2P streaming, applications of SVC, NC, and Social Network in P2P systems is studied in detail. State-of-the-art studies are noted with their key features, both advantages and disadvantages.
- ▷ Problem Formulation: After the literature review, we know which problems have been fully/partially tackled. We now have to define in detail technical problems for each main research questions.
- ▷ Hypothesis construction: *what are the hypotheses for the work?* Identifying the hypotheses helps evaluate the solutions and compare with state-of-the-art work.

1.3.2 Step 2: Approaching to Problems

For each research question, possible solutions are found and listed. Preliminary analysis and evaluation are carried out. Since most of the problems are in the form of how to make things work and work efficiently, approaches to the problems are about designing systems/protocols with different design options, *e.g.*, designing an adaptive and robust P2P streaming protocol using SVC and NC. Following the discipline of computing [26], the main paradigm used in this work is the *design* paradigm.

The design paradigm is one of the three paradigms (the other two are *theory* and *abstraction* paradigm) approved by the ACM Education Board as an intellectual framework for the discipline of computing and a basic for computing curricula [26]. Theory is concerned with the ability to describe and prove relationships among objects. Abstraction is concerned with the ability to use those relationships to make predictions. Design is concerned with the ability to implement specific instances of those relationships and use them to perform useful actions. The design paradigm includes the four following steps in the construction of a system to solve a given problem:

- State requirements: the required features of the system from the user perspective are determined. Satisfying the requirements is the criteria to evaluate the system.
- State specifications: from the requirements, more technical details about the system are defined.
- Design and implement the system.

- Test the system: evaluate the performance and functionality of the system. The testing method is particularly presented in the step 3 of the methodology.

1.3.3 Step 3: Evaluating Solutions

Proposed systems need to be evaluated. A system design can be implemented and evaluated in a simulator (simulation), in an emulator (emulation), or in a real environment (prototype). On one hand, simulation offers simplest implementation, repeatable experiments, and quick result collection. However, not all aspects of the system are considered, there are always assumptions with simulation depending on what features of the system are considered. On the other hand, building a prototype and evaluating it in a real environment requires a lot of programming effort, and experiments may not be repeatable, especially for networked applications, as the real network changes quickly and totally distributed, *e.g.*, we may not have permissions to change environment parameters. However, once the prototype is tested, its performance is guaranteed as most of the aspects of the system are considered in the real world. The simplicity and reality of emulation is in between those of simulation and real world testing as the system is evaluated in a well-controlled real environment.

With the above tradeoffs of the different evaluation methods, simulation is chosen as the main method to evaluate proposed solutions for the following reasons:

▷ It is highly risky and requires a lot of programming effort to implement and test a large-scale P2P system in the real world. Even if we could develop a complete system in a short period of time, it is impossible to attract up to thousands of users for the evaluation. In addition, experimental results in this setting may not be analyzed and diagnosed by tuning parameters. It is also difficult for other researchers to independently reproduce and verify the results. Therefore, with the period of three years for one PhD student, targeting to a real prototype is not feasible.

▷ One reason for using emulation is that some existing emulators and test beds allow to evaluate system performance with taking real underlying network communication into account. However, implementing a system in emulation is also time consuming with all network programming problems because the emulated implementation should be very close to the real implementation. More importantly, the focus of this work is more about the feasibility of the design at the application levels. Taking all actual network transmissions is unnecessary.

▷ It is firstly important to understand how a system behaves in large-scale scenarios with its design before actual deployment. In the case of new large-scale P2P streaming protocols, collective behaviors of the whole system may be more important than the network

performance of a particular peer. Therefore, with reasonable assumptions, it is believed that simulation is convincing enough and acceptable in this work. In addition, to compare with state-of-the-art studies, some of which are also evaluated in simulation, simulation allows easily extracting and evaluating key features of the system.

An important issue related to simulation is the input data for experiments. While there are reasons for avoiding real network transmission, to have more convincing simulation results, real input data should be used whenever possible, *e.g.*, real peer dynamic traces, actual SVC streams, and actual social network data. This makes the study different from those which use synthetic input data and brings it closer to the reality.

1.4 Contributions

The main technical contributions are summarized and and referred to the corresponding publications in this section.

- ▷ C1: A segmentation method to divide SVC streams into scalable units to be delivered in P2P networks. The method is demonstrated to preserve the scalability of the stream, *i.e.*, adaptation can operate on segments and the re-generated stream at each peer is a valid stream.
- ▷ C2: A novel and complete adaptive P2P streaming protocol, named Chameleon. Chameleon uses the segmentation method and combines SVC with network coding to achieve high performance streaming. The core of Chameleon is studied, including neighbor selection, quality adaptation, receiver-driven peer coordination, and sender selection with different design options.
- ▷ C3: A SCAMP-based neighbor selection protocol for layered P2P streaming.
- ▷ C4: A peer sampling-based membership management protocol for layered p2P streaming.
- ▷ C5: A novel social-based P2P streaming system, named Stir. Stir introduces a novel idea of spontaneous social networking, in which users who join the same streaming session can make friends and communicate with each other by cheap yet efficient communication means: instant messaging and Twitter-like commenting. Such friendship networks are exploited directly by the underlying social-based P2P streaming protocol. The tight integration between the high level social networking of users and the low level overlay of peers brings significant benefits in dealing with a high churn rate and providing personalized streaming services.

▷ C6: Chameleon++. The design of Chameleon and that of Stir are combined in Chameleon++. Although the design of Chameleon++ is mainly based on that Chameleon and Stir, it shows that the approaches are complementary and can be combined to yield an even better system. Chameleon++ offers very low skip rates like Stir, and high quality satisfaction like Chameleon. SVC, NC, and social networking are all used efficiently to achieve adaptive and robust P2P streaming.

C1 and C2 are presented in [27], which was published in the $29th$ IEEE International Conference on Computer Communications (IEEE INFOCOM 2010). This piece of work was mostly done during my tenure at the iQua research group of Prof. Baochun Li, at the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada. I proposed the segmentation method and designed Chameleon, Prof. Li helped with ideas on doing simulations. Both Prof. Li and Prof. Eliassen gave great help in writing up the paper. C3 and C4 were presented at the International Communications Conference (IEEE ICC 2010) [28], and the IEEE Consumer Communications and Networking Conference (IEEE CCNC 2011) [29], respectively. In both C3 and C4, I proposed and evaluated the protocols. The other authors contributed in writing. The idea of using spontaneous social networking in P2P streaming was initially proposed by Prof. Li. I designed the schematic architecture of Stir with his help. I proposed and evaluated the P2P streaming protocol of Stir. Prof. Li and Dr. Welzl gave useful comments on finding input data for evaluation. At the time of this writing, C5 was submitted to the 14th IEEE Global Internet Symposium. Prof. Eliassen, Prof. Li, and Dr. Welzl helped in writing up the submission.

1.5 Outline of the Dissertation

The remainder of this dissertation presents a thorough examination of the work. To make this dissertation self-contained, Chapter 2 briefly introduces P2P streaming and gives an overview and important features of SVC, NC, and social networking. Readers who already know about these technologies can ignore this chapter, otherwise understanding the background will help understand the proposed solutions presented later on. Chapter 3 discusses related work including traditional P2P streaming with both mesh-based and tree-based overlays, layered P2P streaming with different layered coding techniques, applications of SVC, NC, and social networking in P2P systems in general and in P2P streaming in particular. Chapter 4, 5, 6, and 7 describe in detail the proposed solutions and evaluation. They are reported in the order of the time at which the work was done. *First*, Chapter 4 is about Chameleon. *Second*, Chapter 5 concentrates on overlay construction for layered P2P streaming with the two new overlay construction schemes. *Third*, Chapter 6 presents Stir with its design and evaluation. *Finally*,

Chapter 7 puts it all together with the design and evaluation of Chameleon++. This dissertation is concluded in Chapter 8 with revising the goals of the work and suggesting directions for future research. *Finally*, Appendix A describes in detail the simulator that was used.

Chapter 2

Background

This chapter gives an overview of technologies that are used in this work. It also explains some terminologies used through out this dissertation. Readers who are familiar with P2P streaming, network coding, scalable video coding, and social networks can skip this chapter.

2.1 Peer-to-Peer

The following introduction to P2P is based on [1].

2.1.1 Definition

Oram [30] gives a basic definition of the term “Peer-to-Peer”:
A Peer-to-Peer system is a self-organizing system of equal, autonomous entities (peers) which aims for the shared usage of distributed resources in a networked environment avoiding central services.

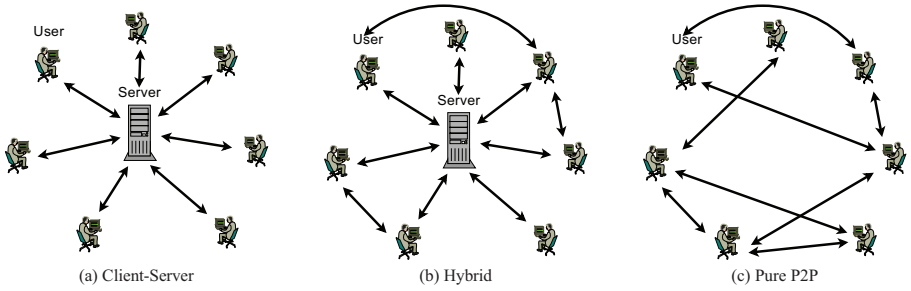
Decentralized resource usage includes:

- ▷ Resources of interest, *e.g.* bandwidth, are used in a manner as equally distributed as possible and are located at the edges of the network, close to the peers.
- ▷ Within a set of peers, each utilizes the resources provided by other peers. The most prominent examples for such resources are storage and processing capacity.
- ▷ Peers are interconnected through a network and in most cases distributed globally.
- ▷ Peers’ Internet address typically changes because they are dynamically assigned new Internet addresses every time they connect to the network (transient connectivity). Often, they may be disconnected or shut down over longer periods of time.

Decentralized self-organization includes:

- ▷ In order to utilize shared resources, peers directly interact with each other. In general, this interaction is achieved without any central control or coordination. This represents one of the main properties of P2P systems which is markedly different from client-server systems: while the the latter rely on centralized coordination through a server as a structural paradigm, P2P systems establish a cooperation between equal partners. This departure from a centralized infrastructure most importantly avoids bottlenecks but is concomitant with the reduced availability of end-systems compared to client-server solutions.
- ▷ Peers directly access and exchange the shared resources they utilize without a centralized service. Thus, P2P systems represent a fundamental decentralization of control mechanisms, *i.e.*, peers can act both as clients and servers. This is radically different from traditional systems with asymmetric functionality.
- ▷ Each peer is fully autonomous regarding its respective resources.
- ▷ Ideally, resources can be located without any central entity. However, performance issues may lead to centralized elements being part of a complete P2P system, *e.g.* for efficiently locating resources. Such systems are commonly called hybrid P2P systems.

Figure 2.1 shows the architecture of a client-server model, P2P, and hybrid model to point out the key difference between them: there is one (or more) central component in the client-server and the hybrid model, and no central services in pure P2P architecture.



(a) Client-Server (b) Hybrid (c) Pure P2P

Figure 2.1: Architectural Comparison among Client-Server, Hybrid, and P2P Model.

There are two kinds of P2P systems: *structured* and *unstructured*. Unstructured P2P systems do not rely on a specific infrastructure offering transport services. They form unstructured overlays focusing on content allocation and distribution based on TCP or HTTP connections. Although unstructured overlays are flexible as peer dynamic do not affect the 'structure' of the overlays, they are not efficient in locating data in the network. On the other hand, structured P2P systems

do have well-structured overlays for efficient look-up services, *e.g.*, DHT (Distributed Hash Table). The main disadvantage of structured overlays is that it may be expensive to maintain the overlay under peer dynamics, *i.e.*, there is always an overlay manager running to keep the overlay in form.

2.1.2 Research Challenges in P2P Systems and Applications

One of the main challenges of P2P systems lies in the decentralized self-organization of a distributed system and in achieving a high level of quality of service without the need for centralized services. It is central to a solution to this problem to efficiently look up and locate data items and manage them accordingly. Many aspects of P2P systems are based on this functionality. In contrast to centralized server applications, for which the location of data items is inherently known, decentralized systems store content in multiple, possibly distant, locations within the system. These distributed features lead to the following main reasons against P2P (more reasons are mentioned in Figure 2.2 with the time at which they were raised and discussed[1]).

- ▷ Intellectual Property and Digital Right Management: since any anonymous peer can upload data to the network for sharing with others, it is really challenging, even impossible, to deal with copyright issues of the shared data, *e.g.*, videos, musics, and books.
- ▷ Still Low Bandwidth End-nodes and Best-effort Service Insufficient for Most Applications: for delay tolerant data, P2P is possibly the best approach. However, beyond file-sharing, the suitability of the P2P paradigm to various types of applications is questioned, especially when the bandwidth of end-nodes is still low (*e.g.*, compared to the bandwidth requirements for some applications, *e.g.*, HD (High Definition) video streaming) and the current Interest infrastructure is best-effort.
- ▷ Lack of Trust: security in a distributed system is generally challenging. In P2P, it is even more difficult to guarantee a certain level of security due to its anonymity and high decentralization.

The above issues have attracted a lot of attention. Different research focuses have been reported on different aspects of P2P systems. Figure 2.3 and Figure 2.4 shows different challenges researchers have focused on[1].

[1]Figure 2.2, Figure 2.3, and Figure 2.4 are copied from Fig. 2.3, Fig. 2.5, and Fig. 2.4, respectively in [1] with the permission that is specified at http://www.springerpub.com/resources/Authors/Permissions.

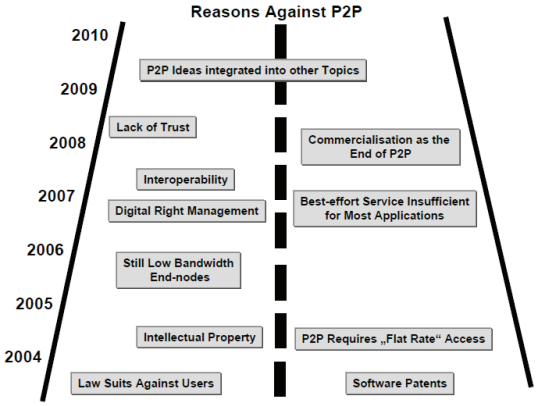


Figure 2.2: Reasons against P2P [1].

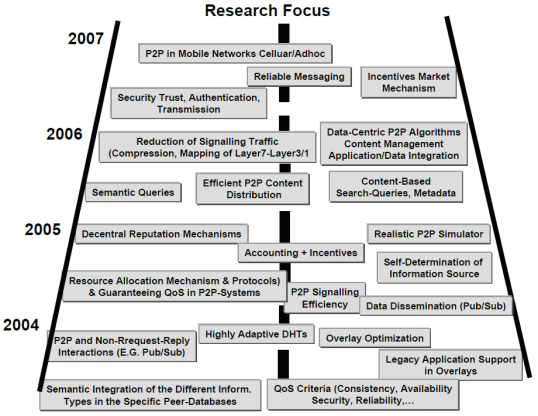


Figure 2.3: Research Focus on P2P (2003-2007) [1].

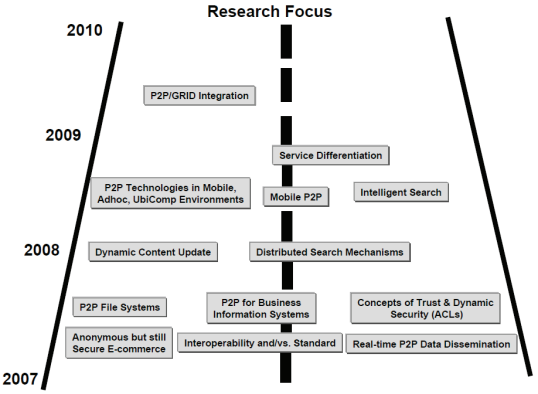


Figure 2.4: Research Focus on P2P (2007-2010) [1].

2.2 P2P Streaming

The idea of using the P2P paradigm for video streaming is very simple. The video stream is divided into chunks (a small unit of video data), and viewers act as relays, *i.e.*, when a peer receives a chunk, in addition to adding it to the playback buffer, the peer also uploads that chunk to other peers. Eventually, all peers receive enough chunks for playback. P2P streaming systems offer the same advantages as other P2P systems, *e.g.*, scalability and low load on servers, as peers contribute their bandwidth to deliver video chunks in the network. However, they face additional challenges since there is a strict timing constraint for the arrival of chunks to meet the playback deadline. This is particularly difficult since the peers are connected to the Internet by links which may have different capacity and reliability. Moreover, data delivery paths may simply disappear without prior notice, *e.g.*, when a peer leaves the system. Figure 2.5 illustrates the chunk delivery process among peers in a P2P streaming system. There are two kinds of video streaming applications: live P2P streaming and on-demand P2P streaming. They are different from each other in the way video churns are delivered.

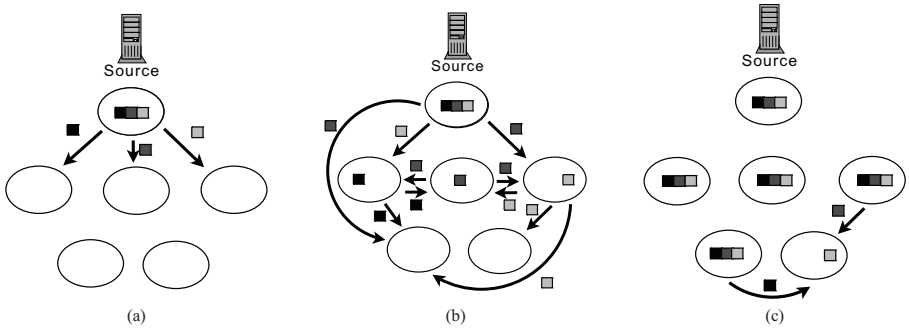


Figure 2.5: Video Chunk Delivery in P2P Streaming.

In live P2P streaming, the content provider streams a live event to peers. Since each peer expects to view the stream in real time, there are two critical criteria for the arrival of video chunks. First, chunks have to arrived at each peer on time, *i.e.*, before the playback deadline synchronized with the source. Second, time lag among peers should be small enough so that all peers receive the stream 'almost' at the same time. Owing to the synchronization with the source, late chunks are skipped and peers always expect to receive chunks of future playback segments (after the current playback point). This is the key different from on-demand streaming. In on-demand P2P streaming, users can watch a video whenever they want, *i.e.*, there is no synchronization among peers and with the source. Consequently, large time lags among peers are possible. Peers usually wait to receive every chunk. If the playback buffer is empty, the player is stopped and runs again when expected chunks arrive. Table 2.1 lists main differences between live P2P streaming and on-demand P2P streaming.

Table 2.1: Differences between live P2P streaming and on-demand P2P streaming.

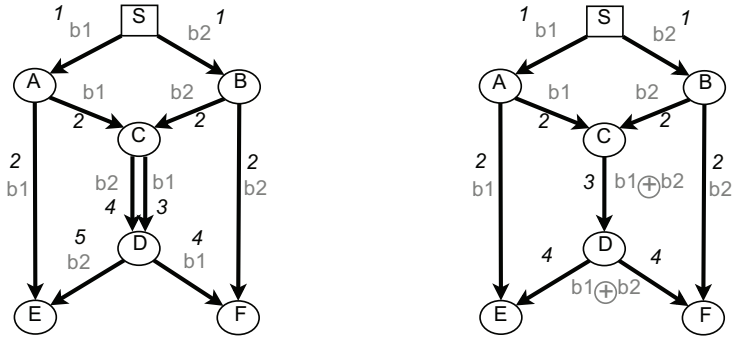
	Live	On-Demand
Content Availability	Online (the source does not have the whole content before streaming)	Offline (the whole content is already there)
Playback Synchronization	High	No
Late Chunks	Discarded	Played back
Buffer Size	Small (seconds)	Large (minutes)

2.3 Network Coding

NC has been originally proposed in information theory [14], and has since emerged as one of the most promising information theoretic approaches to improve performance in P2P networks. With NC, instead of simply forwarding data, nodes may recombine several input packets into one or several output packets. It has been demonstrated in both theory and practice that allowing coding at intermediate nodes can effectively improve the throughput of multicast communication sessions in directed acyclic graphs. To understand the mechanism and benefits of NC, we present a typical example of NC, the 'butterfly' network, described in [31].

Assume that we want to send two data bits b1 and b2 from the source S to both the nodes E and F in the network depicted by Figure 2.6(b), and the bandwidth of each link is 1 bit/second. The delivery process is described in Figure 2.6(a) for the traditional relay network with the number on a link denoting the time the transmission on that link occurs. As observed in Figure 2.6(a), it takes 5 seconds to send both b1 and b2 to E and F: in the 1st second, S sends b1 to A, and b2 to B; in the 2nd second A sends b1 to C and E while B sends b2 to C and F; and so on. Since the link between C and D is the bottleneck, only one bit can be sent at once in the 3rd and 4th second. Now, with NC, instead of simply forwarding b1 and b2 outward, C calculates b1 $\bigoplus$ b2 and sends the result (only 1 bit) to the link CD. After that, D sends b1 $\bigoplus$ b2 to E and F in the 4th second. When receiving b1 $\bigoplus$ b2, E and F solve the linear equations to get b1, and b2 because they already have one of the values. The delivery with NC is illustrated in Figure 2.6(b), which demonstrates that using NC, the time is saved by 1 second, *i.e.*, the bandwidth on the link CD is saved. By sending coded data through communication links and decoding them at nodes, NC transfers bandwidth consumption to CPU consumption. Since the CPU power increases much faster than the bandwidth capacity does, NC improves performance in terms of bandwidth utilization.

Theoretically, any coding schemes can be used by intermediate nodes to encode received data. However, it has been shown that random linear codes, one of the simplest coding schemes, using Galois fields of a limited size are sufficient to implement network coding in a practical network setting [12, 32]. The idea of randomized network coding, proposed by Ho *et al.* [32], is that a



(a) Relay Network (b) Network Coding.

Figure 2.6: Traditional Relay Vs. Network Coding.

node transmits on each outgoing link a linear combination of incoming messages, with independently and randomly chosen coding coefficients over some finite field. The practicability of the coding scheme is demonstrated in [33]. The authors have concluded that randomized network coding can be designed to be "robust to random packet loss, delay, as well as any changes in network topology and capacity". It was shown that sessions with randomized network coding can achieve "close to the theoretically optimal performance".

2.4 Scalable Video Coding

The term "Scalable Video Coding" (or SVC) is used for both the concept of scalable coding schemes in general and the particular new scalable design that has been standardized as an extension of the H.264/AVC standard. In this dissertation, SVC is mainly used for the new standard while the general concept is referred to as layered coding or scalable video coding (without capitalization).

The concept of layered coding is that a video is encoded into different quality layers. The base layer (Layer 0) is the most important one as it is required to decode other layers. Other layers are enhancement layers as they provide better quality, *i.e.*, the more layers, the higher quality. There is also coding dependency among enhancement layers. Usually, higher layers depends on lower ones for decoding. An alternative of scalable video coding is multiple description coding (MDC) [34, 35]. In MDC, a video is encoded into different descriptions. Different from layers in layered coding, descriptions in MDC are coding independent, which means that each description can be decoded independently. The most common implementation of MDC generates two equal rate descriptions so that each description alone provides low but acceptable quality and both descriptions together lead to higher quality. Although MDC offers independent descriptions, it is not efficient in coding compared to single layer coding, and does

not provide different scalability modes as in layered coding. In addition, for the same video quality, MDC often generates higher bit rate streams than SVC does.

In the following, the structure and the scalability features of SVC are presented. For more details, interested readers are referred to [9, 36, 37, 8].

2.4.1 Scalability Modes

SVC supports three modes of scalability: *temporal*, *spatial*, and *quality* scalability. In spatial scalability and temporal scalability, subsets of the bit stream represent the source content with a reduced picture size (spatial resolution) or frame rate (temporal resolution). With quality scalability, the substream provides the same spatial-temporal resolution as the complete bit stream, but with a lower fidelity (Signal-to-Noise Ratio). The scalability structure is characterized by three syntax elements: D_ID, T_ID, and Q_ID for spatial, temporal, and quality scalability respectively. Different scalability dimensions can be combined in one stream. However, an SVC bit stream does not need to provide all types of scalability. Since the support of quality and spatial scalability usually comes with a loss in coding efficiency relative to single-layer coding, the tradeoff between coding efficiency and the provided degree of scalability can be adjusted according to the needs of an application.

In addition to different (combined) scalability dimensions, SVC defines coding profiles for high level uses. Profiles and levels specify conformance points to facilitate interoperability between applications that have similar functional requirements. A profile defines a set of coding tools that can be used to generate a bit stream, whereas a level specifies constraints on certain key parameters of the bit stream. The SVC Amendment of H.264/AVC specifies three profiles for SVC [8]: Scalable Baseline, Scalable High, and Scalable High Intra. The Scalable Baseline profile is mainly intended for conversational and surveillance applications that require a low decoding complexity. In this profile, the support for spatial scalable coding is restricted to resolution ratios of 1.5 and 2 between successive spatial layers in both horizontal and vertical direction and to macroblock-aligned cropping. Furthermore, the coding tools for interlaced sources are not included in this profile. For the Scalable High profile, which was designed for broadcast, streaming, and storage applications, these restrictions are removed and spatial scalable coding with arbitrary resolution ratios and cropping parameters is supported. Quality and temporal scalable coding are supported without any restriction in both the Scalable Baseline and the Scalable High profile. Bit streams conforming to the Scalable Baseline and Scalable High profile contain a base layer bit stream that conforms to the restricted Baseline profile and the High profile of H.264/AVC, respectively. Bit streams conforming to the Scalable High Intra-profile, which was mainly designed for professional applications, contain only IDR pictures (for

all layers). Beside that, the same set of coding tools as for the Scalable High profile is supported [9].

2.4.2 SVC Structure

An SVC video is organized into Network Abstraction Layer (NAL) units, which are packets that each contains an integer number of bytes. NAL units are grouped into logical entities. A *layer representation* (LR) consists of all NAL units representing an original picture and pertaining to a combination of D_ID and Q_ID. An *access unit* (AU) consists of all LRs that represent an original picture. The decoding of an AU results in exactly one decoded picture. A *group of pictures* (GOP) is a group of successive pictures. The GOP structure specified by picture types (I-, B-, P- pictures) determines the temporal scalability of the stream. A *coded video sequence* represents an independently decodable part of a NAL unit bit stream. It starts with an instantaneous decoding refresh (IDR) AU, and following AUs can be decoded without decoding any previous pictures of the bit stream. It ends before the next IDR AU or at the end of the bit stream, whichever is earlier. Fig. 2.7 shows the structure.

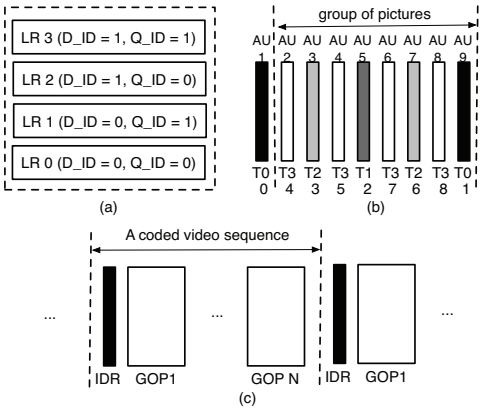


Figure 2.7: An example of the SVC structure. (a) An AU consisting of four LRs. (b) A GOP consisting of eight pictures (AUs) and coded with hierarchical B-pictures. The symbols *Tk* specify the temporal layers with *k* representing the corresponding T_ID. The numbers below specify the coding order. (c) A coded video sequence.

2.4.3 Decoding Dependency

At any AU, an LR of a smaller D_ID may be used for inter-layer prediction by an LR with a greater D_ID, and for a particular D_ID, an LR with Q_ID always uses the LR with Q_ID-1 for inter-layer prediction. An LR is called an MGS (Medium Granularity Scalability) LR if its

Q_ID is greater than 0. For the target dependency layer for output, the maximum value of Q_ID may freely change across AUs without affecting the conformance of the stream. However, removal of MGS LR usually causes a drift between the decoded pictures reconstructed in the encoder and in the decoder. Finally, a given T_ID typically depends on the lower T_ID but never depends on any higher T_ID. An example of scalability features of SVC is presented in Table 2.2.

Table 2.2: An example about the scalability structure of an SVC stream.

Scalable Layer	Resolution	Framerate	Bitrate	(D_ID,T_ID,Q_ID)
0	176x144	3.7500	106.00	(0,0,0)
1	176x144	3.7500	206.00	(0,0,1)
2	176x144	3.7500	341.00	(0,0,2)
3	176x144	7.5000	430.00	(0,1,0)
4	176x144	7.5000	454.00	(0,1,1)
5	176x144	7.5000	508.00	(0,1,2)
6	176x144	15.0000	594.00	(0,2,0)
7	176x144	15.0000	631.00	(0,2,1)
8	176x144	15.0000	721.00	(0,2,2)
9	352x288	3.7500	666.00	(1,0,0)
10	352x288	3.7500	1010.00	(1,0,1)
11	352x288	7.5000	1364.00	(1,1,0)
12	352x288	7.5000	1454.00	(1,1,1)
13	352x288	15.0000	1838.00	(1,2,0)
14	352x288	15.0000	1963.00	(1,2,1)

The stream described in Table 2.2 has 3 quality levels, 3 temporal levels, and 2 spacial levels. With the combined scalability, it offers a wide range of scalability. The base layer produces a video stream with a resolution of 176x144, a framerate of 3.75 frame/second, and a bitrate of 106 Kbit/second; and the complete stream has a resolution of 352x288, a framerate of 15 frame/second, and a bitrate of 1963 Kbit/second. The scalable layer determines the coding dependencies of the streaming. Generally, higher scalable layers depend on lower scalable layers for decoding. Going through the table from Layer 0 to Layer 14, we can examine the coding dependency rules defined by the SVC standard.

2.4.4 BitStream Switching

SVC allows switching between different scalable levels during streaming to provide adaptability. However, switching operations can only occur at specific points of a stream:

- ▷ Switching between spatial layers can only occur at IDR AUs.
- ▷ Switching between quality layers within a spatial layer can occur at *any* AU.

▷ Switching between temporal layers within a spatial layer can occur at *any* AU or only at temporal layer switching points depending on encoding parameters.

With flexible switching capacity, SVC enables adaptation along the dimensions. However, when using them exclusively, no conclusion can be drawn on which adaptation path along multiple scalability dimensions is subjectively most pleasing. The encoder of the stream may have knowledge about this property and specify a recommended adaptation path through the $Priority\_ID$ syntax element, which is assigned per NAL unit. It should be noted that no assumption on a relation between $Priority\_ID$ and the values of D_ID, Q_ID, or T_ID is explicitly made in the SVC standard.

2.5 Social Networking

The notion of a social network and the methods of social network analysis have attracted considerable interest from the social and behavioral science community in recent decades. A simple definition of a social network is that it is a network in which social entities are considered as nodes, and social relationships are considered as links. Social network analysis focuses on relationships among social entities, and on the patterns and implications of these relationships, not on individuals and their attributes. Many researchers have realized that the network perspective allows new approaches to answer standard social and behavioral science research questions by giving a precise formal definition to aspects of the political, economic, or social structural environment. Social network data are defined by social entities and relations. They can be represented by matrices or graphs. With the matrix representation, the value at element (i,j) represents relationships between i and j, *e.g.*, if $M(i, j) = 1$, i has a relationship with j. In the graph representation, a vertex is an entity, and an edge represents for a relationship. Such representations are compact, systematic, and allow applying computers to analyze data.

For example, consider a social network of a group of four people: Bob, Carol, Ted, and Alice. Suppose that Bob likes Carol and Ted, but not Alice; Carol likes Ted, but neither Bob nor Alice; Ted likes all three of the other members of the group; and Alice only likes Ted. This social network can be represented by a matrix in Table 2.3 or by a graph in Figure 2.8.

Table 2.3: The matrix representation of the example social network.

	Bob	Carol	Ted	Alice
Bob	—	1	1	0
Carol	0	—	1	0
Ted	1	1	—	1
Alice	0	0	1	—

With the matrix representation in Table 2.3, there are many things that might be immediately inferred when we see the arrayed data that we might not have thought of from reading the description of the pattern of ties in words. For example, scanning across each row, we notice that Ted likes more people than Bob, than Alice and Carol. *Is it possible that there is a pattern here? Are men more likely to report ties of liking than women are?* (actually, research literature suggests that this is not generally true). In addition, it should be noted that the values on the main diagonal are set based on the definition of the relationship, *e.g.*, if we need to consider whether Bob likes himself, the value at (Bob, Bob) is set to 0 or 1, otherwise it is empty. Such self-relationships may not be clearly revealed when we describe relationships in words or graphs.

Graphs that represent social network data are called "sociograms" or "social graphs". Viewing a social network in such a graph with nodes and links allows us to apply graph theory to extract useful information, *e.g.*, clusters, in-degree and out-degree of nodes, average path length, diameter, etc. For example, if we change the "liking" relationship in our example to "know-about" relationship, we will know that Ted is most famous in the network because everyone knows about him and he knows everyone. Consequently, if Carol wants to know about Bob, she should ask Ted.

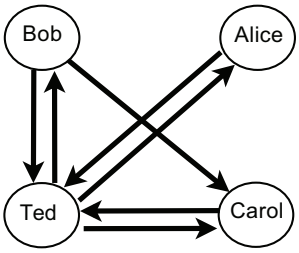


Figure 2.8: The graph representation of the example social network.

Chapter 3

Literature Review

P2P streaming has been extensively studied over the last decade. P2P-based streaming software, *e.g.*, PPLive [3], Coolstreaming [4], UUSee [6], SopCast [38], TV Ants [5], is serving millions of users to watch streamed media. The success of existing P2P streaming systems demonstrates the possibility of streaming video over the Internet. However, as the number of users significantly increases and the user expectation about streaming quality gets higher and higher, providing satisfactory P2P streaming services becomes more and more challenging. This chapter summarizes related studies and discusses differences between this research work and the literature. The main goal is to provide a big picture of research in P2P live video streaming and to emphasize the contributions of this work. In this chapter, *first*, a taxonomy of P2P streaming systems is presented. *Second*, notable studies are summarized and discussed. *Finally*, pioneering work in applying network coding, SVC, and social networking to P2P systems in general and to P2P streaming systems in particular is described.

3.1 Taxonomy of P2P Streaming

P2P streaming systems can be classified based on their overlay construction or on their streaming behavior. Since the overlay determines the data delivery process, different overlay structures often go with different streaming algorithms. Therefore, a P2P streaming protocol, including overlay construction and streaming algorithm, can be classified by its overlay structure. An overlay-based taxonomy is depicted in Figure 3.1.

In general, in structured P2P streaming systems, the overlay has a rigid structure, which is built and maintained strictly during the streaming session. Since the structure of the overlay is fixed, the data flow is deterministic. For example, in tree-based overlays, video packets are distributed from the root to the leaves of a tree. The most popular overlay structure in structured P2P streaming is tree-based structure, in which, peers are organized into one tree or multiple

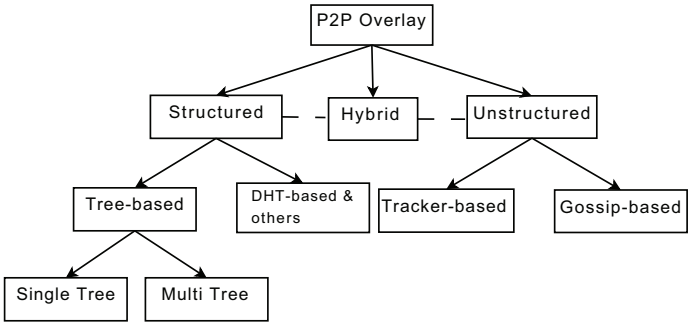


Figure 3.1: Overlay-based taxonomy of P2P streaming systems.

trees to receive video packets. Besides, DHT-based (Distributed Hash Table) algorithms can be used to construct structured overlays. On the other hand, unstructured P2P streaming systems do not maintain fixed overlays. Peers can frequently change their connections with other peers depending on the data availability. Each peer has a list of other peers, called neighbors. Peers exchange data availability information with their neighbors and connect to those which have packets that they need. To deal with peer dynamics, the neighbor list at each peer needs to be updated frequently to find better neighbors. To know about other available peers in the system for such updates, peers can request a tracker (tracker-based unstructured P2P streaming) or run a gossip-based membership management protocol (gossip-based unstructured P2P streaming).

The other way to classify P2P streaming systems is based on their streaming behavior with respect to adaptability. We consider that traditional P2P streaming systems are those which use single-layer video coding techniques and provide the same quality to all users regardless of their available bandwidth. On the other hand, adaptive P2P streaming systems encode a video into different quality levels and can offer differentiated QoS in terms of video quality to their users. Several video coding techniques can be used to create (sub-)streams with different quality levels for a video. In multiple version approaches, a raw video is encoded several times to create different independent versions with different quality levels. Multiple Description Coding (MDC) encodes a video into different descriptions. Each description is playable and provides a basic quality. The overall quality at each peer increases with the number of descriptions it receives. Scalable (layered) coding techniques encodes a video into different layers. The base layer provides the lowest (basic) quality level. The more layers a peer receives, the better quality it perceives. However, different from MDC, there are coding dependencies among layers. Higher layers depend on lower layers for decoding. Therefore, the base layer is the most important one as it is required to decode other layers. The classification is summarized in Figure 3.2.

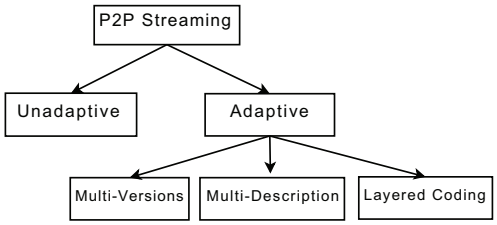


Figure 3.2: Adaptability-based taxonomy of P2P streaming systems.

3.2 Traditional P2P Streaming Systems

Without the P2P paradigm, a server broadcasts the content to all participating clients. However, as the number of users increases, the server may become overloaded, resulting in high loss rates, *i.e.*, the client-server model is not scalable for large-scale video streaming. To alleviate the bandwidth demand on the server, the P2P principle is exploited.

Tree-based P2P streaming systems are firstly introduced. Jannotti *et. al.* introduce Overcast, a pioneering system of application-level multicast [39]. Overcast uses a single-tree overlay to distribute the content. One disadvantage of single-tree overlays is that the leaf peers only receive the content, *i.e.*, the upload capacity of the leaf peers is not utilized. To better utilize peer upload capacity, Mutualcast [40] encodes the content into multiple stripes and distributes the stripes across separate multicast trees with disjoint interior peers. Any peer could be an interior node in one of the multicast trees, and contribute its upload capacity. Despite the advantages of push-based approaches in fast data delivery, constructing and maintaining a well-organized distribution tree burdens peers and links with heavy control overhead, especially in a dynamic environment, as demonstrated by SpreadIt [41]. To relieve the burden of control overhead in the multicast trees to a certain degree, NICE [42] and ZIGZAG [43] manage peers as multi-layer hierarchical clusters, *i.e.*, a tree of clusters. These approaches show improvements in overlay management overhead, but still the overhead is unavoidable. In a nutshell, although the push-based approaches lead to short delays in distributing the content, it is not generally employed in practice, mainly due to the complexity and difficulty in maintaining the structured overlay, especially under the presence of peer dynamics.

In sharp contrast to the push-based approaches, the pull-based, also known as mesh-based, approaches do not enforce any rigid structure among peers. Instead, connections are established dynamically based on the content availability at each peer. The streaming content is presented as a series of segments, each representing a short duration of playback. The content exchange in this approach is best described as a “data-driven” or “swarming” style of multicast. In data swarming, each peer advertises to its neighbors the segment availability in its buffer, and the neighbors explicitly request the segments as needed. The primary advantages here are simplic-

ity in maintaining peer connectivities and robustness in dynamic networks. Nonetheless, the delay in delivering the live content to each participating peer is inevitably increased, ascribed to the periodical exchange of segment availability among the peers. Bullet [44], for example, constructs a mesh on top of a multicast tree to improve overall bandwidth utilization. The underlying tree structure provides an environment for periodic dissemination of peer identifiers and content availability. CoolStreaming [4] completely abandons the tree structure and employs a gossiping protocol for peer discovery. A peer in CoolStreaming maintains not only a list of neighboring peers, but also a summary of available content on its neighbors. Based on this information, segments are scheduled to be streamed from the appropriate neighbors, while striving to meet the playback time. The performance of pull-based approaches highly depends on the choice of peer discovery protocols and data swarming strategies. Hybrid designs [45, 46, 47, 48, 49] are proposed to combine the better resilience to dynamics from pull-based approaches with the better delay and stability from push-based approaches. Essentially, the connections are initialized based on the content availability at each peer, and then portions of the original content are pushed onto each connection.

Discussion: Among others, these notable studies have contributed to the success of working P2P streaming systems nowadays. However, they have the common problem of unadaptability to user heterogeneity, as presented in Chapter 1. In addition, although significant efforts have been spent on dealing with high churn rates, it is still one of the most challenging problems in P2P streaming. Different from those studies, this work towards adaptive and robust P2P streaming systems, which can provide best possible streaming quality to users according to their bandwidth under network fluctuations and high churn rates.

3.3 Adaptive P2P Streaming

Recognizing that the weaknesses of traditional P2P streaming systems is the lack of unadaptability, people have turned their attention to adaptive P2P streaming.

Orchard [50], SplitStream [51], and CoopNet [52] use multiple-tree overlays and multiple description coding (MDC) to provide robust and adaptive P2P streaming services. The general idea is that each video description is delivered over one tree, and peers can receive more than one description by being a node of more than one tree. With MDC, any description is decodable and the overall quality improves with the number of descriptions received. Thanks to this feature of MDC, these systems can better deal with peer dynamics and user heterogeneity, compared to traditional P2P streaming systems.

In layered P2P streaming, the work of Cui *et al.* [53] and Rejaie *et al.* [54] can be considered as the two first major efforts. In [53], Cui *et al.* formulate the problem of bandwidth and

data availability constraints as an optimization problem to maximize the net benefit which is defined by system benefit – the overall streaming quality of all peers – excluding system cost and the server bandwidth consumption. They use global knowledge about participating peers and a greedy approach, in which a receiving peer requests individual layers from supplying peers so that it always maximally utilizes the outbound bandwidth of the peer with the smallest number of layers. Although the algorithm is shown to be able to address the constraints of bandwidth and data availability, this work did not take dynamics of bandwidth variations into account. Later, Liu *et al.* [55] present another approach to the problem. They formulate it as an optimization problem with the constraints of available bandwidth and layers, and use an approximation algorithm, FABALAM, to simplify the problem. Although FABALAM has been demonstrated to be able to achieve better performance than the approach of Cui *et al.*, they both rely on static layer-to-sender mapping, and a layer is provided by only one sender.

Rejaie and Ortega introduce PALS [54], a receiver-driven approach for quality adaptive playback, which addresses the bandwidth variations in a timely manner. In PALS, a layer is divided into packets and provided by multiple senders. Therefore, it better utilizes the available bandwidth of senders. In addition, PALS addresses bandwidth variations and peer dynamics in a timely manner. The receiver peer monitors the available bandwidth from its senders and periodically requests a list of packets from each sender, and each sender delivers the requested packets to the receiver in the given order over a congestion-controlled connection. Each peer is able to adapt to bandwidth variations by adding another layer or dropping the current top one. Recently, Magharei and Rejaie present an extended version of PALS with extensive simulations in ns-2 [56]. However, although PALS is designed to P2P streaming, its performance has only been currently evaluated for the case of streaming from multiple senders to one receiver. Its performance in P2P scenarios has not yet been shown.

The authors of [57] propose an adaptive streaming mechanism which is based on an efficient LC content scheduling scheme to coordinate among receiver and sender peers. Sender peers are selected on the basis of RTT in their work. Then, all the senders are sorted by RTT. The sender with the minimum RTT to the receiver is assigned to deliver the base layer while enhancement layers are assigned to the sender peers in the increasing order of RTT. They also introduce buffer management to (1) control lost packets and re-request them using ARQ (Automatic Repeat Request); (2) drop higher layer packets when lost packets of a layer cannot arrive on time.

Discussion: Compared to the above initial work in adaptive P2P streaming, this thesis is different in the following aspects. Some of the related work uses tree-based overlays, while this thesis targets mesh-based systems. Existing unstructured adaptive P2P streaming studies do not propose a complete P2P streaming protocol, while this thesis considers different aspects of adaptive P2P streaming from overlay construction to quality adaptation. In addition, no

previous studies use a specific video coding technique, while the SVC standard is considered in this work. This does not limit the application of the proposed protocols in practice, but actually, makes them more practical as the SVC is the only scalable video coding standard available.

3.4 Overlay Construction in P2P Streaming Systems

Overlay construction is an important component in a P2P streaming system. By summarizing P2P streaming protocols, Sections 3.2 and 3.3 briefly describe different overlay construction mechanisms (structured overlays including single-tree, multi-tree, and unstructured overlays), and how they are used to distribute multimedia contents. This section goes into detail on how such overlays are constructed in the literature.

The tree-based approaches are stemmed from the philosophy of IP multicast. In such a paradigm, peers are organized into one or more multicast trees rooted at the source. The source splits the original content into a set of small data pieces and "pushes" them to descendants among the trees. Due to bandwidth limitations, usually a subset of the data pieces can be transmitted between peers; hence, a receiver rarely receives the entire content from the same parent. The single tree building protocol in Overcast is a basic protocol [39], and is summarized as follows: when a new node contacts the root to join the network, the root becomes the current node. Next, the new node begins a series of rounds to locate itself further away from the root without sacrificing bandwidth back to the root. In each round, the new node considers its bandwidth to the current node as well as the bandwidth to the current node through each of the current node's children. If the bandwidth through any of the children is about as high as the direct bandwidth to the current node, then one of these children becomes the current node and a new round begins. In the case of multiple suitable children, the child closest (in terms of network hops) to the new node is chosen. If no child is suitable, the search ends with the current node being the parent of the new node.

The construction of one tree in multiple tree overlays is similar to the above single tree construction in the way that a new node is placed such that the bandwidth bottleneck from the root node to leaf nodes of the tree is minimized. The main difference between constructing a single-tree overlay and a multi-tree one is that a multi-tree construction method selects which tree should contain the new node and creates connections between the new node and other nodes in other trees. Magharei *et. al.* summarize a representative multiple tree construction algorithm as follows [58]: each peer is an internal node in only one tree and leaf node in other participating trees. When a peer joins the systems, it contacts a rendezvous node to identify a parent in the desired number of trees. To keep the balance among trees, the new node is added as an internal node to the tree that currently has the minimum number of internal nodes. To maintain short

trees for faster delivery, the new internal node is placed as a child for the node with the lowest depth that can accommodate a new child or has a child that is a leaf. In the latter case, the new node replaces the leaf node and the partitioned leaf should rejoin the tree similar to a new leaf. When an internal node of a tree leaves, each one of its child nodes as well as their subtree are partitioned from the original tree, and should rejoin the tree. Nodes in such a partitioned subtree initially wait for the root of the subtree to rejoin as an internal node. If the root is unable to join the subtree after a certain period of time, individual peers in a partitioned subtree independently rejoin the tree with the same position (as leaf or internal node).

Magharei *et. al.* [58] and Seibert *et. al.* [59] present comparisons of the two kinds of overlays and demonstrate that mesh-based approaches consistently exhibit a superior performance over tree-based approaches in dynamic environments, while, in stable environments, tree-based systems are better in terms of delivery time. Due to the advantages of unstructured overlays, this thesis only considers unstructured P2P streaming protocols.

An unstructured overlay is built by a membership management method, run at each peer independently from other peers, and peers can change their connections frequently to achieve better quality. The main function of membership management at each peer in unstructured overlays is to select neighbors with which that peer exchanges data. It is the membership management which sets up the local view of each peer to the system and directly affects the video quality each peer receives. SCAMP [60] is a well-known gossip-based membership management protocol. The mechanism of SCAMP is as follows: when a new peer P joins the network, it sends a subscription request to an arbitrary peer. P starts its membership management protocol with the neighbor list containing only the identifier of that arbitrary peer. When a peer receives a new subscription request, it forwards the new node identifier to all of its neighbors. It also creates c additional copies of the new subscription and forwards them to randomly chosen neighbors (for failure tolerance). When a peer receives a forwarded subscription, it adds the new peer to its neighbor list with a probability p depending on the current size of the neighbor list. If it does not add the new peer, it forwards the subscription to a randomly chosen neighbor. Those subscriptions are only destroyed until some peers keep them. Each peer also runs a periodic check to avoid being isolated. If a peer does not receive any messages for a given period, it resubscribes through a randomly chosen neighbor or a rendezvous peer. Such membership management offers a randomized and scalable partial view of the system at each peer, which is robust to peer dynamics.

Jelasity *et. al.* [61] propose a framework to implement peer sampling services, which are also based on gossiping. Different from SCAMP, the idea here is that a peer chooses one of its neighbors to exchange views. After the view exchange, both peers know more peers in the system for updating their view. The authors demonstrate that such peer sampling services

construct more scalable and robust overlays than SCAMP-based protocols. In addition, since explicit attempts are made towards the construction of an overlay in SCAMP, peer sampling services are less expensive in terms of bandwidth cost as fewer requests are sent in the network. However, the current peer sampling services are not quality-aware as they do not take peer capacity into account when choosing neighbors.

As one of the first efforts on constructing mesh-based overlays for layered P2P streaming, Zhao *et al.* proposed LION [62], a layered overlay multicast system. LION progressively organizes peers into layered meshes. Within each mesh, the delivery of one quality layer is carried out. Each peer can subscribe to a proper number of meshes to maximize its throughput by fully utilizing its available bandwidth. Although each video layer is delivered in a mesh which is unstructured, the whole overlay structure of LION is quite well-organized and maintained by a distributed heuristic algorithm. LION aims at supporting small-scale applications in stable environments.

There has been work tailoring SCAMP for layered P2P streaming. Xiao *et. al.* propose OCals [63], which constructs the overlay in two stages. The first stage, SCAMP-based, is to probe existing nodes to find a certain number of logical partners, which are interested in the same set of layers. In the second stage, it will select neighbors for each layer based on the RTT.

The authors in [64, 65] present the idea of gradient overlays, which locate higher capacity peers closer to the source than lower capacity peers. Generally, such overlays have some similar features with our proposed quality-aware overlays. However, there are several main differences between them. The work of Sacha *et. al.* [64] is not targeted to P2P streaming applications. It elects super-peers with highest utility to discover globally similar neighbors while lower utility peers have mostly random neighbors. The election method is unaffordable for P2P streaming with the strict timing requirement. Payberah *et. al.* present gradienTv [65] using gradient overlays for live P2P streaming. GradienTv is a multiple tree based system for single-layer P2P streaming as the media source splits the media into a number of stripes and constructs an overlay tree for each stripe. The proposed protocols are designed for unstructured layered P2P streaming. In addition, GradienTv constructs the gradient overlay by using two neighbor lists – random view and similar view – and tries to maintain the tree structure for all peers, while, in our protocol, peers locate themselves in different locations by exchanging their *local* view.

Discussion: While overlay construction in traditional P2P streaming systems has been studied extensively, constructing overlays suitable for layered P2P streaming has not been paid much attention. Existing tree-based overlays for adaptive P2P streaming are not scalable for large scale systems, while there is a limited number of studies on unstructured ones. Towards high performance adaptive P2P streaming, this thesis also proposes efficient unstructured overlays which are specific for layered P2P streaming.

3.5 Network Coding in P2P Systems

The first application of NC in P2P systems is in P2P file sharing applications. In a large P2P file sharing system, finding an optimal packet propagation scheme that minimizes the peer download time is very difficult; especially, practical systems can not rely on a central scheduler and, instead, allow peers to make local decisions. The scheduling problem becomes increasingly difficult as the number of peers increases, when peers are at different stages in their downloads, or when incentive mechanisms are introduced to prevent leeching clients. One pioneering work on applying NC in P2P systems is the Avalanche project of Microsoft Research [66, 67]. Its authors demonstrate, in both simulation studies and realistic experiments, that randomized network coding may improve the overall performance of P2P content distribution. The idea is that every time a peer sends a packet to another peer, the source peer generates and sends a linear combination of all the packets available to it (similarly to XORing multiple packets). After a peer receives enough linearly independent combinations of packets, it can reconstruct the original information. With randomized NC, when the probability of receiving innovative coded packets is high, peers do not need to explicitly schedule when and where to receive coded packets. Whenever a peer receives enough linearly independent coded packets, it can decode and get the original data. The NC-based P2P file sharing scheme has been compared with BitTorrent [68], one of the most successful P2P content distribution protocols, and the claimed performance benefits provided by network coding are: "the expected file download time improves by more than $20 - 30\%$ with network coding compared to coding at the server only and, by more than $2 - 3$ times compared to sending unencoded information" [66]. Other related work also demonstrates significant benefits of NC in P2P file-sharing networks [69, 70].

Annapureddy *et al.* [71] show that network coding helps to provide high quality Video-on-Demand (VoD) services. Network coding is applied over small time-windows (e.g., a segment with a few seconds worth of video frames) of a single-layer stream. The coding prevents the occurrence of rare blocks within a segment. In addition, it ensures that bandwidth is not wasted in distributing the same block multiple times, i.e., it minimizes the risk of making incorrect upload decisions.

In single layer P2P live streaming, Wang and Li present Lava [72], the first fair evaluation on the feasibility and effectiveness of random network coding in live P2P streaming sessions, with strict timing and bandwidth requirements. While a traditional P2P streaming protocol sends original segments, they implement a "plug-in" to do random network coding on blocks of each segment before sending them to other peers and to decode received coded blocks on-the-fly. They discover that NC provides some marginal benefits when peers are volatile with their arrivals and departures, and when the overall bandwidth supply barely exceeds the demand. While Lava focuses on a fair comparison study without any changes of the P2P streaming pro-

tocol, in their follow-up work, Wang and Li redesign the whole P2P streaming protocol to take full advantage of NC. Their new NC-based P2P streaming protocol, R^2, is considered as the current state-of-the-art in applying NC to live P2P streaming. In R^2, random network coding is used as follows: the live stream is divided into segments. This step is similar to traditional P2P streaming protocols. Each segment is then further divided into n blocks $[b_1, b_2, ..., b_n]$, each b_i has a fixed number of bytes k (referred to as the block size). When a peer needs to send out a coded block, it applies NC on a certain number of blocks of a particular segment it has received so far. In R^2, NC is used for each segment separately, not across different segments. For example, when a segment is selected to serve for a peer p, the sender independently and randomly chooses a set of coding coefficients $[c_1^p, c_2^p, ..., c_m^p]$, $(m \leq n)$ in GF(2^8). It then randomly chooses m blocks – $[b_1^p, b_1^p, ..., b_m^p]$ – out of all the blocks in the selected segment that it has received so far (all the original blocks in the segment if the seed is a streaming server), and produces one coded block x of k bytes:

$$x = \sum_{i=1}^{m} c_i^p \cdot b_i^p$$

With Gauss-Jordan elimination implemented in the decoding process, a peer starts to progressively decode a packet, as soon as it receives the first coded block of the segment. When a total of n linearly independent coded blocks $X = [x_1, x_2, ..., x_n]$ have been received, the original blocks can be immediately recovered as Gauss-Jordan elimination computes $b = A^{-1}X^T$, where A is the matrix formed by coding coefficients of X. In addition to the use of NC, another important design point of R^2 is the random push mechanism. In a traditional pull-based protocol, a segment is explicitly requested by a peer p. The sender then serves the request by sending the segment. To better take advantage of random network coding, in R^2, the sender randomly chooses a segment whenever it produces one coded block, among all remaining segments that p has not completely received. The coded block is then sent to p without the need of any requests. Since all coded blocks are equally innovative, all senders of p cooperatively serve the missing segments on p, without any explicit exchanges of protocol messages. This is referred to as perfect collaboration. To meet the playback deadline, segments are chosen based on the playback buffer of p (available via buffer map exchanges): segments close to the playback deadline have a higher priority in being selected. With random push and random network coding, R^2 is evaluated by experiments on an actual implementation, real network traffic, and emulated peer upload capacities and demonstrates the feasibility and benefits of NC in live P2P streaming.

NC has also been used in adaptive P2P streaming. The general method is to use NC for distribution of one video description or layer, *e.g.*, in LION [62], where different coding engines (different sets of coefficients) are used for different descriptions or layers.

Discussion: There has been work in applying NC to adaptive P2P streaming. However, the

main difference between this thesis and state-of-the-art studies is that this thesis combines NC with the particular scalability structure of the SVC standard.

3.6 Social Networking in P2P Systems

Generally, social networking can be applied to P2P systems in two ways. *First*, P2P systems can mimic how people form a social network and how they query, by preference, their friends or acquaintances to construct overlays. Such overlays achieve more efficient routing and data locating. Following this approach, Lin *et al.* introduce SocioNet, a social-based multimedia access system for unstructured P2P networks [20]. SocioNet describes objects (multimedia files) using multiple attributes, and, in turn, characterizes peers by the objects they hold. For example, a music file can have attributes about genre, artist, title, and language. Peers are characterized by their profile, which is a weighted vector of preferences. Then, SocioNet clusters peers based on their similar characteristics. The similarity estimation is calculated by the cosine similarity measure. In addition, peers also maintain a certain number of connections to others with different interests as "shortcuts" that connect them to other parts of the network. They show that such social-based overlay construction creates small-world networks, and achieves a higher success ratio than nonsocial-based overlays.

Second, P2P systems can import social graphs from other social networks, and establish connections among peers based on their relationships within these social graphs. TRIBLER [19] is such a system. It (1) facilitates the formation and maintenance of social networks by importing existing user contacts from other social networks, *e.g.*, MSN, and (2) exploits the social networks to create connections among peers. Implemented as a set of extensions to BitTorrent, TRIBLER has been demonstrated to be able achieve fast, trusted content discovery and recommendation, and a significant improvement in download performance. More recently, Liu *et al.* [21] present a new incentive paradigm, Networked Asynchronous Bilateral Trading (NABT), based on social networking. The idea of NABT is that each peer has a set of friends, which can be potentially derived from other social networks, and each pair of friends keeps track of a credit balance between them. Credits can be used for 'buying' services, *e.g.* downloading a file, and can be exchanged via friends-of-a-friend relations. Simulations show that NABT can have high trading efficiency, provide service differentiation, and discourage free-riders.

Discussion: While the above social-based systems are designed for P2P file sharing, the idea of establishing connections among peers based on social relationships can be used in other P2P applications, including live P2P streaming. However, the existence of any existing designs with respect to social P2P streaming has not been reported. To the best of our knowledge, Stir (Chapter 6) is the first attempt using spontaneous social networking in P2P streaming.

Chapter 4

Coding Approaches: Network Coding Meets SVC in P2P Streaming

Convinced that SVC is the enabling coding scheme for adaptive streaming services and NC could help exploit SVC in P2P streaming, we can begin by turning our attention to combination of SVC and NC in P2P streaming. This chapter presents Chameleon, a novel adaptive P2P streaming protocol with NC and SVC. The motivation of using NC with respect to technical problems is presented in Section 4.1. In Section 4.2, a segmentation method for using SVC with the P2P paradigm is described. The combination of NC with SVC is presented in Section 4.3. The design of Chameleon with different design options for its components is presented and justified (with experiments) in Section 4.4. Section 4.5 analyzes simulation results to demonstrate the feasibility and benefits of the combination in building up an adaptive P2P streaming system.

4.1 Motivation for Using Network Coding

In addition to conventional challenges in P2P streaming, *e.g.,* peer selection and packet scheduling, layered P2P streaming poses unique and challenging problems, of which two of the most important issues are *peer coordination* and *quality adaptation*.

With respect to *peer coordination*, the bandwidth and data availability of each peer are constrained and varied, which further limit the data availability (content bottleneck) and bandwidth (bandwidth bottleneck) of downstream peers. Peer coordination is critical to the system performance because it controls the collaboration of sending peers to utilize the available bandwidth of each sender to maximize the delivered quality at the receiving peer. Two important questions are:

- ▷ *Is a layer supplied by one or more than one sender?* If a layer is delivered by only one sender, coordination may be simpler, but the residual bandwidth of each peer may not be fully utilized. In addition, when the sender leaves, the delivery process may have to start from scratch. On the other hand, assigning partial layers to senders better utilizes bandwidth but requires a proper division of a layer across multiple senders.

- ▷ *How do we map packets to senders appropriately?* Given an ordered list of required packets and a list of potential senders, the problem is to determine which packets are to be delivered from each sender. Taking bandwidth heterogeneity and different important levels of packets into account, this problem was shown to be NP-hard [55].

The purpose of *quality adaptation* is to avoid playback skips and to maximize the video quality when bandwidth variations occur. Challenges in quality adaptation are:

- ▷ *How does a peer choose layers to be requested at a point of time?* The selection should be based not only on playback deadlines, but also on streaming quality (the number of layers) the peer aims to achieve. For example, when bandwidth is available, the quality can be increased by receiving more layers. On the other hand, when bandwidth is reduced, the expected quality is decreased to maintain continuous playbacks.

- ▷ *How and when is quality adaptation invoked?* The quality adaptation process should be invoked at reasonable moments to save resources and maintain best quality.

It is believed that NC can help to solve the above problems with ease. With NC, a peer only needs to check if it has received a sufficient number of linearly independent coded blocks, without being concerned with who has been sending them. The probability of receiving "duplicate" blocks is so low that multiple senders can serve blocks to the same receiver without the need of any explicit coordination. In addition, since coded blocks are equally useful to the receiver, the responsibilities of a particular sender can be easily transferred to other senders if it leaves the system. Even the computational complexity of network coding is no longer a concern: Wang and Li [72] have implemented a decoding process using Gauss-Jordan elimination, such that it can be performed while coded blocks are progressively received. Shojania *et al.* [73] propose an implementation that efficiently takes advantage of multiple CPU cores and SIMD instruction sets in modern CPUs. In a nutshell, network coding can be efficiently implemented, and it maximizes the collaboration potential among peers.

However, combining NC with SVC is not straightforward. SVC prioritizes video data to provide different quality levels by allowing the extraction of substreams. Meanwhile, network coding makes data packets equally important to ease the data delivery, and the original data is only recovered when enough linearly independent blocks are received (the *all-or-nothing* property). *How do we combine network coding and SVC? How much can NC help?*

4.2 Proposed Segmentation Method

To be transmitted across IP networks, an SVC stream needs to be divided into segments. The segmentation method should preserve the scalability of the stream so that (i) adaptation can operate on segments (*adaptability*), and (ii) the re-generated stream is a valid stream (*validity*). In an SVC video file, the video entities are arranged in the specific order: from one GOP to another. Within a GOP, AUs are sorted on the decoding order; and within an AU, LRs are sorted on (D_ID, Q_ID). Figure 4.1 depicts this order.

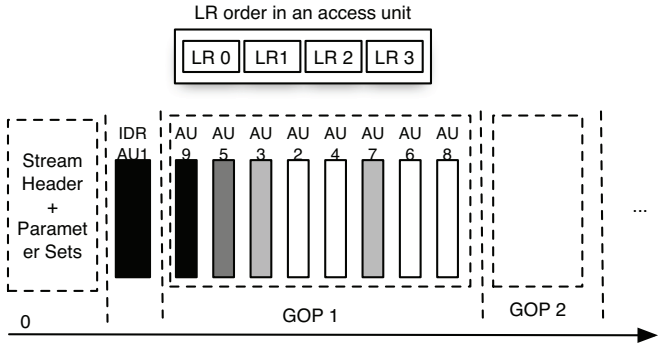


Figure 4.1: The store order of the entities in the video file

In P2P streaming, a peer does not receive a complete stream to extract valid substreams for other peers; it, instead, receives only pieces of video data to constitute a stream according to its download capacity. Therefore, to maintain the adaptability and validity, the video entities should be re-arranged, and the segmentation method should be based on layer switching enabled points to support particular scalability modes. A segmentation method is proposed to segment an SVC stream based on the boundary of GOPs, because switching between temporal and quality levels within a GOP is independent from other GOPs, and spatial switching is only allowed at IDR AUs (outside of any GOP).

In the following, the proposed segmentation for SVC with quality scalability is described. However, it is straightforward that the method can be applied to other scalability modes. An SVC stream is first divided into segments, each of which consists of an integer number of GOPs. Then, within each segment, NAL units are grouped into packets based on Q_ID from the lowest to the highest value. Since each NAL unit header contains (D_ID, Q_ID, T_ID) of the scalable layer it belongs to, it is always possible to recover the original order [37]. Figure 4.2 illustrates the segmentation method.

During streaming, packets are exchanged among peers. Packet 1 in each segment contains the base layer and is necessary for every peer, while other packets can be received or not depending on available download capacity. Since each packet contains all NAL units of one quality

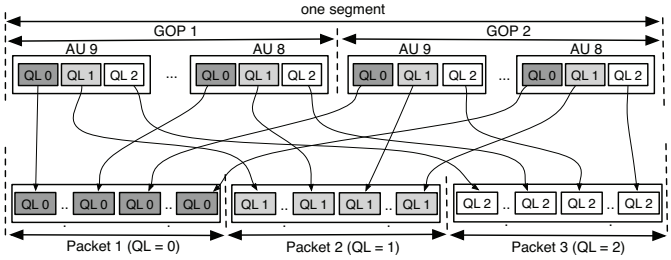


Figure 4.2: An example of the segmentation method where the stream has three quality levels and is divided into segments of two GOPs. The symbols ***QL k*** specify $Q\_ID = k$.

layer in the segment, streams that are generated by dropping one or more packets (except the base packet) are valid streams, ***i.e.***, the segmentation method guarantees the adaptability and validity requirement.

4.3 SVC with Random Network Coding

The approach in *Chameleon* is to apply random network coding to scalable layers, based on which scalability mode the system aims to support. For example, if an SVC stream with a certain number, N_s, of scalable layers is divided into segments as previously proposed, each segment would now contain N_s packets for N_s layers. Each packet is further divided into N blocks $[b_1, b_2, ..., b_N]$, all blocks of a packet have the same number of bytes k (referred to as the ***block size*** of that packet). When a peer performs network coding for layer l, it randomly chooses a set of coding coefficients $[c_1, c_2, ..., c_M]$ ($M \leq N$) in GF(2^8). It then randomly chooses M blocks of layer l — $[b_1^l, b_2^l, ..., b_M^l]$ — out of all the blocks of the layer it has received so far, and produces the coded block x of k bytes:

$$x = \sum_{i=1}^{M} c_i \cdot b_i^l$$

With Gauss-Jordan elimination implemented in the decoding process, a peer starts to progressively decode a packet, as soon as it receives the first coded block of this packet. As a total of N linearly independent coded blocks $X = [x_1, x_2, ..., x_N]$ have been received, the original blocks can immediately be recovered as Gauss-Jordan elimination computes $B = A^{-1}X^T$, where A is the matrix formed by coding coefficients of X. Figure 4.3 shows the combination of network coding and SVC for the stream in Figure 4.3.

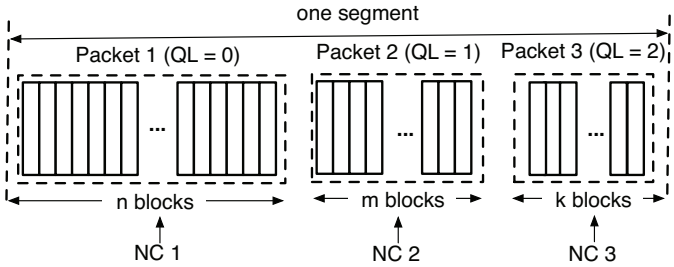


Figure 4.3: An example of the combination of network coding and SVC. Packet 1, 2, and 3 are divided into n, m, and k blocks, respectively. Network coding with different numbers of unknowns (n, m, and k) is used for different quality levels.

4.4 Chameleon: Adaptive P2P Streaming with Network Coding

Chameleon seamlessly integrates SVC with network coding. We consider a typical P2P streaming session with a number of dedicated streaming servers, and a large number of peers. Peers participate in and depart from a session in unpredictable ways, and they are heterogeneous with different bandwidth capacities.

4.4.1 System Overview

The primary design goal of Chameleon is to effectively utilize available bandwidth of each peer to maximize delivered quality under bandwidth variations. Figure 4.4 shows the internal architecture of Chameleon with key components and their relations. When a peer joins the system or when it needs to update the neighbor list for better quality, it creates neighboring relationships with other peers. A list of available peers can be provided by a rendezvous peer or by exchanging membership information, *e.g.*, using SCAMP [60]. The *neighbor selection* component chooses a number of peers to be neighbors. Information about each neighbor, *e.g.*, IP address, average experienced quality, current number of neighbors, etc. is stored in the neighbor list. During streaming, a peer needs to decide how many quality levels it aims to receive according to its current bandwidth capacity. The *quality adaptation* component will make decisions on keeping, increasing, or decreasing the current quality level, based on the status of the playback buffer and the available download capacity. When adaptation occurs, the *sender selection* component will select potential senders from the neighbor list, based on the decision from the adaptation process. The *peer coordination* component will send layer requests to the selected senders. The senders are expected to collaboratively send coded blocks of the requested layer to the receiver. When the playback deadline is reached, one segment in

the playback buffer is sent to the player. In addition to the above components, *buffer maps* are

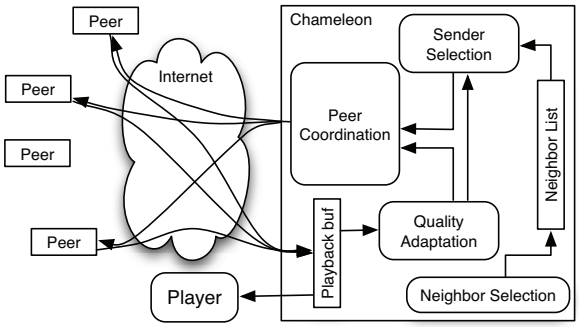


Figure 4.4: Architecture of Chameleon with key components.

used to exchange necessary information among peers. In traditional systems, only one bit is used to represent the availability information of video data. With Chameleon, 2 bits are used to represent the following four meanings (summarized in Table 4.1): (1) the peer has received enough linearly independent blocks to decode the packet; (2) the peer has not received enough linearly independent blocks; (3) the peer has not received enough linearly independent blocks to decode but enough to serve other peers; and (4) the peer does not need to receive the packet (quality adaptation). For a layer, a *ready-to-serve* peer is the peer that has received $\alpha \cdot N$ coded blocks from other peers ($0 < \alpha \leq 1$) for that layer, in which the tunable parameter α is referred to as *aggressiveness* [13]. A peer sends out its buffer map to its neighbors when the status of the buffer map changes. Using one more bit causes slightly more overhead. However, as pointed out for R^2 [13], this overhead is still acceptable and less than traditional protocols, since much larger segments are used with network coding.

Table 4.1: Meanings of the two bits used in buffer maps.

Value	Meanings
00	Do not want this packet
01	The downloading has not completed
10	Ready to serve
11	The downloading has completed

4.4.2 Design Space of Key Components

We now turn our attention to the details of each key component. Different design options for each component are presented and experimentally compared. For the simulation setting used in this section, please refer to Section 4.5.

Class-based vs. Quality-based Neighbor Selection

It is likely that peers with similar capacity should be connected to each other to maximize collaboration potential because they are supposed to receive the same quality. As in some previous studies, a class-based selection method is tried first. Peers can be classified into classes based on the highest quality level they can achieve. We say that a peer belongs to class C when its best possible quality level according to its download capacity is C. A peer connects to other peers in the following priority: peers of the same class, peers of higher classes, and peers of lower classes. If there are more peers than needed, choose randomly. However, as we will see, class-based selection does not work very well. In Chameleon, a quality-based selection method is also considered. In quality-based selection, each peer calculates the average quality level it has perceived so far. When a peer selects a neighbor, it chooses the candidate whose average quality level is closest to its class. If there are more than one peer, it chooses a random one.

To experimentally evaluate the neighbor selection methods, a generic method is proposed as follows. We denote C_i and AQ_i as the class and the average quality level of peer P_i, respectively. When a peer P_k chooses a new neighbor from its candidate list L_k, it chooses peer P_q that satisfies the following condition:

$$|C_k - AQ_q| - \min_{j \in L_k}(|C_k - AQ_j|) \leq \tau$$

If more than one peer satisfies the condition, class-based selection is applied. The above condition is designed to choose peers whose average quality level is closest to the peer class of P_k within the range τ. If τ is equal to 0, we have pure quality-based selection. If τ is equal to the highest quality level of the stream, we have pure class-based selection because all peers in L_k satisfy the condition. Otherwise, we have a hybrid approach. By experimenting with different values of τ, we explore how different neighbor selection methods affect Chameleon. Figure 4.5 plots the skip rates and the average quality satisfaction with τ ranging from 0 to 2.6 (the skip rates and the average quality satisfaction with $\tau \geq 2.6$ are the same). Note that the stream has four quality layers.

Figure 4.5 shows two important insights regarding the selection methods. *First*, quality-based selection is better than class-based selection, and the hybrid approach is the best. The reason is that the quality-based method reflects the peer situation better than the class-based method. A peer who has average quality Q likely belongs to class $\lceil Q \rceil$ or above, while a peer who belongs to class C may not perceive an average quality level up to C, due to the content or bandwidth bottleneck of its neighbors. However, the average quality level only reflects the quality level a peer has experienced so far, and it may be very low compared to the peer class. Consequently, a strict quality-based selection with very small value of τ may "trap" a high ca-

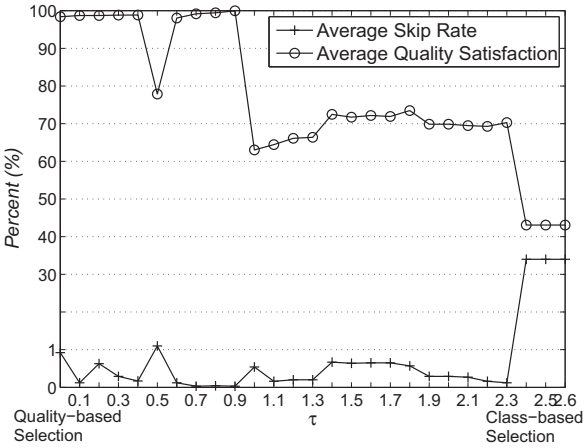


Figure 4.5: The effect of the neighbor selection methods on Chameleon.

pacity peer within an area of low capacity peers. By using a larger τ, high capacity peers that are currently experiencing low quality have opportunities to connect to other high capacity peers, thanks to the class-based selection. Therefore, the hybrid approach offers better performance. *Second*, there is a sweet spot for the value of τ that should be set appropriately to achieve best performance, *e.g.*, $\tau = 0.7, 0.8,$ or 0.9 in Figure 4.5. It should be noted that suitable values of τ may depend on the number of layers of the stream. A hint to determine suitable values of τ in practice, inferred from these experiments, is that the value of τ should be less than $0.3 \cdot NL$, where NL is the number of quality layers to prefer quality-based selection.

Finally, in an unstructured overlay, the topology is formed by the neighbor selection at each peer. There are no global mechanisms to create and maintain the overlay structure. An interesting question here is: *what does the topology look like under the neighbor selection method?* To answer the question, for every peer of class C, we calculate the percentage of its neighbors of class C_k, $k = 1, 2, 3, 4$. In this experiment, the network size is 700, every peer has an average of 50 neighbors. The hybrid neighbor selection with $\tau = 0.7$ is used. Table 4.2 shows the neighboring relationships between peer classes in Chameleon. The value at element (i, j), $T(i, j)$, is the average percentage of peers of class j in the neighbor lists of peers of class i. The value in the parentheses at (i, i) is the percentage of peers of class i in the network. As shown in Table 4.2, the neighbor selection method creates clusters of peers that belong to the same class: $T(i, i) \geq T(i, j), \forall i, j$. This feature is desirable because bandwidth is better utilized when peers with similar bandwidth capacity are connected. The desired feature can be considered as an *emergent property* of Chameleon, because each peer selects its neighbors with only partial knowledge about the network.

Table 4.2: Peer clustering in Chameleon.

Peer class	1	2	3	4
1	**74**(21)	23	2.5	0.5
2	24	**31**(20)	28	17
3	2	26	**48**(24)	24
4	0.5	10	17	**72.5**(35)

Quality Adaptation

In Chameleon, adaptation is mainly based on the current status of the playback buffer. In the first design, the playback buffer is divided into two regions by a threshold drop_threshold. The adaptation process is invoked when the status of the buffer changes, *i.e.*, when the downloading of one segment is finished, or when a segment is played. A peer updates the current quality level, which is the target level for the next segment as follows. If the number of playable segments in the playback buffer (the buffer level) is below drop_threshold, the current quality level is decreased by one. Otherwise it is increased by one, but limited by the highest quality level for that peer's class. The intuition of increasing the quality level here is that it is expected a peer will achieve its best possible quality as fast as possible. However, fluctuations of the perceived quality have been experienced because the number of segments in the buffer may vary around the threshold. To stabilize the perceived quality, another threshold is used, add_threshold. If the number of segments is greater than add_threshold, the current quality level is increased by one. Otherwise, it is unchanged. The adaptation process is shown in Algorithm 1, and the playback buffer with the two thresholds is illustrated in Figure 4.6.

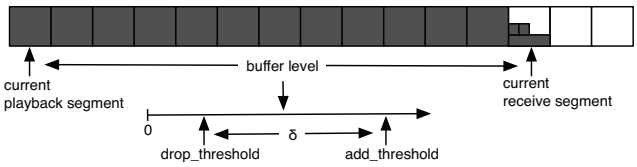


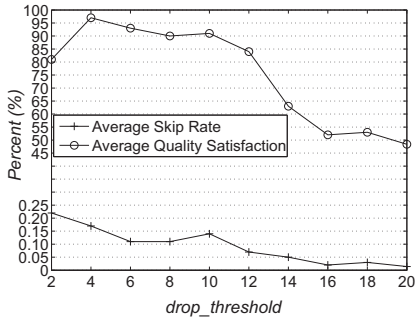
Figure 4.6: The playback buffer in Chameleon: The dark shade indicates the receiving status of each segment.

To understand the effect of the two thresholds on Chameleon, two experiments are carried out with different values of drop_threshold and add_threshold. In both cases, the buffer size is set to 20. *First*, the value of drop_threshold is varied from 2 to 20, and add_threshold is set to (drop_threshold+6). Figure 4.7(a) shows the performance of Chameleon with different values of drop_threshold. We can observe a tradeoff between skip rates and quality satisfaction when increasing drop_threshold: both skip rates and quality satisfaction are reduced. This can be explained as follows. On one hand, a peer may try to maintain/increase the current video quality by using a low drop_threshold. However, the risk is that the skip rate may

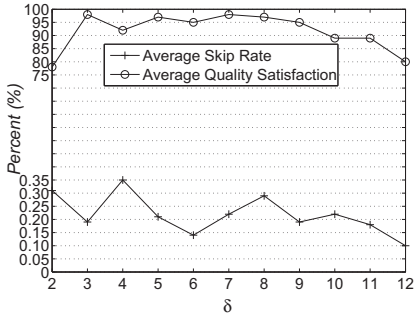
Algorithm 1 Quality Adaptation

```
CL: current quality level.
plb_seg: the segment ID of the current playback segment.
rec_seg: the segment ID of the segment being downloaded.
if (rec_seg − plb_seg < drop_threshold) then
  if (received(QL0, rec_seg)) then
    CL ← CL − 1;
    rec_seg ← rec_seg + 1;
  end if
else if (rec_seg − plb_seg > add_threshold) then
  if (received(CL, rec_seg) ∧ CL < QL_MAX) then
    CL ← CL + 1;
    rec_seg ← rec_seg + 1;
  end if
end if
```

be increased because the playback buffer is exhausted rapidly when the buffer level reaches the low drop_threshold. On the other hand, if the peer is more "*conservative*" by using a higher drop_threshold, it is willing to drop the current layer and moves to the next segment to minimize the skip rate. As a result, the experienced quality may be lower than expected. When drop_threshold $= 20$ (the buffer size), all peers are very conservative, and they receive only the base layer.



(a) The effect of drop_threshold



(b) The effect of add_threshold

Figure 4.7: The effect of the quality adaptation parameters on Chameleon.

Second, drop_threshold is set to 6, and add_threshold $=$drop_threshold$+\delta$, $\delta = 2, ..., 12$. In Figure 4.7(b), it is also observed a tradeoff between the skip rate and the quality satisfaction when δ is greater than 8. Intuitively, if the current buffer level is between the two thresholds, the peer keeps its current quality level. In other words, the larger δ is, the more conservative the peer is. Consequently, the skip rate and quality satisfaction are decreased with a large δ. However, the relation between the two performance metrics is not very clear in the range from 2 to 8. The performance fluctuation can be explained as follows. Since high capacity peers

often fill up the buffer faster than low capacity peers, it is likely that δ should be smaller for high capacity peers so that they can quickly achieve their best possible quality. On the other hand, low capacity peers should use larger δ to keep the skip rate low. However, currently, the same value of δ is used for all peers. The performance could be improved if different values of δ are used appropriately for different peer classes. This feature is left to future work.

To observe the quality adaptation process, the download capacity of a typical peer P (not connected to the server) is varied and its playback graph is examined. The playback graph shows the quality level of all video segments that have been played. The download capacity can be varied slightly ($\pm 10\%$ of the streaming rate of the current quality level) or extremely (to another quality level). C points of time are generated randomly for the download capacity being varied extremely within the period of 10 minutes in the middle of the streaming session. The playback graph of peer P for $C = 15$ is shown in Figure 4.8; it demonstrates that Chameleon adapts to the bandwidth variations well. Minor variations are covered by buffering, while major variations are adapted accordingly. There is only one playback skip, which occurred when the download capacity drops below the streaming rate of the base layer (point 1). The figure also shows that adaptation takes effect few seconds after a significant variation occurs (point 2), *i.e.*, the perceived quality graph is a little shifted to the right of the available download capacity graph at the changing points. This is also because of the buffering effect. When the bandwidth capacity changes, there may be segments in the buffer which have been received before; and the bandwidth variation only takes effect when the buffer level reaches the thresholds.

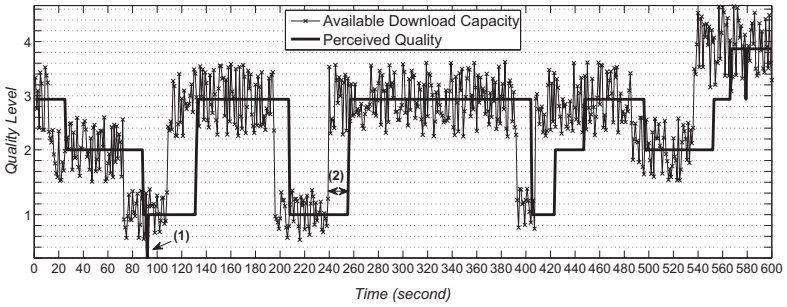


Figure 4.8: An example of the playback graph of a typical peer.

Receiver-Driven Peer Coordination

Chameleon follows a receiver-driven approach to coordinate peers. A peer actively sends requests to senders. However, the requests are sent at the layer level (not at the block/packet level as in traditional approaches). In addition, all senders receive the same requests, and serve the

receiver collaboratively. The receiver does not need to assign packets to each sender separately. The coordination mechanism at the receiver side and the sender side is as follows:

Each receiver:

▷ sends requests for the lowest unavailable layer to all senders.

▷ progressively decodes arrived blocks.

▷ when having received enough linearly independent blocks, sends a stop notification (via buffer maps) to the senders, and finishes the decoding process.

Algorithm 2 Receiver-side

```
PS: list of potential senders.
N: number of NC blocks necessary for decoding.
NumberOfReceivedBlocks ← 0;
L ← getLowestUnavailableLayerID();
newPS ← chooseSenders(PS, L);
sendRequest(newPS, L);
while (NumberOfReceivedBlocks ≤ N − 1) do
  receive(B);
  decode(B);
  if (linearlyIndependent()) then
    NumberOfReceivedBlocks++;
  end if
end while
sendStopNotification(newPS);
```

Each sender:

▷ on receiving a request, performs network coding on available blocks of the requested layer, and sends newly coded blocks to the requesting peers automatically and continuously as soon as possible.

▷ on receiving a stop notification, stops sending.

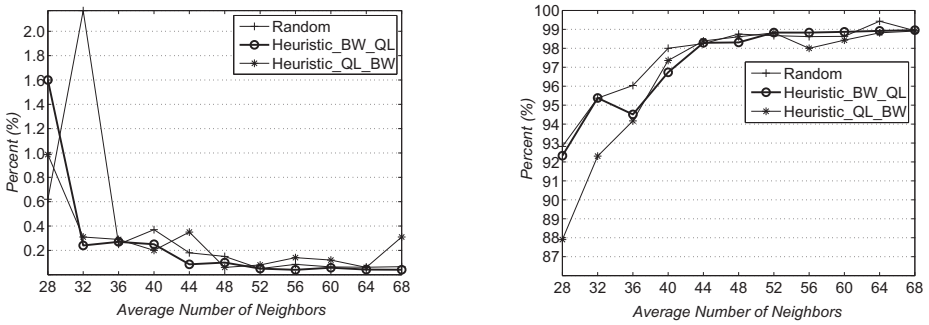
Random-based vs. Heuristic-based Sender Selection

In the above peer coordination mechanism, a peer chooses senders from its neighbors to send layer requests. The first criterion for choosing a sender is that if it can provide coded blocks for the requested layer. This information is available in the buffer maps. If the number of potential peers is greater than the number of connections the peer can create, it needs to choose a subset. The simplest way is to choose a subset randomly. However, choosing senders randomly may

Algorithm 3 Sender-side

```
R: buffer map message.
RL: requested layer (from R).
while (active) do
    receive(R);
    if (isStopNotification(R)) then
        break;
    else
        encodedBlock ← encodeAvailableBlocks(RL);
        sendToReceiver(encodedBlock);
    end if
end while
```

not optimally utilize available bandwidth capacity and layers of the senders. For example, the download capacity of a peer is 150 Kbps. Two potential senders $S1$ and $S2$ who are able to create only one more connection have the upload capacity of 150 Kbps and 200 Kbps, respectively. The *random-based* method may choose $S2$ and render 50 Kbps unused, while choosing $S1$ is obviously a better utilization of bandwidth. A similar rationale is also applied to choose senders based on available layers. Intuitively, the following heuristics are used to choose senders for a peer P: (H1) prefer potential senders whose available upload capacity is closest to the currently available download capacity of P, and (H2) prefer potential senders who have the smallest number of layers.



(a) Average playback skip rate

(b) Average quality satisfaction

Figure 4.9: The performance of Chameleon with different sender selection methods.

To investigate benefits of the heuristics, two heuristic-based methods (Heuristic_BW_QL and Heuristic_QL_BW) are compared with the random-based method (Random). Both Heuristic_BW_QL and Heuristic_QL_BW use the aforementioned heuristic rules but in different order. Heuristic_BW_QL uses the priority order H1, H2, and random; while Heuristic_QL_BW uses H2, H1 and random. We plot the playback skip rate and the average quality satisfaction of the selection methods in Figure 4.9. The average number of neighbors (ν) each peer maintains

is varied from 28 to 68, and the network size is set to 400. It is reasonable that the performance gets better when ν increases because it is more likely that a peer can choose good senders in a big neighbor list than in a small one. The performance is stable when ν is greater than a specific value, *e.g.*, 48. However, it is quite surprising that Chameleon achieves impressive performance with the random-based method. There are no significant differences between the three methods, especially when ν varies from 48 to 64. Heuristic_BW_QL seems to be the best one with respect to both the skip rate and quality satisfaction.

4.5 Performance Evaluation

In this section, the performance of Chameleon is evaluated by comparing it with FABALAM [55], used as a benchmark. Both Chameleon and FABALAM are implemented in our own discrete-event flow-based simulator developed from scratch. The simulator is decribed in detail in Appendix A of this dissertation. The latest JSVM Software, Version 9.17, is used to generate a real two-hour video sequence with four quality levels. The average bit rate of the (sub-)stream with quality level up to 1, 2, 3, and 4 is 620, 825, 945, and 1065 Kbps, respectively.

The main configuration parameters related to the quality scalability used in this paper are presented in Table 4.3. The download and upload capacity of each peer are determined based

Table 4.3: Main configuration parameters used in the simulation.

Configuration File	Parameter	Value
main.cfg	BaseLayerMode	2
	MGSControl	2
	NumLayers	2
layer0.cfg	MGSVectorMode	0
	QP	34
	MeQP0-MeQP5	32
layer1.cfg	MGSVectorMode	1
	MGSVector0	4
	MGSVector1	4
	MGSVector2	8
	QP	30
	MeQP0-MeQP5	30

on the stream rate at different quality levels of the test sequence. This setting is to reveal the benefit of SVC: different bandwidth capacities perceive different quality levels. With the test sequence above, we use four peer classes (corresponding to the four quality levels) in which the download and upload capacity of each peer of class Q are set to $8-12\%$ and $6-10\%$ higher than the stream rate at quality level Q, respectively. Each peer is randomly assigned to a peer class. The server upload capacity is set so that it can serve $8-10\%$ of the total number of peers,

and we use only one server in our experiments. There are no super peers in the system. The following metrics are used to evaluate Chameleon:

▷ *Average playback skip rate*: the average skip rate of all peers in the system. The skip rate of a peer is the percentage of segments the peer skips during playback. This metric is calculated as follows:

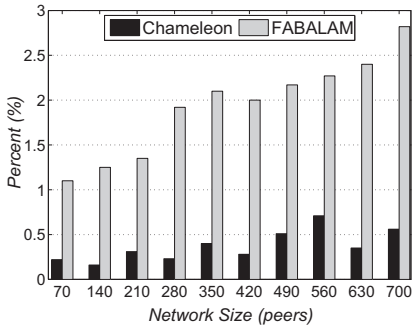
$$SkipRate(\%) = \frac{No\_Skipped\_Segments}{Total\_No\_Segments} \cdot 100$$

▷ *Average quality satisfaction*: the average quality satisfaction of all peers in the system. The quality satisfaction of a peer is the ratio of the average quality level of played segments to its expected quality level (corresponding to its class).

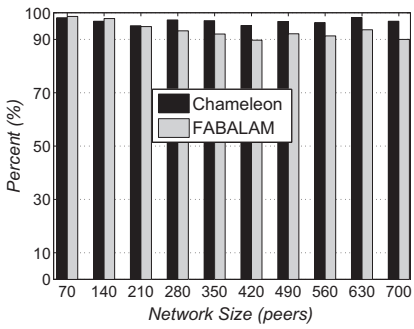
$$QualitySatisfaction(\%) = \frac{AverageQualityLevel}{ClassIdentifier} \cdot 100$$

4.5.1 Scalability

Chameleon is firstly compared with FABALAM on the system scalability in stable environments by varying the number of peers from 70 to 700. Peers join the network randomly, and stay connected until the session ends. Figure 4.10(a) shows that Chameleon achieves very low skip rates: $70 - 80\%$ lower compared to the benchmark and less than or about 0.5% for various network sizes. Regarding quality satisfaction, Chameleon offers very good and stable quality satisfaction when the network size increases. In Figure 4.10(b), the quality satisfaction of Chameleon is always greater than 90% which means that peers can enjoy 90% of the best possible quality according to their download capacity. The system scalability is demonstrated by the stable performance when increasing the network size.



(a) Average playback skip rate

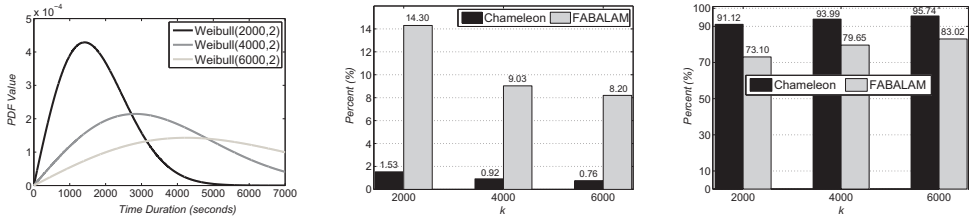


(b) Average quality satisfaction

Figure 4.10: The performance of Chameleon and FABALAM in different network sizes.

4.5.2 Coping with Peer Dynamics

To evaluate the performance of the protocols under peer dynamics, the Weibull distribution — Weibull$(k, 2)$ — is used to randomly generate the lifetime of peers because, as shown in [74], the peer session lengths are best captured by the Weibull distribution. With a two-hour streaming session, we use three different values of $k = 2000, 4000,$ and 6000 to generate different mean lifetimes. The lower the value of k is, the more volatile the session becomes. The plot of each distribution is shown in Figure 4.11 for clarity, together with the skip rate and quality satisfaction of the protocols. In this experiment, the network size is 350.



(a) Weibull distribution for peer lifetime (b) Average playback skip rate (c) Average quality satisfaction

Figure 4.11: The effects of peer dynamics on Chameleon.

Figure 4.11 shows that Chameleon can adapt to peer dynamics well to achieve stable performance, whereas the performance of FABALAM is much impacted by peer dynamics. The reason is that FABALAM suffers the “rarest piece" problem. Without network coding, when a sending peer leaves, the receiver needs to receive exactly the blocks that were assigned to that sender. However, with network coding, the receiver can receive any blocks as long as they are linearly independent with those that have been received so far. In other words, thanks to network coding, Chameleon is robust to peer dynamics. This is demonstrated in the case $k = 2000$ (highly dynamic), the skip rate of Chameleon is only 1.53%, almost ten times lower than that of FABALAM; while the quality satisfaction is still high, up to 91.12%.

4.6 Summary

This chapter presented the design and the performance evaluation of Chameleon, a new adaptive P2P streaming protocol that combines the advantages of network coding and SVC. The objective of this work is to design and preliminarily test a practical adaptive P2P streaming protocol, by taking advantage of network coding and SVC to mitigate the inherent challenges in unstructured layered P2P streaming. Chameleon demonstrates that the combination of network coding and SVC is feasible and beneficial. Network coding helps to simplify the streaming protocol

and improve the system performance. Detailed studies in the design space of Chameleon also bring interesting and useful results in building an adaptive P2P streaming system in practice.

Chapter 5

Overlay Construction Approaches

In Chameleon, the neighbor list of each peer is created and maintained by a SCAMP-based protocol with enhancements for layered P2P streaming. The join time and the class of each peer are generated randomly. Notable performance differences are observed between experiments with significantly different patterns of peer join. The question here is: *How does the join pattern (the join order of peers) affect the system performance with the proposed neighbor selection protocol?* In practice, users can join and leave the system at any time. Therefore, the system performance should not be affected by the join order of users. Answering this question is important to the design of neighbor selection. Experiments with Chameleon reveal that overlay construction for layered P2P streaming should be considered carefully. This chapter's aim is to better understand suitable features of overlays that are specific for layered P2P streaming. Starting from the above questions, experiments with the neighbor selection protocol in two extreme join orders are carried out. Then, a new mechanism is proposed such that the neighbor selection protocol is not affected by different join orders. In addition, the literature demonstrates the advantage of the peer sampling method over SCAMP with respect to bandwidth consumption and robustness to peer churn. Therefore, to more completely consider overlay construction for layered P2P streaming, the application of the peer sampling method in layered P2P streaming is considered with a new peer sampling based membership management.

5.1 SCAMP-based Overlay Construction

5.1.1 Problem Identification

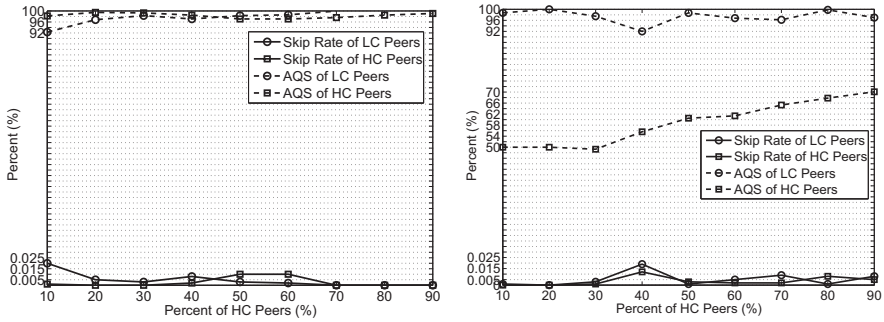
We consider a typical P2P streaming session with a number of dedicated streaming servers, and a large number of peers. Peers participate in and depart from a session in unpredictable ways, and they are heterogeneous with different bandwidth capacities. Peers can be classified into

classes based on their bandwidth capacity. A layered coding technique, e.g. SVC [9], is used to encode raw video data into quality layers. Peers are organized in an unstructured overlay, *i.e.*, there are no global mechanisms to build and maintain the overlay. It is assumed that every peer is willing to contribute its bandwidth to upload data to other peers, *i.e.*, no selfish or fraud peers in the system.

Although this chapter focuses on neighbor selection, a complete streaming protocol is necessary to evaluate the proposed methods. Chameleon with its quality-based neighbor selection is used as a baseline method, which is summarized as follows. When a peer joins the system, or when it needs to update the neighbor list for better quality, it creates neighborships with other peers. A list of available peers can be provided by a rendezvous peer or by exchanging membership information, *e.g.* using SCAMP [60]. Each peer calculates the average quality level it has perceived so far. When a peer selects a neighbor, it will choose the peer(s) whose average quality level is closest to its class identifier within a range τ. If there are more peers than needed, a subset is selected based on the peer class in the following order: peers in the same class, peers in higher classes, and peers in lower classes.

To reveal the effect of the join order and the population percentage of each peer class to the system performance, two extreme cases are considered in the following experiments. Without any loss of generality, the JSVM Software [75] is used to generate a two-hour video sequence with two quality levels, and peers are classified into two classes: high capacity (HC) and low capacity (LC). The download and upload capacity are set so that HC peers are able to receive two quality levels (the full quality) while LC peers are only able to receive one quality level (the base level). In particular, the download and upload capacity of peers are set to $6 - 10\%$ and $4 - 8\%$ higher than the stream rate of the quality level corresponding to each peer class. There are no super peers in the system. Only one server is used, which can serve $8 - 10\%$ of the total number of peers in the system. The performance of Chameleon is evaluated with the quality-based neighbor selection method in Case A: all HC peers join the session before LC peers, and Case B: all HC peers join after LC peers. In each case, the number of HC peers is set to 10, 20, ..., and 90% of the peer population.

Figure 5.1 shows the performance of Chameleon in Case A and Case B. In general, the average skip rates are very low in both cases, but the average quality satisfaction (AQS) is very different. In Figure 5.1(a), when HC peers join the system first, the average quality satisfaction for both classes is very high ($> 92\%$) regardless of the number of HC peers in the system, which means each peer can perceive 92% of its best possible quality level according to its bandwidth capacity. On the other hand, in Figure 5.1(b), when LC peers join first, the average quality satisfaction of HC peers is low and increases from 50% to 70% when the percentage of HC peers increases. The reason is that, in Case B, LC peers join first, connect and stay close to



(a) Case A (b) Case B

Figure 5.1: The performance of Chameleon in the two cases.

the server in terms of the number of hops from the server. Since the number of connections the server can create is limited, when HC peers join, they may be not able to connect directly to the server and content bottlenecks occur. For example, when there is only 10% HC peers in the system, all HC peers receive only the base layer. Consequently, the quality satisfaction is 50% (because they are expected to be able to receive two quality levels). When the population percentage of HC peers increases, the chance of connecting to the server increases, and the top layer can be delivered to some HC peers. In the other case, if HC peers join the session first and connect to the server, they can receive the top quality layer from the server and deliver it to other HC peers. In addition, since they also have the base layer, LC peers are served well.

From the above experiment, it is observed that the join order and the percentage in population of different peer classes may not affect the performance of single-layer P2P streaming systems much, as the skip rates are low in the two cases, but they do impact the average quality satisfaction of different peer classes in layered P2P streaming systems. The question here is: *How to provide high average quality satisfaction for different peer classes regardless of their join time and population*? This question is answered in the next section the proposed scalable and effective neighbor selection method.

5.1.2 A Quality- and Context-Aware Neighbor Selection Method

The difficulty in designing a neighbor selection method is that each peer only knows information of a certain number (not all) of peers in the system. Although it is possible to update information about the population of each peer class in the system by tracking join requests at rendezvous peers, it causes traffic overhead to transmit the information, which changes frequently when peers join and leave. To keep the system scalable, our proposed method is based only on local information of candidates to choose neighbors.

As in single-layer streaming, it is reasonable that high capacity peers should have higher priority than low capacity peers in being located at *good* positions in the overlay, *e.g.*, close to the server or other high capacity peers because when they can receive more, they will contribute more to other peers in terms of bandwidth and layers. Based on this fact and taking the effect of the peer join order into account, a *preemption rule* is proposed as follows: when a peer P in class C_i wants to connect to a peer Q which has reached its maximum number of neighbors defined by the system, if one neighbor K of Q belongs to class C_j, $C_j < C_i$, then P can replace K to be a neighbor of Q. However, if the rule is applied strictly everywhere in the overlay, peers in the lowest class (the lowest bandwidth capacity) can only connect to each other and create clusters of low quality peers, which will suffer high playback skip rates. Therefore, a peer P in class C_i should only be able to replace another peer K in class C_j, $C_i > C_j$, with a probability P_preemp, P_preemp is higher when K is *closer* to the server. This rule guides high capacity peers closer to the server even if they join the system after low capacity peers, and low capacity peers further from the server even if they join the system before high capacity peers. The distance between a peer P and the server is calculated by the minimum number of peers (hops) between P and the server through neighborship with the following algorithm, whose pseudo-code is presented in Algorithm 4:

- ▷ The distance from the server to itself is 0.
- ▷ The distance from a peer P to the server is calculated by the minimum distance of its neighbors to the server $+1$.

  $$D_P = \min_{i \in NL_P} (D_i) + 1$$

  in which D_i is the distance of peer i to the server, and NL_i is the neighbor list of peer i.
- ▷ The distance of a peer is updated when its neighborship changes, *e.g.*, a neighbor is added or deleted.

Algorithm 4 Distance Calculation

```
D: the distance.
NL: the neighbor list.
N: the current number of neighbors.
int_min_dist ← MAX_INT;
for i = 1 to N do
  if (NL[i].distance < int_min_dist) then
    int_min_dist ← NL[i].distance;
  end if
end for
D ← int_min_dist +1;
```

It is noted from Algorithm 4 that it is not required to update the distance of each peer in a timely manner, *e.g.*, when the distance from a peer to the server changes, the distance from its neighbors to the server may also be changed but is not updated. The update algorithm is only invoked at a peer when its neighborship changes. The reason for this *relative* calculation is that the timely update requires message exchanges, which cause traffic overhead, between peers to inform their distance has changed. In addition, as being demonstrated later, the relative distance is good enough to point out the *vicinity* of the peer location in the overlay. When each peer maintains its distance to the server, the preemption probability P_preemp to replace a peer K is calculated by the following formula.

$$P\_preemp = \frac{1}{1 + \alpha(D_K - 1)} * 100$$

in which α is a tunable parameter. From this formula, we can see that if low capacity peers connect directly or stay close to the server (because they join the session first), they are likely replaced by high capacity peers. For example, if a low capacity peer connects directly to the server, *i.e.* its distance is 1, then the $P\_preemp = 100\%$, so it will be replaced by a high capacity peer. However, if low capacity peers are far from the server, they can keep their connections to high quality peers to maintain their quality, as the preemption probability is low. The value of α determines how P_preemp reduces on the way far from the server, *e.g.*, if $\alpha = 1$, P_preemp for $D = 1, 2, 3, 4, ...$ is 100%, 50%, 33.33%, 25%, ... respectively. Currently, values of α are experimentally chosen. However, how to choose a good α in general cases is an important issue and is left for future work.

We are now ready to present the complete neighbor selection method. A peer will create neighborships when it joins the system or when it wants to improve the video quality, *e.g.*, a neighbor leaves the system, or the current quality level drops below a threshold for a period of time. When a peer P needs one or more neighbors, it:

- ▷ contacts a rendezvous peer with necessary information such as its estimated bandwidth capacity. The rendezvous peer will return a list of available peers in the system and the class identifier P belongs to.
- ▷ sends requests containing its class identifier to all candidates, who are selected by the quality-based neighbor selection method in [27].
- ▷ on receiving notifications from the candidates, creates connections and stores neighbor information to the neighbor list, or waits for a period of time and tries again.

For each candidate, on receiving a request:

Algorithm 5 Neighbor Selection - Request

```
AL: the peers list returned by the rendezvous peer.
CL: the candidate list.
NL: the neighbor list.
CL ← QualityBasedSelect(AL);
SendRequest(CL);
while (active) do
  if (accept(Q)) then
    Insert(Q, NL);
  end if
end while
if (too_few_neighbors) then
  Wait();
  Request();
end if
```

▷ if the current number of neighbors is below its maximum number of neighbors, accepts the request.

▷ otherwise, selects the neighbor whose class identifier is lowest, calculates its P_preemp and decides to accept the request with the probability P_preemp.

Algorithm 6 Neighbor Selection - Respond

```
N: the current number of neighbors.
MAX_N: the maximum number of neighbors.
NL: the neighbor list.
ReceiveRequest(P);
if (N < MAX_N) then
  SendReply(P, ACCEPT);
  Insert(P, NL);
else
  R ← SelectLowestCapacityPeer(NL);
  if (P.class > R.class) then
    P_preemp ← 1/(1 + α * (R.distance − 1));
    if (Rand() < P_preemp) then
      SendReply(P, ACCEPT);
      Insert(P, NL);
    else
      SendReply(P, REJECT);
    end if
  end if
end if
```

5.1.3 Evaluation

The proposed neighbor selection method is implemented in Chameleon and evaluated with respect to the average playback skip rate and the average quality satisfaction. *First*, the two extreme cases in Section 5.1.1 are revised with the proposed selection method. *Second*, its efficiency is demonstrated in more general cases with different network sizes to check the system scalability. *Finally*, features of the overlay formed by the method are examined. The bandwidth settings are the same as mentioned in Section 5.1.1.

The proposed method offers adaptability and scalability

Figure 5.2 shows the performance of Chameleon in Case B (the graph for Case A is similar) when LC peers join the session before HC peers. It is clearly observed that the quality satisfaction of HC peers is significantly improved compared to Figure 5.1(b). Thanks to the preemption rule, HC peers can gradually be located in good positions. This not only enables HC peers achieving high quality video, but also minimizes the skip rate of LC peers.

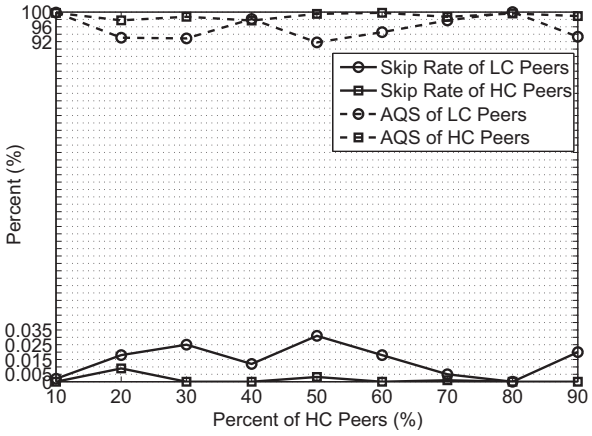


Figure 5.2: The system performance with the proposed protocol in Case B

In practice, peers can join and leave the system at any time. Therefore, a more general case should be evaluated to confirm the advantages of the proposed method. Another video sequence is generated with four quality layers, and four classes of peers corresponding to the four quality layers are used. The join time and the class identifier of each peer is generated randomly, and the peer life time is generated by the Weibull distribution – Weibull(k, 2) – because, as shown in [74], the peer session lengths are fit by the Weibull distribution. Figure 5.3 shows the average skip rate and quality satisfaction of each peer class. It demonstrates that the neighbor selection

method helps to achieve best possible quality for each peer class in the system, while keeping the system scalable under peer dynamics.

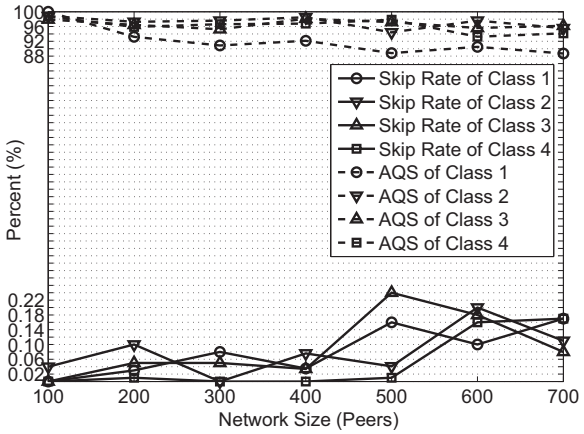


Figure 5.3: Performance of Chameleon with different network sizes

Features of the topology

An interesting question is that: *what does the topology actually look like under the neighbor selection method?* To answer the question, for every peer in class C, we calculate the percentage of its neighbors which belong to class C_k, $k = 1, 2, 3, 4$. In this experiment, the network size is 700, every peer has an average of 55 neighbors. The population percentage of class 1, 2, 3, and 4 in the system is 25.71%, 24.29%, 16.99%, and 33.01%, respectively. Table 5.1 shows the neighboring relationships between the peer classes. The value at element (i, j), $T(i, j)$, is the average percentage of peers of class j in the neighbor list of peers of class i.

Table 5.1: Topology

Peer class	1	2	3	4	Avg. Distance
1	**73.51**	22.42	3.96	0.11	4.14
2	22.22	**54.25**	16.99	6.54	3.47
3	7.07	29.30	**42.42**	21.21	2.73
4	3.29	11.93	12.35	**72.43**	1.92

As shown in Table 5.1, the neighbor selection method creates clusters of peers that belong to the same class: $T(i, i) \geq T(i, j), \forall i, j$, *i.e.*, peers of the same class tend to connect to each other. In addition, the average distance of peers of each class to the server is also calculated. The result is presented in the right most column of Table 5.1, which shows that the higher the class identifier is, the closer to the server the class is. In summary, with the proposed neighbor

selection method, peers are grouped into clusters based on their peer class, and clusters of higher capacity peers stay closer to the server than those of lower capacity peers. This can be considered as an *emergent property* of the method, because each peer selects its neighbors with only partial knowledge about the network.

5.2 Peer Sampling Based Overlay Construction

SCAMP has been demonstrated to be scalable and robust to peer dynamics. However, to use it in layered P2P streaming, the quality-based neighbor selection should be run on top of SCAMP. In addition, the preemption rule is required to gear peers to suitable positions in the overlay regardless of their join time. The distance calculation and the preemption probability cause 'overhead' to the system. The question here is: *Is there another way to build similar overlays with less overhead?* In this section, peer sampling service, another gossip-based membership management protocol, is considered for layered P2P streaming. Based on the peer sampling protocol, a new membership management protocol for layered P2P streaming is proposed. This peer sampling-based membership management protocol is demonstrated to be scalable and can build up overlays with similar features as the overlays that are built by the SCAMP-based protocol.

5.2.1 Protocol Design

This section describes in detail the quality-aware membership management protocol for layered P2P streaming. The protocol is initially based on a generic peer sampling service [61]. Therefore, the first part of this section gives a brief overview of the peer sampling service, while the second part presents the proposed protocol.

Peer Sampling Protocol: An Overview

A peer maintains a local view including up to S (the maximum view size – defined by the system) neighbors. At the beginning, when a peer P joins the system, it contacts a rendezvous peer (a well-known peer) to receive the IP address of a peer Q in the system to start with. Q will send its view to P, and P will consider this view as its initial view. After the above join process, periodically, P selects one neighbor N to exchange its view with. This selection is called *PeerSelection*. After the exchange, P and N have new candidates for updating their view: some neighbors may be removed from, or some candidates may be added to the neighbor list. This step is called *SetView*. It has been demonstrated that this simple gossip-based protocol is scalable and creates overlays with self-organizing and self-healing (self-*) properties [61].

A Quality-aware P2P Membership Management Protocol

To inherit the advantageous features of the peer sampling protocol, the proposed protocol is based on the view exchange mechanism, but with enhancements for quality-awareness. The idea is to differentiate peers in the view exchange by taking peer capacity into account. In particular, by setting different priorities for different neighbors in PeerSelection and SetView of each peer, it is expected that the resulting overlay would have high priority peers at good locations, while low priority peers are located in the remaining part of the overlay. To achieve a certain level of randomness, which is important to the self-* properties [61], peers having the same priority are chosen randomly in PeerSelection and SetView.

In layered P2P streaming, the video stream is encoded into a number of quality layers. The number of quality layers a peer receives depends on its bandwidth. It has been shown that (1) peers with similar capacity should connect to each other to maximize bandwidth and layer utilization [27] (F1), and (2) high capacity peers should be closer to the streaming source than low capacity peers to avoid layer bottlenecks [28] (F2). To achieve the two features, the new membership management protocol works as follows:

- ▷ When a new peer P contacts a rendezvous peer R to join the system, R will send properties of the stream, *e.g.*, number of quality layers (L) and bit rates back to P. It also assigns a class identifier C_i ($0 < i \leq L$) to P based on P's bandwidth capacity and the video properties. Peers of class C_i have higher bandwidth capacity and can receive more quality layers than peers of class C_j, if $i > j$. The server is assigned to the class C_{L+1} (or any value bigger than the highest identifier used for peers).

- ▷ Neighbors in P's view are arranged based on their class identifier. Those who have the same identifier as P are located at the head of the view, then those with higher identifiers in the increasing order, finally those with lower identifiers in the decreasing order. The arrangement is illustrated in Figure 5.4.

- ▷ PeerSelection: a peer Q will be selected from the head of the view. If more than one peer of the same class exist, one is chosen at random.

- ▷ SetView: after a view exchange, new candidates are inserted into the current view of P according to the above arrangement. The view is likely to have more than S peers. Then, S peers are selected from the head of the current view to make the new view. This view update is also carried out at Q's side.

The arrangement of the neighbor list and the ordered selection in PeerSelection and SetView are simple but able to achieve the two preferred features for the overlay. It can be seen from Figure 5.4 that, starting from the head of the list:

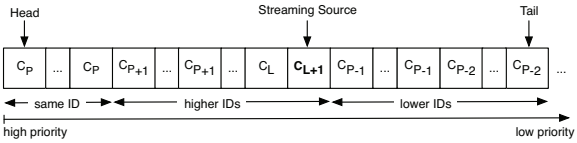


Figure 5.4: The arrangement of neighbors in the neighbor list of peer P. C_P is the class identifier of P

- ▷ F1: P prefers neighbors who belong to the same class.
- ▷ F2: If there are not enough such neighbors, P chooses those with higher class identifiers. Since the streaming source has the highest class identifier, the higher P's class identifier is, the more likely it is that the source will be added to the new view because it is closer to the head of the list. On the other hand, the source is placed towards the tail of the list if P's class identifier is low. Consequently, low capacity peers do not have good chances to connect directly to the source.

However, initial experiments have shown that, if we just take the S first neighbors from the head of the list, peers of the same class are gradually connected to each other and do not have connections with peers of other classes. Consequently, separate clusters are created in the overlay. To keep the overlay connected, each peer should have at most K ($K < S$) neighbors of the same class, and keep at least $S - K$ connections with other classes, which are also chosen based on priority in the list. These $S - K$ connections are particularly helpful for lower classes as they have links to higher classes to receive data. The ratio of K to S is called *clustering ratio* as it determines how well peers of the same class connect to each other.

5.2.2 Evaluation

PeerSim [76] is chosen to implement the proposed membership management protocol because it provides a simple network model and good support for investigation of graph properties of the overlay in both static and dynamic scenarios. This section is started with introduction of the metrics that are used to evaluate the protocol. Then, simulation results are presented and analysed.

Evaluation Metrics

To separately evaluate the mechanism, no complete streaming protocol is used. Rather, it is assumed that a *good* overlay leads to a high performance streaming system. The important question is: *what is a good overlay?* The graph of an overlay is generated by considering peers as nodes and neighbor relationships as edges. Analyzing the graph helps understand how the

protocol behaves and what the overlay looks like over time. From experience with Chameleon [27], the following metrics are particularly interesting:

- ▷ Connectivity: is the graph well connected during streaming? The most basic requirement for the protocol is that no peers are disconnected from the system, *i.e.*, there is always at least one path from the source to any 'alive' peer at any time even with peer churn.
- ▷ Clustering: the clustering coefficient of a group of peers is defined as the ratio of links existing among the peers of the group over the total number of links of those peers. If we consider that peers of the same class are in one group, the clustering coefficient of that group shows how well peers of the class are connected to each other. If the cluster coefficient of a group is 1, the group is disconnected from the network (called separate cluster). On one hand, a high clustering coefficient shows that peers having similar capacity are well connected to each other, which is good in terms of bandwidth utilization. On the other hand, the higher the clustering coefficient, the higher the probability of the class being disconnected from the network, especially under high churn rates. This is a trade-off between connectivity and clustering.
- ▷ Average path length: is the average of the shortest path lengths (number of hops) between any two peers. However, in this work, the average length between peers and the source is more relevant than that between any two peers. Therefore, we will evaluate the average path length of different peer classes to the source. In this context, we call the average path length between peers and the source the *distance* to the source. It is desired that high capacity peers have smaller distances to the source than low capacity peers.

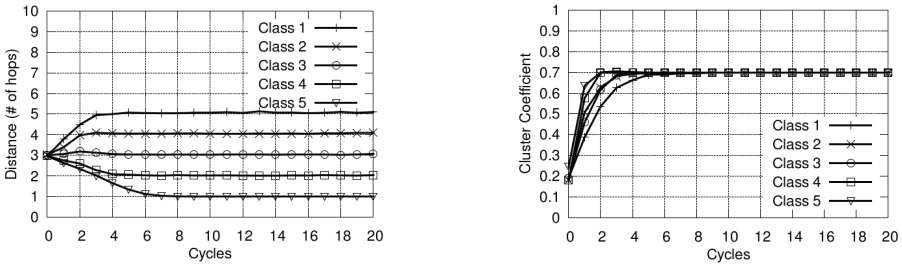
Simulation Results

It is difficult to choose existing quality-aware membership management protocols to compare with the proposed method because none of the previous protocols looks into the graph properties of the overlay during the streaming session to see how they affect the performance. Previous protocols are evaluated indirectly through the overall streaming performance, which is impacted by other components, *e.g.*, peer coordination and streaming algorithms. Convinced that understanding the topology of the overlay and how it evolves over time is important to improve the overall performance of a streaming system, in this paper, the overlay construction is separately evaluated by concentrating on its graph properties. Rather than comparing with a related work of constructing overlays in layered P2P streaming, CYCLON [77] is used, a traditional membership management protocol, as a baseline. The purpose is to emphasize differences and trade-offs (if existing) between quality-aware and non quality-aware protocols.

A layered stream with 5 quality layers is used. Corresponding to the stream, there are 5 peer classes in the system. The number of peers in our experiments is set to 10000 in static scenarios. This is also the maximum number of peers in dynamic networks. Each peer is assigned randomly to one (and only one) class. The view size S is set to 50 as in [27], and the clustering ratio is set to 0.7, which means that each peer has at most 70% neighbors of the same class, and the other 30% are neighbors from other classes. Different values of S and the clustering ratio were also experimented and presented. The cycle-based engine of PeerSim is used. In each cycle, the protocol is run at every peer. After each cycle, properties of the graph are plotted. To evaluate the effect of randomness in our experiments, each experiment is run several times with different random seeds in PeerSim. Since the standard deviations of the metrics are very low, which means that the randomness does not cause instability to the overlay, only the average values of the metrics are plotted.

Static networks

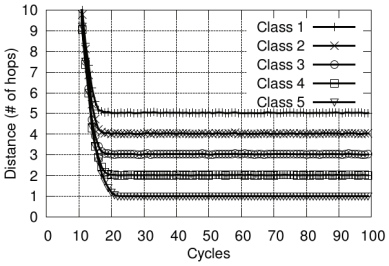
In this scenario, the overlay has a fixed number of peers (10000), and starts with a certain topology. We would like to observe how the protocol behaves and how it constructs the overlay. Two initial topologies used are ring and random topology. Figure 5.5 and Figure 5.6 show the clustering coefficient and the distance of the peer classes for the two cases. Since CYCLON does not take peer capacity into account when choosing neighbors, the metrics have approximate values for different classes (not shown in the figures for clearness).



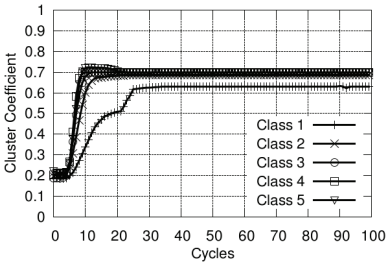
(a) Distance (b) Clustering Factor

Figure 5.5: Overlay evolves from a random topology.

First, the overlay is well connected in all cases (not shown in the figures). *Second*, it can be seen from Figure 5.5 and Figure 5.6 that the overlay can converge to a stable overlay regardless of its initial topology with desired properties: the peer classes have high clustering coefficients (approximate to the clustering ratio), and the higher the class identifier, the smaller the distance



(a) Distance



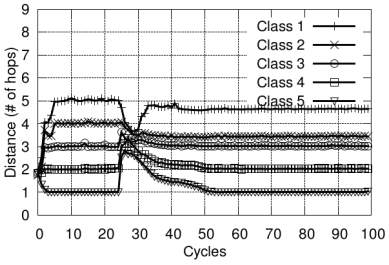
(b) Clustering Factor

Figure 5.6: Overlay evolves from a ring topology.

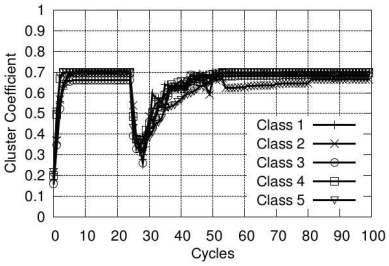
to the source. In particular, at the beginning, the peer classes have similar distances and clustering coefficients. However, when the quality-aware protocol is invoked, the overlay quickly changes itself to differentiate peer classes. In Figure 5.5, it takes only about 6 cycles, which means each peer only exchanges its view 6 times, to achieve stability from a random topology. The ring topology requires more time (about 25 cycles) because each peer has only two neighbors at the beginning (Figure 5.6). However, after converging, the overlay is still able to achieve as good values for the metrics as in the case of the random topology.

Dynamic networks

We now turn our attention to how the protocol behaves under peer churn. In the first experiment, a stable overlay with 200 peers is initialized until the 24^{th} cycle. From the 25^{th} cycle, in each cycle, 200 peers join the system until the size of the overlay reaches 10000. Figure 5.6 shows the clustering coefficient and the distance of the peer classes with peers join.



(a) Distance



(b) Clustering Factor

Figure 5.7: Overlay evolves with peers join.

It can be seen from Figure 5.7 that, at the 25^{th} cycle, the clustering and the distance change rapidly. The fluctuation is because the network size is doubled from 200 to 400 peers (flash crowd scenario). However, after that, the overlay quickly converges to a stable state (in about 25 cycles), even when new peers keep joining the system.

Finally, to evaluate the robustness to peer leaves, peers are removed randomly, and the number of separate clusters are calculated. 50 experiments are carried out. From a stable overlay, 50%, 51%, ..., and 99% of the total number of peers are removed at random. Figure 5.7 shows the number of separate clusters created in the experiments, with the results obtained from similar experiments on CYCLON, a more random-based protocol, to compare their robustness.

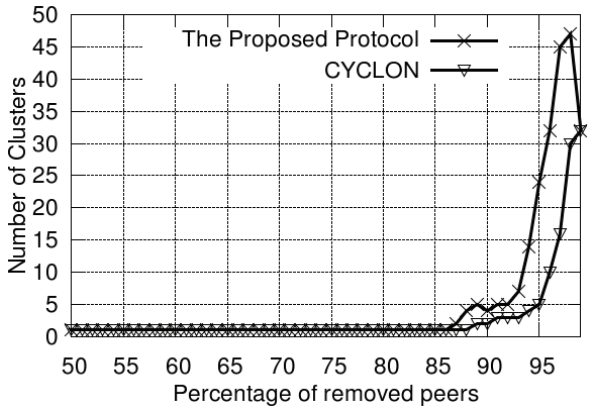


Figure 5.8: Number of clusters generated when peers leave the system

Figure 5.8 shows that the proposed protocol is robust to peer leaves as the first disconnection occurs only when up to 86% of the total number of peers are removed. However, the proposed protocol is less robust than the random-based protocol, whose first disconnection occurs only when at least 88% of peers are removed. In addition, the number of separate clusters (disconnected sub-graphs) created by CYCLON is smaller than our protocol. The reason is that the clustering coefficient of peer classes in the proposed protocol is higher than that of the peer classes in CYCLON. This can be considered as a trade-off between bandwidth utilization and robustness. However, with the robustness up to 86% of the number of peers left the system, there is reason to believe that the proposed protocol is robust enough for P2P streaming.

5.3 Summary

In unstructured P2P streaming, membership management plays an important role in system performance because good neighbors will offer good streaming quality. This chapter reported work on membership management in layered P2P streaming. Both SCAMP-based protocols and peer sampling based protocols were considered. Two quality-aware membership management protocols for layered P2P streaming were proposed. Simulation results have demonstrated that

the protocols boost the streaming protocol and can achieve high performance regardless of peer join orders and peer dynamics. Using the SCAMP-based protocol or the peer sampling-based one depends on particular applications and their implementation. Fair comparison of the two protocols is only valid when they are both implemented in an actual system with real network topology and user dynamics. Due to its simplification and efficient implementation in the simulator, the peer sampling-based membership management protocol is used later on in this work.

Chapter 6

Social Networking Approaches

In addition to the focus on coding techniques and on overlay construction mechanisms, presented in Chapter 4 and Chapter 5, respectively, applications of social networking to improve the robustness of P2P streaming systems with respect to peer dynamics are considered. The intuition is that social networking could help mitigate the limitations caused by the anonymity of traditional P2P systems, *e.g.*, free riding and high churn rates. Different from P2P file-sharing systems, applying social networking to P2P streaming poses unique challenges, *e.g.*, the use of existing social graphs may not be beneficial due to the spare friendship network problem, introduced in Chapter 1. This chapter presents a social-based P2P streaming system, named *Stir*. *First*, the motivation of the idea of spontaneous social networking – forming social networks inside a P2P system – is discussed in Section 6.1. Section 6.2 introduces a typical working scenario of Stir and its key component via schematic and architectural design. Section 6.3 goes into detail of the social-based P2P streaming protocol. An evaluation of Stir is described in Section 6.4 from the phase of data collection and assumptions to the analysis of experimental results.

6.1 The Case for Spontaneous Social Networking

In social networking, the term *homophily* is defined as the tendency of people with similar characteristics to be connected [78]. This homophily principle has been studied and applied in many areas of computer networks. In addition, it has been demonstrated that there is a correlation from social networks to user behavior on the Web [79]. Particularly, people who chat with each other are more likely to share interests. The more time they spend talking, the stronger this relationship is.

What are actual benefits of forming social networks inside P2P streaming? From the above social principles, users who join and stay in a streaming session are likely to have some similar

interests in the stream. Therefore, providing means of communication between users will not only offer more entertaining services, *e.g.* it is more exciting to watch a football game with others than alone, but also create social relations among them. From social communications and activities, user behavior can be predicted, *e.g.*, the more friends a user has in a session, the more she is interested in and the longer she stays in the session. Therefore, if connections between peers are established based on such social relationships, they are more reliable and durable. In other words, reliable users should connect to each other, and as such they will not be seriously affected by departures of those who do not have a strong interest in the session and stay for a short period of time. This naturally minimizes the impact of churn to maintain continuous playback. Friendship is spontaneously formed during a session but lasts longer than one session. Users have their profiles with a friend list. Connections among friends are immediately established when they join another session together. It is believed that the coincidence of joining the same session of friends has a high probability because they have similar interests. Such a streaming system offers personalization as users can show their personality and make friends. In addition, users that stay long in the session are rewarded with stable quality regardless of high churn rate.

What is a suitable social communication means in the scenario? The foremost requirement for a communication means in our system is that it does not consume much bandwidth, which is a critical resource in streaming. Another criterion is that it is able to achieve a certain level of synchronization because live streaming is highly synchronized among users (small time lag). Among many means ranging from non-interactive, *e.g.* emailing, to real time interactive, *e.g.*, voice chatting, IM is probably the most suitable one. Many studies on characteristics of IM [80, 81, 82] have shown that (1) chat messages exchanged among users are usually very short, but (2) they are expressive enough to support a variety of informal communication tasks in a semi-synchronous way.

How about costs of IM servers? IM needs a server to route text messages since each user is identified by a unique *screen name*, not by an IP address. The place to store and update user profiles is also important when the system has thousands, or even millions, of users. Although IM does not consume much bandwidth of each user, the bottleneck is at the server because it needs to handle thousands of connections, *e.g.*, in a large P2P IPTV system. In Stir, rather than putting the tasks to the streaming server, separation of these social services from the streaming server is proposed. Taking advantages of cloud infrastructures (IaaS) and platforms (PaaS) [83], large-scale services can be built and deployed without the need of caring about expensive servers. Google have proposed a framework for Cloud-to-Device Messaging services that allow a third-party application server to send data to its clients via the Google Cloud [84]. Such services are already available for Google Android devices. It is believed that decoupling social services from P2P systems will be a trend in future.

6.2 Stir: Schematic and Architectural Design

Stir is a multi-channel P2P streaming system that provides live streaming content. One example of such a system in reality is P2P-based IPTV. Each Stir user has a profile containing a unique identifier, a password, a friend list, and the current IP address. A user U needs to log in to the system first before watching any channel. There are no privacy concerns in the Stir architecture, as such login information does not have to reveal any personal information — an email address would suffice. The authentication process is controlled by a cloud-based user manager. If U is authenticated, the manager sends the profile back to U and a notification with U's IP address to the streaming server. The streaming server will send the channel list to U. When U selects a channel, the streaming server will send an IP address list of some available peers in the system to U. This step is similar to the joining process in traditional P2P streaming systems. Now, U can create connections with other users to download video data. This initial phase is illustrated in Figure 6.1.

In Stir, friendships are to be established on-the-fly and *spontaneously*, as a group of users watch the same channel. While watching a channel, U can post comments to one of the Stir online forums. This forum is only visible to those who are watching the same channel with U. U can have a private chat (via IM) with another user V if she knows V's identifier, and the chat can be entirely conducted in a web browser, as the channel is being played live. U can add V to her friend list at any time spontaneously. The list of friends constitutes a state that carries over from one session to the next: being in the friend list of U means that U will know the status (if V is in the system or not, which channel V is watching) of V whenever U logs in to the system. The friend list is updated to the user manager when it is changed.

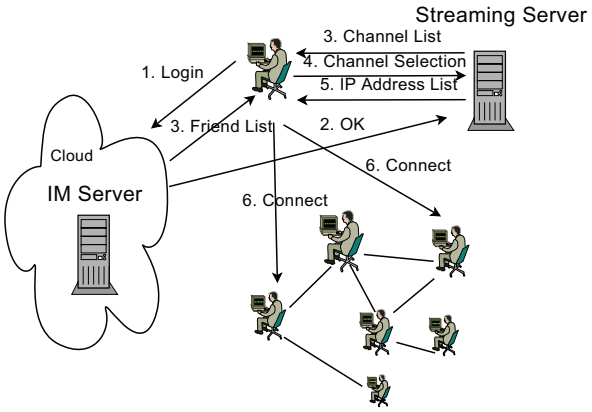


Figure 6.1: Initial steps in the Stir schematic design. After a user logs in, a list of friends may be established spontaneously.

Stir uses a pull-based streaming protocol, which means each peer will pull video data from

other peers based on buffer map exchanges. *e.g.*, similar to CoolStreaming [4]. However, different from traditional P2P streaming which has one list of anonymous peers (called neighbors), each Stir peer has two lists for potential partners. In addition to the neighbor list, which can be updated by a gossip-based mechanism, the friend list containing friends who are also watching the same channel is also used. In addition to an IP address, each item in the lists may contain some statistical data collected from the social activities of users, which are stored in the social log. These statistical data will be used by the partner manager and the scheduler. The partner manager selects potential partners from the lists for requesting data based on data availability in the playback buffer, social factors and network metrics. The scheduler schedules requests of missing data, and sends them to the selected partners. Received video packets are stored in the playback buffer and will be sent to the player when the playback deadline is reached. Figure 6.2 shows the components and their interactions.

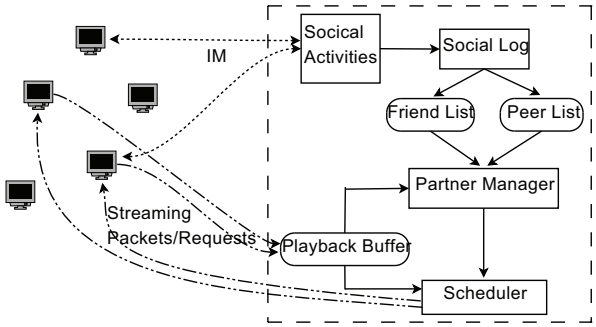


Figure 6.2: The architectural design of Stir, with an emphasis on the tight integration between streaming quality and spontaneous social network relationships.

As shown in Figure 6.2, a user may talk with some people, while video packets are exchanged with other ones. This is a key difference compared to existing studies on social-based P2P, which assume that there are always enough friends in the system to have trusted connections and discourages connections with strangers. In Stir, both social relationships and network metrics are taken into account to choose partners.

6.3 A Social-based P2P Streaming Protocol

With the main components being presented in the previous section, this section goes into design details of the partner manager and the scheduler to understand how the underlying P2P overlay relates to the high level social network, and how social data can be used in streaming. Other components can be inherited with modifications from a traditional pull-based streaming protocol, *e.g.*, CoolStreaming [4], or an adaptive one, *e.g.*, Chameleon [27].

6.3.1 Relying on Friendship or Bandwidth: a Tradeoff

The general idea of a social-based P2P streaming system is that the establishment of network connections between peers is guided by social relationships between users. However, if a protocol design is heavily based on friendship and discounting connections with others, peers may not acquire enough qualified connections to maintain smooth playback, because their friends may not have sufficient bandwidth. The objective is to not only give friends some priority in resource allocation but also achieve smooth playback for the peer itself. Use of a utility function, which takes both social factors and network metrics into account, is proposed as the basis for mitigating the tradeoff.

Social Metrics and Network Metrics

The following three metrics should be considered when a peer P evaluates another peer Q:

- ▷ Network Capacity: includes 'physical' capacity of Q. This kind of metric can include bandwidth capacity, RTT to P, physical distance, etc.
- ▷ Social 'Capacity': captures the 'prestige' of Q in the social network based on its social relations and activities. This can combine several factors: number of friends, number of social messages Q has delivered, etc.
- ▷ Friendship: represents the direct social relation between P and Q. Is Q a friend of P? How often do they chat with each other? How many IM are exchanged between them?

Utility Function

There could be several ways to define a utility function combining all above metrics. This work experiments with the following simple, yet effective, weighted combination:

$$U_Q = (1 - \alpha - \beta) \cdot B(Q) + \beta \cdot S(Q) + \alpha \cdot F(P, Q)$$

where C is a capacity-related function, S is a social-related function, and F is a friendship-related function. β is called *social coefficient*, as it determines how important social capacity is in the evaluation. α is called *friendship coefficient*, as it determines the priority of friendship. In traditional non-social P2P streaming, α and β are 0 as social factors do not exist. In previous social-based P2P systems, α is close to 1 as they discourage connections with strangers. By experimenting with wide ranges of values of α and β, we understand interactions of the social network and the P2P network, and how benefits may be derived from these interactions.

In this paper, the capacity function C, the social function S, and the friendship function F are calculated for Q in a list L as follows:

$$B(Q) = \frac{B_Q}{\max_{i \in L}(B_i)}$$

$$S(Q) = \frac{N_Q}{\max_{i \in L}(N_i)}$$

$$F(P,Q) = \begin{cases} 1, & \text{if P and Q are friends} \\ 0, & \text{otherwise} \end{cases}$$

where B_i is the bandwidth capacity of peer i, N_i is the number of friends of i.

6.3.2 Partner Manager

The interaction between the social network and the overlay network happens in the partner manager, which determines a group of active peers for sending and receiving data. When a Stir peer P selects partners, its partner manager calculates utility values of peers in the neighbor list and the friend list based on the utility function. After that, it sends partner requests to a certain number of peers, which has the highest utility values. If a candidate Q accepts the request, P adds Q to its partner list, which is used for buffer map exchanges and video data requests. The partner acceptance check is also based on the utility function. The algorithms for the selection of partners at P and the acceptance check at Q are presented in Algorithm 7 and Algorithm 8, respectively.

As shown in Algorithm 8, each peer has an acceptance list that contains peers whose partner requests have been accepted. Being in the acceptance list of Q means that P can send data requests to Q and will be served if Q has sufficient resources. Different from the partner selection process that depends fully on the utility value of candidates, the acceptance check has to give friends higher priority than others, regardless of their social capacity or bandwidth. This is a design principle of Stir to encourage people to make friends and share their interests.

6.3.3 Packet Scheduler

In traditional pull-based P2P streaming protocols, buffer maps are exchanged between a peer and its partners to decide who will deliver which packets. However, in Stir, before the buffer map exchange, the peer needs to send a confirmation request to each partner to make sure that it is still in the acceptance list of the partner. After that, buffer maps can be exchanged, and packets can be requested from confirmed partners.

Algorithm 7 Partner Selection at P

```
NL: neighbor list.
FL: friend list.
U: the list of utility values.
N: the maximum number of partners (system parameter).
SL: sorted list in increasing order.
utlCal(X): calculates the utility value of X.
sortOnUtl(X, Y): sorts Y on X and returns the sorted list.
isAccepted(X, Y): sends a request to Y for acceptance check on X.
addPartner(X): adds X to the partner list.
L ← NL ∪ FL;
for i = 1 to L.length do
   U[i] ← utlCal(L[i]);
end for
SL ← sortOnUtl(U, L);
i ← 0;
while (i < SL.length ∧ Q.no_of_partners < N) do
   if (isAccepted(P, SL[i])) then
      addPartner(SL[i]);
   end if
   i ← i + 1;
end while
```

The reason for this step is that the acceptance list of a peer can be changed during the streaming session. For example, at the beginning, a peer is willing to serve non-friend peers because its friends have not joined the session yet. However, when receiving partner requests from friends and the acceptance list is full, it has to remove some non-friends from the list to serve the friends better. As a result of this, the partner list of those non-friend peers is changed, and needs to be updated by invoking the partner selection. This additional action does not cause much overhead because the size of the partner list is small, $5 - 10$ peers. In addition, only the partner list of those who have very few friends and low social capacity may be changed frequently due to their low utility values. Other tasks of the scheduler, *e.g.*, sending requests for urgent packets first, can be done in traditional ways, *e.g.*, CoolStreaming [4]. Thanks to knowledge from the social network, with a set of socially selected partners, it is expected that a traditional pull-based packet scheduler, could achieve significant improvements in streaming quality.

6.4 Stir: Experimental Results

To evaluate our design, Stir is implemented in the discrete-event flow-based simulator. Since user behavior and the friendship establishment process can not be simulated, the focus is on

Algorithm 8 Acceptance Check at Q

```
AL: acceptance list.
AN: the maximum number of accepted peers.
addAcceptedPeer(X): adds X to the acceptance list.
isFriend(X, Y): whether X is a friend of Y.
removePeer(X): remove X from the acceptance list.
if (P ∈ AL) then
   return true;
end if
if (AL.length < AN) then
   addAcceptedPeer(P);
   return true;
else
   min_utl ← ∞;
   for i = 1 to AL.length do
      U ← utlCal(AL[i]);
      if (not isFriend(AL[i], Q) ∧ min_utl > utlCal(AL[i])) then
         min_utl ← U;
         r ← AL[i];
      end if
   end for
   U ← utlCal(P);
   if (isFriend(P) ∨ U > min_utl) then
      removePeer(r);
      addAcceptedPeer(P);
      return true;
   end if
end if
return false;
```

interactions between the social network and the P2P overlay, while making assumptions on the social network formation.

6.4.1 Data Preparation and Assumptions

We need (1) a real-world social graph representing a friendship network, and (2) join and leave times of real-world users.

Social Graph

The authors are not aware of the availability of social graphs formed spontaneously in a particular context as in the case of Stir. As an alternative, a network of people who are interested in a particular topic is used to represent the network of users joining a streaming session. In particular, a "graph data provider" plug-in in NodeXL [85] is developed to retrieve friend lists of members of a group in Flickr. With the URL of any public group, the provider can collect user IDs of members of the group and their friends[1]. From the dataset, a social network for experiments is formed as follows:

- ▷ All group members are considered as users of a streaming session, *i.e.*, a Flickr group is considered as a streaming session.
- ▷ If a member P is in the contact list of another member Q, they are friends of each other in the session. Since we consider relationships among users who join the same session, non-members in the contact lists of the members are removed.

Different datasets of Flickr groups with different sizes are collected to choose one, which has about one thousand members (a medium-large streaming session). It may seem a bit far-fetched to assume that common interest in a topic for pictures in Flickr would correlate with common interest in a live stream in a P2P streaming system. On the other hand, social interaction data from people who are, *e.g.*, watching a stream together were not available to us, and it is therefore believed that a network of common interests and friendship (as opposed to mere friendship connections) is as close as we can get.

The dataset from the group "Photo Computer Art" is selected. When the dataset is collected, this group has 1280 members. The CDF of the degree of the members in the friendship network is shown in Figure 6.3, which indicates that $\sim 10\%$ of members have no friends, $\sim 80\%$ have fewer than 20 friends, and the other 20% have from 20 to 54 friends.

[1]The graph data provider is open source and available upon request. Since the member list of a public group and the friend list of a user are public, this does not violate any privacy rules of Flickr.

Peer Dynamics

From the snapshot trace of PPLive, available at [86], the join times and the leave times of users for a period of time (2 hours) on a particular channel are extracted. Three datasets with different levels of peer churn – low, medium, and high – are used. The CDF of the stay duration of peers in the datasets are shown in Figure 6.4.

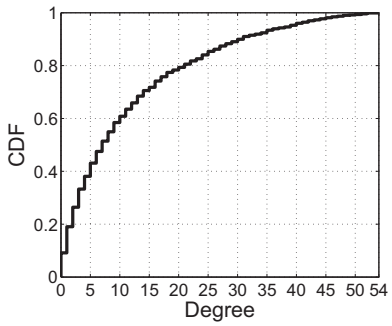


Figure 6.3: Friendships among peers

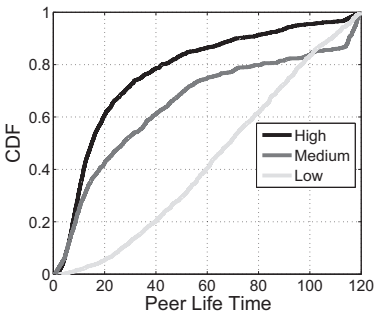


Figure 6.4: Peer Dynamic Scenarios

Figure 6.4 shows that in the highly dynamic scenario, up to 60% of peers stay in the session less than 20 minutes and about 90% stay less than one hour. On the other hand, 60% of the peers stay longer than one hour in the case of low dynamic.

Assumptions on User Behavior

We need to combine the peer dynamic datasets and the social dataset to have a complete picture: who are friends of whom, and when they join and leave the session. Based on the discussion in Section 6.1 about the spontaneous social networking formation, there are reasons to believe that the following assumptions are reasonable: (1) friendship indicates similar interests in the content, and (2) the more friends a user has, the longer she stays in the system. With these assumptions, the datasets can be joined as follows:

- ▷ Sort peers on their stay duration.
- ▷ Sort users on the number of friends they have.
- ▷ Assign join and leave times to users so that peers with longer stay duration have a higher number of friends.

In practice, there could be other reasons for user departures, e.g., network availability problems. Therefore, the above assignment is only valid with another assumption: users are not unexpectedly disconnected from the network, i.e., the arrivals and departures are simply from the interest of users in the session. This helps us understand separately the role of social factors in the system performance.

Bandwidth Settings

The streaming rate is set to 400 Kbps, and the bandwidth capacity of each peer is assigned randomly to one of the following values (download, upload): $(450, 300)$, $(550, 450)$, $(700, 650)$, and $(750, 700)$ Kbps. With 1280 peers, the server upload rate is set to serve 65 ($\sim 5\%$) peers simultaneously.

6.4.2 Comparison with Existing Work

A CoolStreaming-like protocol, described in [4], and a network coding (NC) based protocol, called NCStream, are implemented in the simulation to evaluate their performance in terms of playback skip rates. The packet scheduler of CoolStreaming and Stir are quite similar, except for the change mentioned in Section 6.3.3. On the other hand, the implementation of NCStream is based on the implementation of Chameleon (Chapter 4). The key difference between Stir and the other protocols is the partner manager. While CoolStreaming and NCStream are based only on network metrics to choose partners, Stir takes social relationships into account.

On Peer Dynamics

In this experiment, (α, β) in the partner selection and the acceptance check of Stir are set to $(0.2, 0.3)$, and $(0.7, 0.2)$, respectively. The skip rate of the systems under different peer dynamic scenarios is shown in Figure 6.5.

Figure 6.5 shows that the three systems can achieve similar low skip rates when the network is quite stable (low dynamic). Actually, NCStream is the best protocol in this case because NC helps to utilize the bandwidth better. However, the performance of them are notably different in the medium and high dynamic scenarios. Only $\sim 0.54\%$ of playback segments are skipped by Stir peers in the high dynamic case, while the percentage for NCStream and CoolStreaming is 0.71% and 4.17%, respectively. The reason for the superior performance of Stir under high churn rate is that peers who stay longer in the session are likely to have a certain number of friends and exchange data to each other. In addition, the social-based acceptance check gives friends higher priority in data delivery. Consequently, even when a large number of users who

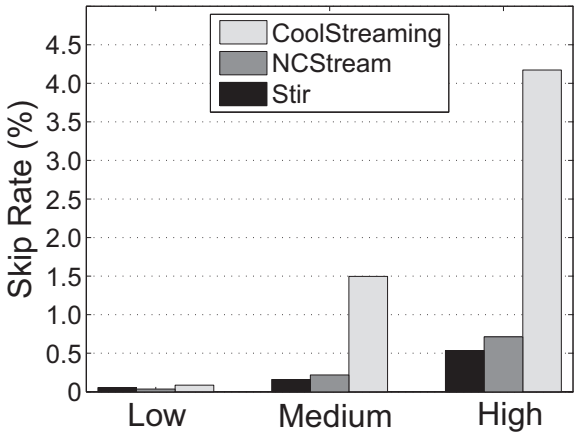


Figure 6.5: Stir minimizes the impact of peer churn

are not interested in the session leave, the communities of friends are not seriously affected. For example, the traffic (number of packets exchanged) between friends and that between non-friends in the protocols are calculated for the high dynamic case: $\sim 60.6\%$ of traffic in Stir is among friends, while, without social knowledge, up to $\sim 65.8\%$ of traffic in CoolStreaming ($\sim 64.2\%$ in NCStream) is between 'strangers'. Although the percentages depend on how dense the friendship network is, this experiment indicates the ability of exploiting social knowledge in Stir.

One may be concerned that the skip rate also depends on the choice of segment sizes in Stir and CoolStreaming, and the NC block size in NCStream. Generally, the larger the segment size is, the higher the skip rate is for CoolStreaming and Stir. As shown in [13], there is a tradeoff in choosing the NC block size. Small block sizes are more robust to peer churn, but cause more overhead for coding coefficients. In the above experiments, we set the segment size to 2 seconds of playback in CoolStreaming and Stir, and the NC block size to 1 KB. In our experiments, with the same segment size Stir always achieves much better performance than CoolStreaming. Since the use of social knowledge and NC can be combined, we expect that the combination even offers better performance.

On The Size of Neighbor Lists

In CoolStreaming and NCStream, the neighbor list is the local view of peers to the network. Therefore, the streaming quality a peer receives completely depends on its neighbors. It has been shown that the size of the neighbor list should be from 50 to 60 regardless of the network size. In Stir, in addition to the neighbor list, a peer has a friend list. It should be noted that peers in the friend list can also appear in the neighbor list because the neighbor list is updated

by gossiping, independently from the social network. Two important questions here are: *how important is the neighbor list in Stir?* and *How big should it be?* The skip rate of the protocols with the size of the neighbor list ranging from 20 to 60 for Stir, and 40 to 80 for CoolStreaming and NCStream are plotted in Figure 6.6. The reason for CoolStreaming and NCStream having larger neighbor lists is to have a 'fair' comparison between them and Stir, because Stir peers may also have a large friend list.

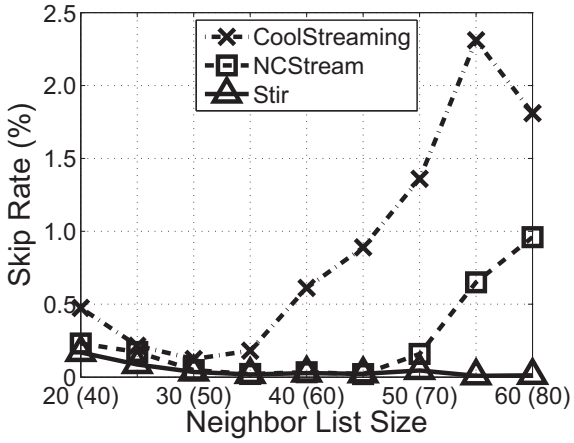


Figure 6.6: Skip rates with different sizes of the neighbor list. The numbers in parentheses denote the size of the neighbor list in CoolStreaming and NCStream.

It can be concluded from Figure 6.6 that (1) using a larger neighbor list, in fact, does not always improve the performance in CoolStreaming and NCStream, and (2) with the existence of the friend list, the size of the neighbor list in Stir can be smaller than in traditional systems. The counter effect of larger neighbor lists in CoolStreaming and NCStream can be explained as follows. The larger the neighbor list is, the higher the probability of more than one peers choosing the same set of high capacity peers, *i.e.*, bottlenecks at high capacity peers are likely to occurs. The problem does not occur in Stir due to the acceptance check that guarantees that a peer 'reserves' resources for its friends.

6.4.3 Insights of Stir

Convinced that by exploiting social knowledge Stir deals with peer churn much better than previous work, we now turn our attention to the insights of Stir. Experiment with different values of α and β in the partner selection process and the acceptance check are carried out to answer the following questions:

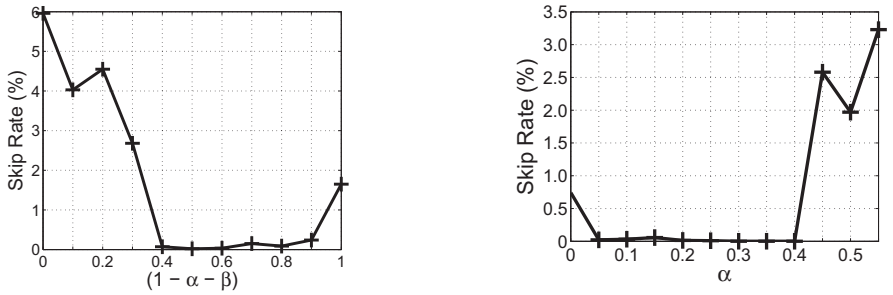
▷ Is the role of friendship, network capacity, and social capacity in choosing partners different or similar?

▷ Between network capacity and social factors (including friendships and social capacity), which one is more important to the system performance?

▷ Between friendship and social capacity, which one is more important to the system performance?

On the Partner Selection

The value of α and that of β are fixed in the acceptance check, while wide ranges of values for the coefficients are used in the partner selection process. Figure 6.7 shows the skip rate of Stir when (1) $\alpha + \beta = 0, 0.1, ..., 1$ to understand the role of the network capacity and the social factors (Figure 6.7a), and (2) $\alpha + \beta = 0.6$ and $\alpha = 0, 0.05, ..., 0.55$ to understand effects of social capacity and friendship in choosing partners (Figure 6.7b).



(a) Network capacity Vs. Social factors

(b) Social capacity Vs. Friendship

Figure 6.7: The effect of α and β in partner selection.

Figure 6.7a indicates that network capacity is important in the partner selection process. If a peer connects to others based heavily on social factors (high values of $\alpha + \beta$), it may suffer playback skips because (1) a high social capacity does not imply a high network capacity, (2) peers with high social capacities have more friends to serve, and (3) its friends may not have sufficient bandwidth. However, a peer should also not depend only on the network capacity (high values of $1 - \alpha - \beta$) because of peer churn: peers with high network capacities may only stay in the session for a short period of time. Very low skip rates can be achieved if peers consider network capacity as important as, or slightly less important than, the social factors ($1 - \alpha - \beta = 0.4$, or 0.5).

Between social capacity and friendship, from Figure 6.7b, we can see that preferring high social capacity peers gives better results than preferring friends for data requests, as high values

of α (> 0.4) increase the skip rate significantly. Although this phenomenon is somewhat contradictory to the idea of connecting friends with each other, it is reasonable from the peers' point of view because the higher its social capacity is, the more durable a peer is. However, friendship has a certain importance as setting α to 0 does not achieve the best performance. The reason is that if a peer chooses its friends as partners, it will have a certain priority at the friend side in their bandwidth allocation. Therefore, if friends of a peer have sufficient bandwidth, the peer will receive higher quality. In addition, being a partner of a high social capacity peer does not guarantee that it will be served. In a nutshell, *network capacity, social capacity and friendship have their own roles in the partner selection process*. However, different from existing work, when a peer chooses partners, friendship is not the only important factor, but bandwidth and social capacity as well.

On the Acceptance Check

Since peers have the highest priority to be in the acceptance list of their friends, the role of α no longer exists in the utility function. For non-friends, the acceptance check is based on their social capacity and network capacity. On one hand, a peer could prefer high social capacity peers (by setting high values for β) to reward them as they are 'famous' in the social network. On the other hand, high network capacity peers could have a high priority for the reason that when they can receive packets quickly, they can deliver them quickly to other peers. However, our experiments with different values of β from 0, 0.1, ... to 1 show no significant differences to the overall system performance (the graph is not shown here). The main reason is that peers that have a certain number of friends are served well by their friends, and so they do not need to be partners of non-friends. For those who have very few friends or no friends, their social capacity is quite similar. Therefore, the role of β is minor. In other words, the case of high social capacity peers compete with low social capacity ones to appear in an acceptance list seldom occurs in our experiments. However, keeping the utility function in the acceptance check is useful in practice, because there are still cases that a peer has a number of friends but some of them may not join the session.

The value of being famous in Stir

It has been demonstrated so far that social knowledge helps improve the overall system performance. However, since each peer communicates with a small number of other peers at a time without global knowledge, one question remains: *What may be the system behavior caused by the protocol executed at each peer?* Especially, it is interesting to know for the entire user population of a session: *Do high social capacity peers generally receive better quality than low*

social capacity ones? Figure 6.8 plots the average quality of peers having the same number of friends.

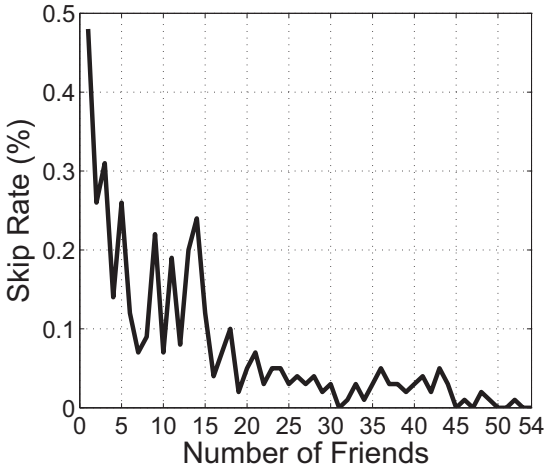


Figure 6.8: In Stir: The more famous you are, the higher the quality you receive.

Figure 6.8 shows that the average skip rates reflect the design objective of Stir: the more friends a user has, the higher the quality she is likely to receive. This encourages users to join the system, and creates a friendly collaborative network among users. In more details, there are two noteworthy points. *First*, the skip rate of those who do not have any friend is less than 0.5%, which is still acceptable in live P2P streaming. This means that Stir does not discriminate against low social capacity peers. They still can receive benefits from the system. In other words, new users who are interested in the session but do not yet have many friends still have chances to enjoy quite a satisfactory level of quality. *Second*, for those who have fewer than 20 friends, higher social capacity does not always bring lower skip rates. When looking back the CDF of the number of friends in Figure 6.3, we can understand the reason. There are many users ($\sim 80\%$ of the population) having fewer than 20 friends. As a result of this, many of those users may not receive enough packets from their friends, and have to compete with others. As shown in the experiments of the acceptance check, the social capacity of those peers does not have a strong effect.

One may concern that a rational peer will make a large number of cheap "spontaneous" friends to improve its quality. As a result, all peers will have a large number of spontaneous friends, and the proposed algorithm can no longer differentiate peers. This is a common issue of social-based applications. For example, Facebook game users may try to make friends with others who they do not really know about to have more friends so that they can get benefits from some game. One possible solution is that some requirements can be used in the friendmaking

process such as friendship can only be formed after two users sometimes chat to each other.

6.5 Social Traffic Costs

In Stir, a certain amount of user bandwidth is consumed by social activities. If social traffic costs are expensive, the overall performance of the system can be seriously affected, due to potentially insufficient bandwidth for video data. Therefore, justification on social traffic costs for using spontaneous social networking inside P2P streaming systems is necessary.

For a user U, social traffic includes: friend list downloading when U joins the system, friend list updating when new friendship is made, instant messaging with friends, and Twitter-like commenting. Important information about a friend in the list includes the user ID, the number of friends, and the IP address. It should be noted that not all friends have their IP address available when U downloads the list from the IM server because some friends have not joined the session (yet). However, the IM server does not need to update the presence of U to her friends because U will later contact them for partnership requests. This 'embedded' status notification is one benefit the streaming protocol brings to the social network because it reduces the cost of status updates for IM.

If we use 20 bytes (20 characters) for a user ID, 4 bytes for the number of friends, and 4 bytes for an IP address, a user with 50 friends that are already in the system will download a friend list of a negligible size — 1.4 KB. When U has a new friend, she only needs to send the user ID of the new friend to the server. During streaming, U can send Twitter-like comments to a forum, as described in Section 6.2, to share with others. Such comments are usually very short as those in Twitter or YouTube. So, the most expensive cost may come from IM among friends.

Xiao *et al.* have shown that the overall IM traffic is about 8.9 Kbps, in which chat messages constitute only a small percentage [80]. However, in Stir, there may be more chat messages exchanged among users because they are watching a real time stream. Although it depends on user characteristics and the content they are watching, let us consider the case of watching an interesting football game to estimate how much traffic IM could cause. As shown in [87], a football game has an average of 300 highlights, including goals, shots, etc. Assuming that a user sends 3 IM to 5 friends at the same time about every highlight during 90 minutes of the game, and each message has 1000 bytes in length (the biggest message length observed in MSN by [80]), she would consume an average rate of ~ 6.67 Kbps for chatting, which is equal to $\sim 1.67\%$ of a 400 Kbps stream. It is believed that the consumption in practice is much smaller than this synthetic case. Due to these reasons, it is convinced that the social traffic costs caused by social networks in Stir are negligible.

6.6 Summary

This chapter presented *Stir*, a new framework towards tightly integrated spontaneous social networking in P2P streaming, as well as a social-based streaming protocol exploiting social relationships of the social network. It has been demonstrated that by offering cheap, yet efficient communication means to users who join the same streaming session, the P2P streaming system is not only able to provide more entertaining services from the perspective of users, but also achieves much better performance compared to previous systems, especially when dealing with high peer dynamics. Simulation with real social network data and real peer dynamic traces demonstrates our approach. It indicates that forming social networks spontaneously on top of P2P overlays would bring significant benefits for both users and the P2P systems. In other words, there are reasons to believe that such social network formation will be a trend in the future, and the study on Stir will shed light on how such a system could be built in practice.

Chapter 7

Chameleon++: Putting It All Together

Chameleon uses NC and SVC to achieve adaptive P2P streaming; and Stir uses social networking to deal with peer dynamics. Both systems are different approaches to the overall goal of offering the best possible video quality to heterogeneous users under network fluctuations and peer dynamics. Now, one important question is: *Can we combine the design of Chameleon and that of Stir to achieve an even better system?* The design of Chameleon focuses on the packet delivery process, *i.e.*, quality adaptation and peer coordination. Meanwhile, exploiting social networking, Stir peers can recognize reliable neighbors to maintain stable quality, *i.e.*, Stir focuses on neighbor management and partner management. Therefore, it is straightforward to infer that the combination is feasible and beneficial as the two aspects are complementary. To confirm the answer and, more importantly, understand how beneficial it is, this chapter presents the design of Chameleon++, which combines the design of Chameleon and that of Stir, and its evaluation by comparing its performance with that of Chameleon.

7.1 The Design of Chameleon++

Since the main components of Chameleon and Stir with their interactions were already presented in detail in Chapter 4 and Chapter 6, this section discusses directly on the architecture of Chameleon++, depicted in Figure 7.1, with the focus on Partner Selection because it integrates Stir and Chameleon. It should be noted that the peer-sampling membership management protocol, presented in Chapter 5, is also used in Chameleon++.

Compared to the architecture of Chameleon in Figure 4.4 and that of Stir in Figure 6.2, Figure 7.1 shows that the combination of Chameleon and Stir is feasible as their components do not fully overlap and do not have contradictory functions. For example, Chameleon++ inherits the neighbor selection component of Chameleon because that of Stir is not quality-aware, and the social-based selection does not have specific requirements for neighbor selection, *i.e.*, the

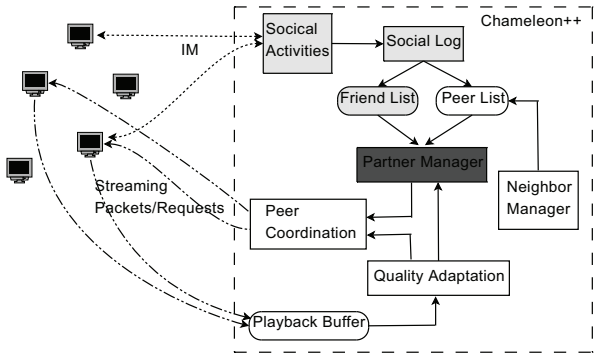


Figure 7.1: The architecture of Chameleon++. Components in light grey color are from Stir. The dark grey color of Partner Manager is to mark that this component is the integration 'point' of Chameleon and Stir

quality-aware neighbor selection does not influence the social-based partner selection. The only interaction point is the Partner Manager, which chooses partners among neighbors to request video packets. In addition to the quality-aware partner selection of Chameleon, Chameleon++ has the social-based selection mechanism of Stir. They need to be reasonably combined to take advantages of both.

On the one hand, the idea of the quality-aware selection is that a peer prefers selecting other peers whose currently perceived quality level is closest to its peer class. Chameleon shows that such selection offers high quality satisfaction as peers receive a quality close to their capacity. On the other hand, the social-based selection takes social factors into account when choosing partners. A peer may prefer one peer with high social capacity over another one with high bandwidth capacity because high social capacity peers are expected to stay longer in the session than low social capacity ones. Maintaining connections with high social capacity peers mitigates the impact of peer dynamic and achieves stable quality. With such features of the two mechanisms, we can ask ourselves: *How to combine them to have a new partner selection method that can not only recognize high social capacity peers but also have connections with peers of similar bandwidth capacity to achieve high and stable quality satisfaction?*, *i.e.*, *How to balance stable quality and high quality satisfaction?* A new utility function, extended from the one used in Stir, is proposed.

In Stir, a utility function is used to evaluate the tradeoff between social capacity and bandwidth capacity. In Chameleon++, the peer class and the current quality level of a peer are added to the utility function to measure its 'value'. The utility function that a peer P uses to calculate

the value of another peer Q is redefined as follows:

$$\begin{cases} U_Q & = \gamma \cdot B(Q) + \beta \cdot S(Q) + \alpha \cdot F(P,Q) \\ \gamma + \beta + \alpha & = 1 \end{cases}$$

in which B is a bandwidth-related function, S is a social-related function, and F is a friendship-related function. α, β, and γ are friendship, social, and bandwidth coefficient, respectively. S and F are defined as in Stir while B is changed to take the peer class of P and the current quality level of Q into account.

$$B(Q) = \frac{CQ_Q}{PC_P}$$
$$S(Q) = \frac{N_Q}{\max_{i \in L}(N_i)}$$
$$F(P,Q) = \begin{cases} 1, & \text{if P and Q are friends} \\ 0, & \text{otherwise} \end{cases}$$

where CQ_i is the quality level i is currently receiving, PC_i is the peer class of i, N_i is the number of friends of i, L is the candidate list. As already noted, the only difference between this function and the one used in Stir is the bandwidth-related function. In Stir, since the stream has only one quality level, its bandwidth-related function is to set high priority for high capacity peers. However, in Chameleon++, peers should have connections with others whose current quality level is approximate to their peer class identifier so that they can receive the best possible quality according to their bandwidth. Such *quality-aware* selection is sufficiently represented by the new bandwidth-related function, which is based on the ratio between the current quality level of Q and the class identifier of P.

With the above utility function, the role of bandwidth capacity and social capacity is identified by tuning the coefficients α, β, and γ as in Stir. The advantages of using the utility function is as follows. Users can specify their own coefficient set for their preference: if they prefer high quality satisfaction and accept a certain level of unstable quality (high values of γ), or if stable quality is the most important criterion (high values of α and β).

7.2 Evaluation

The integrated architecture of Chameleon++ is evaluated by comparing it with Chameleon. *First*, Chameleon++ is evaluated with different values of α, β, and γ to understand effects of the utility function and the role of the new bandwidth-related function when taking quality-aware features into account. *Second*, with suitable values of the coefficients, the performance

of Chameleon++ is compared with that of Chameleon in terms of average playback skip rate and quality satisfaction to demonstrate the feasibility and benefits of using social networking in layered P2P streaming.

Generally, simulation settings are similar to those presented in Chapter 4 for peer bandwidth, SVC streams and NC parameters, and in Chapter 6 for the social data and peer dynamics. In particular:

- ▷ An SVC stream which has 4 quality levels is used. The average bit rate of the (sub-)stream with the quality level up to 1, 2, 3, and 4 is 620, 825, 945, and 1065 Kbps respectively.
- ▷ Four peer classes (corresponding to the four quality levels) are used. The download and upload capacity of each peer of class Q are set to 8-12% and 6-10% higher than the stream rate at quality level Q, respectively. Each peer is randomly assigned to one and only one peer class. The server upload capacity is set so that it can serve 8-10% of the total number of peers, and only one server is used in our experiments. There are no super peers in the system.
- ▷ The social graph of the group "Photo Computer Art" is used.
- ▷ The peer join and leave patterns, extracted from the PPLive traces, are also used.

7.2.1 Quality-aware and Social-based Partner Selection

The role of social capacity and friendship in the partner selection of Stir have been investigated in Chapter 6. Therefore, it is only necessary to consider the role of the new bandwidth-rated function to the system performance. In these experiments, γ is set to $0, 0.1, 0.2, ..., 1$, while α is set slightly higher than β. The system performance is evaluated with different values of γ, and the new bandwidth-related function is compared with the old one. Figure 7.2 plots the average skip rate and the average quality satisfaction of Chameleon++ with different values of γ and with the two bandwidth-related functions.

Figure 7.2 shows two important points. *First*, generally, the bandwidth-related function plays a similar role to the system performance in Chameleon++ as in Stir. Peers should neither heavily depend on the social factors ($\gamma \leq 0.3$) nor on bandwidth ($\gamma > 0.8$). The intermediate values of γ offers high performance. However, while Stir demonstrates that bandwidth should be equal or slightly less important than the social factors ($\gamma = 0.4$ or 0.5) to achieve best performance; in Chameleon++, the best performance can be achieved when the bandwidth-related coefficient is set slightly higher than the social coefficients ($\gamma = 0.6$ and 0.7). *Why should Chameleon++ slightly prefer high bandwidth capacity peers over high social capacity*

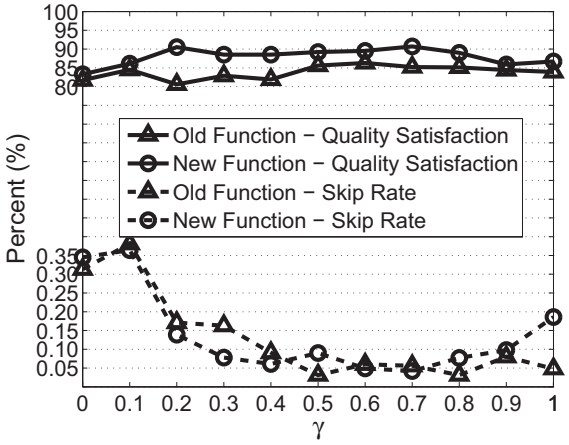


Figure 7.2: The role of the quality-aware selection in Chameleon++

peers? The reason could be the effect of NC. With the perfect collaboration feature of NC, the responsibilities of a particular sender can be easily transferred to other senders if it leaves the system, *i.e.*, NC helps mitigate the impact of peer dynamics at the packet level. Therefore, in a short term at the level of block delivery, high capacity peers can be used to fast distribute coded blocks, while, in a longer term, social factors should still be exploited to have more durable connections.

Second, while both functions achieve similar average skip rates, the new quality-aware function offers better quality satisfaction ($\sim 5\%$ higher). This is a reasonable result as the new function is based on the current quality level each candidate is receiving. Similar to Chameleon, a peer looks for others whose current quality level is higher or equal to its class identifier so that it can receive the best possible quality level according to its bandwidth. Altogether, Figure 7.2 demonstrates the important role of the new partner selection function as it can combine Chameleon and Stir to achieve low skip rates and high quality satisfaction.

7.2.2 Chameleon++ vs. Chameleon

The above results demonstrates two things: the feasibility of the combination as Chameleon++ achieves satisfactory performance (quality satisfaction of $\sim 90\%$, and a skip rate of $\sim 0.05\%$ in the best case), and the important role of the new partner selection method. Now, Chameleon++ is compared with Chameleon on different sizes of the neighbor list and on different peer dynamic patterns.

On the Size of the Neighbor List

Since Chameleon peers do not have friend lists, bigger neighbor lists are used in Chameleon to have a reasonable comparison. The reason to compare the two systems on different neighbor list sizes is that the main difference between them is the existence of friend lists in Chameleon++, which determines different views a peer has about the system. Experimenting on different neighbor list sizes helps understand the impact of social networking on the system performance. Figure 7.3 plots the performance of the systems with the neighbor list size range 20, 30, ..., 70 for Chameleon++, and 40, 50, ..., 90 for Chameleon.

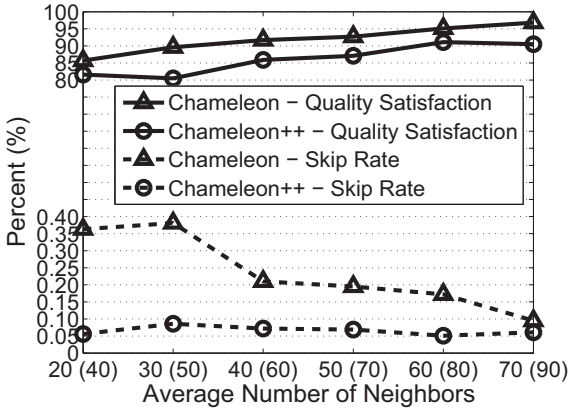


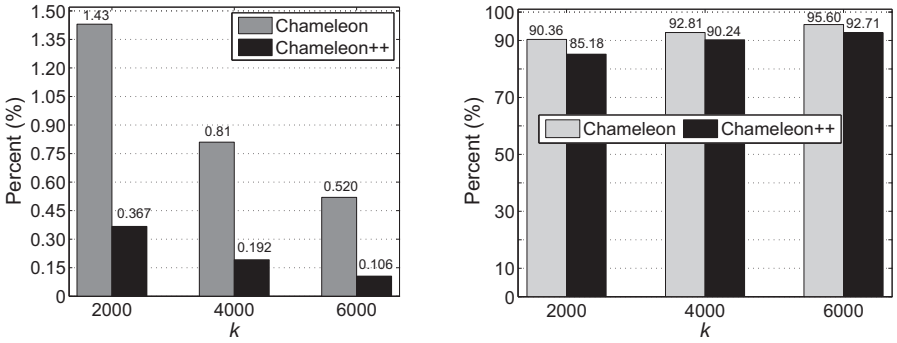
Figure 7.3: Chameleon++ Vs. Chameleon on different neighbor list sizes. The numbers in parentheses denote the size of the neighbor list in Chameleon

As shown in Figure 7.3, generally, the system performance increases when the neighbor list size increases. This is reasonable as peers know better about the network with more neighbors to have better quality. Particularly, the social-based system achieves stable and very low skip rates ($< 0.1\%$), compared to those of the non-social system (up to $\sim 0.38\%$). This result is consistent with the result of Stir, presented in Chapter 6, as very low skip rates can be achieved when exploiting social networking. However, the quality satisfaction of Chameleon++ ($\sim 80 - 91\%$) is lower than that of Chameleon ($\sim 86 - 97\%$). The reason is that high social capacity peers may not have high bandwidth capacity. Consequently, they offer continuous playbacks (low skip rate) but low quality satisfaction.

On Peer Dynamics

To have different distinguishable peer dynamic patterns, the Weibull distribution — Weibull$(k, 2)$ — is used to randomly generate the lifetime of peers, as in Chapter 4. With a two-hour streaming session, three different values of k, $k = 2000, 4000,$ and 6000, are used to generate different

mean lifetimes. The lower the value of k is, the more volatile the session becomes. The neighbor list size in Chameleon++ and Chameleon is 70 and 90, respectively. The performance of Chameleon and Chameleon++ is plotted in Figure 7.4. Figure 7.4 demonstrates that although



(a) Average Skip Rate (b) Quality Satisfaction

Figure 7.4: Chameleon++ Vs. Chameleon on coping with peer dynamics.

NC helps mitigate the impact of peer dynamics, the use of social networking even achieves better skip rates. Again, the tradeoff between skip rate and quality satisfaction exists in this experiment as Chameleon++ offers lower quality satisfaction while Chameleon achieves higher quality satisfaction.

7.2.3 Skip Rate - Quality Satisfaction Tradeoff: Is it worth to combine Chameleon and Stir?

To answer the question, *first*, between skip rate and quality satisfaction, *which one is more important?* Generally, low skip rates means that users can watch the stream smoothly, while quality satisfaction is about the quality of images. It is hard to answer this question convincingly as the overall visual quality from the user perspective depends on the content of the video and user preferences. For example, if a football match is streamed, users would prefer low skip rates with acceptable quality than high image quality with stop-and-play behavior. On the other hand, if a documentary video is streamed, high image quality could be more preferable when the average skip rate is acceptably high. The effect of the metrics to the user experience is not within the scope of this study. However, it is generally believed that when both systems achieves the quality satisfaction within an adequate range, *e.g.* $\geq 80\%$, the overall performance should be based on the skip rate as the difference in quality is not as notable as the occurrence of playback skips. *Second*, *Are social activities important?* If *Yes*, then the integration of Stir and Chameleon is necessary because Stir offers unique social features for P2P streaming services: efficient social activities and personalization.

7.3 Summary

This chapter presented the design and the evaluation of Chameleon++, the combination of Chameleon and Stir. The design of Chameleon++ is mainly based on the design of Chameleon and Stir; the evaluation has confirmed that the combination of social networking, network coding, and SVC is feasible and beneficial.

Chapter 8

Concluding Remarks

Solutions to adaptive and robust video P2P streaming systems have been presented thorough this dissertation. This final chapter, *first*, self assesses the work by revising the research questions that are raised in Chapter 1. *Second*, technical contributions are summarized with emphasis on how they would affect research on P2P streaming. *Finally*, future research directions are suggested.

8.1 Self-Assessment

The main research question of this work is: *How to make a large-scale P2P streaming system adaptable and robust so that it can offer best possible experience to heterogeneous users under highly dynamic network conditions while maintaining its efficiency and scalability?* This question is answered with the novel adaptive P2P streaming protocol, Chameleon, and the robust social-based P2P streaming protocol, Stir. Altogether, Chameleon++ demonstrates that the integration of SVC, NC, and social networking is the key to achieve adaptive, robust, and personalized P2P streaming services.

In particular, Chameleon answers the questions about using SVC in P2P streaming. With the proposed segmentation method, SVC can be used in P2P streaming. However, the layering feature does limit the collaboration potential among peers. Thanks to NC, the use of SVC in P2P streaming is feasible and beneficial as the combination of SVC and NC helps achieve adaptability to user heterogeneity and network fluctuations.

In Stir, the problems of social network formation and exploitation are approached. The social networks that are formed spontaneously during a P2P streaming session are tightly integrated with the underlying P2P overlay because social relationships are based on users' interest in the content of the stream. In such social networks, users' interest is revealed through their

social relations, which are exploited efficiently by the underlying P2P streaming protocol to deal with peer dynamics.

In a nutshell, this research achieves the overall goal as solutions to the defined problems are found. The feasibility of adaptive and robust P2P streaming is demonstrated. However, there are always questions about limitations of a research work. While *pros and cons* of a particular proposed solution were discussed in its respective chapter, one important question remains: *To what degree are the findings valid?*

- ▷ Simulation results are only indicative. With the time constraint, simulation was chosen to be the main evaluation method. No matter how realistic simulation settings are, there is always a gap between simulation and the real world. The simulation results in this work are useful and valid for comparing the performance of different systems with similar settings.
- ▷ Selection of tunable parameters in the proposed protocols follows the 'trial and error' approach. Tunable parameters are helpful in understanding the system behavior and evaluating the system. However, although a wide range of values for each parameter are tested, suitable values in simulation may need to be adjusted when the system is deployed in the real world. With respect to this limitation, simulation is useful in the way that heuristics for choosing suitable values are found.
- ▷ Hypotheses and assumptions can limit the application of the proposed systems in practice. Some hypotheses and assumptions are made to simplify the problems to focus on key features of the solutions. The system performance may be different when the assumptions are fully relaxed or the hypotheses are not valid. For example, in Stir, it is assumed that users who are interested much in the content of a streaming session stay longer in the session and have more friends than those who are not interested in it. In practice, this assumption is not always true as a user is really interested in a session but she may have to leave the session because of unexpected reasons.

8.2 Technical Contributions

8.2.1 Chameleon: Adaptive P2P Streaming with NC and SVC

Chameleon demonstrates the feasibility of combining SVC and NC to provide adaptive P2P streaming services. The design of Chameleon would serve as a sample design of unstructured layered P2P streaming systems with its key components and their interactions. The adaptability would change the behavior of P2P streaming systems. Chameleon users can receive the best

possible video quality according to their available bandwidth capacity rather than receiving the same quality. Rather than suffering playback skips when bandwidth drops, they can perceive reduced quality but continuous playback. These features would improve user satisfaction in using P2P streaming services. Other work could consider Chameleon as a baseline to investigate more problems and bring adaptive P2P streaming to the real world.

8.2.2 Quality-aware Membership Management for Layered P2P Streaming

The study on membership management for layered P2P streaming contributes to the research area in three ways. *First*, it is demonstrated that traditional membership management protocols are not sufficient for layered P2P streaming, *e.g.*, high capacity peers may receive low video quality because they are surrounded by low capacity peers. In addition to P2P streaming algorithms, this study would focus more attention on overlay construction for layered P2P streaming. *Second*, the proposed protocols show that by taking peer capacity into account, a gossip-based protocol is good enough to boost an adaptive P2P streaming protocol. *Third*, there is a tradeoff between the robustness to peer dynamics and the quality-awareness of the overlay in layered P2P streaming. To be quality-aware, the peer selection should prefer neighbors with similar or higher capacity. On the other hand, using more criteria to choose neighbors, the set of potential peers for neighboring is smaller. Consequently, separate clusters may be created, and the overlay is more easily disconnected when peers leave. Other work could inherit the findings about the good features of quality-aware overlays and the current version of the protocols to build a more practical protocol when taking other network metrics into account.

8.2.3 Stir: Spontaneous Social P2P Streaming

Stir is the first attempt to apply social networking in P2P streaming. Allowing social activities during streaming sessions, Stir offers more entertaining streaming services, *e.g.*, users can communicate and share opinions directly about the content they are watching together. In addition, with Stir, faithful users who have spent time in the system and have many friends are rewarded with a high priority in receiving satisfactory streaming quality. Such personalized services are important in attracting users and have not existed in current P2P streaming systems. With the proposed approaches, differentiated QoS and personalized services, which have only been available in client-server streaming systems, are now possible in P2P streaming. Others could learn from Stir: *How does the idea of spontaneous social networking work?* and *How could it be exploited efficiently in P2P streaming systems?*

8.3 Future Directions

For the period of 3 years, the goal was to investigate technical problems of building up adaptive and robust P2P streaming systems. With the work that has been done, it is now clear that the use of SVC, NC, and social networking to achieve adaptive and robust P2P streaming is feasible. However, as new ideas are investigated and the work is simulation-based, there are still many concerns to be studied. In the following, main future directions that could be of interest are listed.

- ▷ With the time constraint, simulation may be the most reasonable approach to examine the key features of the proposed systems. However, simulation results are indicative, *i.e.*, valid for comparisons between systems with similar simulation settings. The practicality of the systems can only be evaluated by real implementations. Convinced that the approaches are practical, implementing real prototypes, and deploying them on test beds, *e.g.*, PlanetLab [88] could be the next step. Evaluation of the real prototypes on test beds guarantees the practicality of the systems in practice.

- ▷ Currently, the capacity of peers is defined by, and only by, bandwidth capacity. When the proposed protocols are used in practice, the definition of peer capacity should be extended. Some other network metrics, *e.g.*, RTT, physical distance among peers, or ISP domains, should also be considered to better represent peer capacity. However, the findings about key system features of the work are still valuable. For example, in the proposed membership management protocols, although only peer bandwidth capacity is taken into account when selecting neighbors, the work demonstrates that peer capacity should be considered in neighbor selection for layered P2P streaming rather than simply selecting randomly or depending on time-to-live metrics in traditional P2P streaming. In addition, regardless of how many metrics are added to the capacity measurement, the principles of constructing layered P2P streaming overlays, discovered in this work, should be kept: peers with similar capacity should connect with each other but a certain number of connections should be reserved for connections with peers with different capacities; and high capacity peers have a good chance to be located at good positions in the overlay.

- ▷ Similarly, the definition of social capacity, in Stir, should be extended to include practical information when it is used in practice. Generally, the social capacity is to measure the interest of users in the content type. Therefore, for a Stir user, the number of social messages, including instant messages, and the communicating frequency may also be valuable to consider. In addition, extending the definition may lead to different utility functions. The current utility function is good enough to demonstrate that social factors play an important role to the system performance. However, the effect of the utility

function is sensitive to the choice of its coefficients, α and β. More effort should be spent on more efficient and practical utility functions so that the system works well under a wide range of network and user conditions.

Appendix A

The Simulator

Discrete-event simulation is the most widely used simulation model in which the system is modeled as a series of events, that is, instants in time when a state-change occurs [89]. In computer network simulation, discrete-event simulators can be classified into two categories: packet-level (packet-based) simulators and flow-level (flow-based) simulators [90]. Packet-level simulation uses events to model the movement of each packet in the network, *e.g.*, packet arrivals or departures at network devices or buffers. Flow-based simulation considers data transmission at the flow level, *i.e.*, events are only generated when the rate of a flow changes. Compared to the packet-based model, if the flow rate changes infrequently, the flow-based approach dramatically reduces memory and computational load because the packet-based approach may issue a huge number of events presenting traffic of all packets. In P2P systems, as connections among peers change frequently, the packet-based model simulates the operation of the system more accurately than the flow-based approach. Therefore, packet-level discrete-event simulation is used in this work.

This appendix, *first*, summarizes available simulators to answer the question: *Why was a new simulator developed for this research work? Second*, it presents the packet-level discrete-event simulator that was developed from scratch for this research with the proposed and more efficient implementation of the bandwidth allocation.

A.1 Available Simulators

The authors in [91] present a survey on currently available simulators. They compare nine simulators under the following criteria: architecture, usability, scalability, statistics and underlying network, and show that no simulator meets all their requirements. Their quantitative survey on the use of simulators also supports the fact that people tend to develop their own simulator to demonstrate their work instead of using an available common purpose simulator. The reason

is that people do not find other simulators suitable for their specific requirements, and it often consumes more time to modify an existing simulator than to build a new one as most of the available simulators are not well-documented. Some notable simulators that the author is aware of at the time the project was started are summarized and discussed as follows to explain why none of them is suitable for this research.

Ns-2 [92] is a well-known packet-based discrete-event simulator, which provides a very detailed packet-level simulation model of the underlying network. However, due to its detailed model, only a few hundreds of nodes may be properly simulated in a reasonable period of time. Therefore, it is not able to evaluate a large-scale systems with thousands of nodes. In addition, building a complete streaming protocol in ns-2 is time consuming as the developing framework of ns-2 is not well-organized. Consequently, although ns-2 is popular in general network simulation, the author is not aware of any study using ns-2 to simulate a P2P streaming system.

In order to simulate large scale P2P networks in a reasonable period of time, the authors in [93] introduce Narses, a flow-based simulator. They simulate the data distribution at the flow level, *i.e.*, neglecting transmissions of single packets but focusing on events of the start and end of a transmission. Since it is required to consider the delivery of each packet in our systems, flow-based simulators in general and Narses in particular are not usable.

PeerSim is another notable P2P simulator, developed in Java language [76]. It is composed of two simulation engines: a cycle-based one and an event-driven one. The cycle-based engine uses some simplifying assumptions, *e.g.*, ignoring the details of the transport layer in the communication protocol stack. In each cycle, the 'application' at every peer is invoked. Since the operation of each peer in a P2P streaming protocol is independent, the cycle-based engine can not be used to simulate a complete P2P streaming protocol. The event-based engine is more realistic as it supports transport layer simulation as well. Unfortunately, at the time this project is started, only the cycle-based engine of PeerSim was documented. Therefore, PeerSim could not be used as the main simulator in this project. However, since the cycle-based simulator provides a simple network model and good support for investigation of graph properties of the overlay in both static and dynamic scenarios, it is used to study different overlay construction approaches.

OverSim [94] is an open-source overlay and P2P network simulation framework for OMNeT++ [95]. OMNeT++ in general and OverSim in particular are more and more mature with more efficient and accurate network communication models. However, at its initial stage, OverSim provides very simple models for structured overlays, *e.g.*, Chord, Kademlia, and Pastry. Since this work is about unstructured overlays, OverSim was not considered as a practical simulator to evaluate the proposed protocols.

From the above discussion, it is clear that building a new simulator, specified for this work, may be challenging at the beginning, but was beneficial later on as specific models and assumptions could be made. In the following sections, the simulator is described in detail.

A.2 Packet-level Discrete-event Simulator

Following the discrete-event simulation model, a simulator is designed with the following components: event controller, event executor, and application. The event controller receives events from the application and sends those events to the executor at their scheduled time. Inside the controller, events are stored in an event queue. The controller manages the queue by inserting events into the queue, taking events from the queue, and updating the time line of the queue. The executor receives events from the event controller and updates the status of the application accordingly. During the execution of an event, new events can be issued, *e.g.* a send event leads to a receive event, and transferred to the controller. The architecture of the simulator is depicted in Figure A.1. It should be noted that both the executor and the application define the functionality and the behavior of the simulated system. However, instead of considering them as one big component, both components are used as the executor is responsible for event-related operations, *e.g.* handling packet delivery, while the application is used to handle non-event tasks, *e.g.*, handling file I/O, inserting initial events, and collecting statistical data. C++ is used to imple-

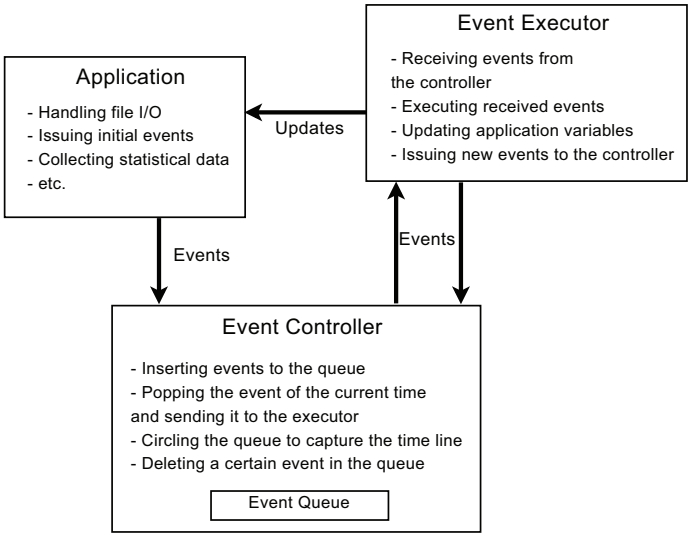


Figure A.1: The components of the simulator with their main functions

ment the simulator. The most important data structure in the simulator is the event queue. Since a huge number of events are inserted and popped during simulation, a suitable data structure

is critical to the performance of the simulator. A linked list is the most suitable data structure as the number of events is unknown beforehand. However, one weakness of linked lists is the accessing cost, *i.e.*, the time it takes to access the i^th item in the list is in the order of $O(i)$. To mitigate the weakness, we use a circle array of linked lists to store events as follows. A circle array of N items is used to keep events whose scheduled time is from the current time t to $t + \delta$. The period of time δ is divided into N equal slots. Events are inserted in suitable slots based on their scheduled time. One linked list is used for one slot to store events whose scheduled time falls into that slot. Such a data structure offers efficient event storage and access:

- ▷ To insert an event: from the scheduled time of the event, it takes (1) $O(N)$ to identify the time slot for the event in the array ($N << $ the total number of events in the queue), (2) $O(k)$ to traverse maximum k events in the linked list of the time slot.
- ▷ To pop an event: the first event in the linked list of the slot of the current time is returned.

The values of δ and N are important to the operation and the performance of the simulator. Since the queue only keeps events that occurred within δ time units (seconds), small values of δ may not capture all occurred events while big values of δ increase access time and storage. To determine suitable values for δ and N, we examine available events that are used for our P2P systems to determine the larger time period of the occurrence of the events. For our P2P protocols whose events will be presented later, we use $\delta = 600$ seconds, and each time slot is equal to 0.5 second ($N = 1200$). With those values, we have never received an 'out-of-range' exception.

A.3 P2P Protocols

To simulate a system with the discrete-event simulator, we need to model its operation by events. Although a P2P streaming protocol is not described in detail in this chapter, typical events of P2P streaming protocols are presented to illustrate how such protocols are simulated.

- ▷ Playback event: when the playback deadline is reached, each peer has to check its playback buffer. If the current playback segment is available in the buffer, it is played back. Otherwise, a playback skip occurs. In live P2P streaming, all peer receive playback events at the same time for synchronization. In VoD (Video-on-Demand) P2P streaming, each peer may receive playback events asynchronously according to their join time. Since we are considering live P2P streaming protocols, the first playback event is issued by the application when the simulation starts. After that, when handling the first playback event at the time t, the executor updates the playback status based on the playback buffer, *e.g.*,

increasing the number of playbacks by one if the segment is playable, otherwise, increasing the number of skips. Finally, the executor inserts a new playback event, scheduled at $t + pb\_period$, to the event controller. pb_period is the playback period defined by the system (1s is currently used). By inserting another event at the end of the execution of one event, playback occurs periodically at every pb_period.

▷ Request event: this event type is issued by the executor for a peer to request packets from other peers when the peer joins the system or when it receives a packet.

▷ Send event: a send event is issued when a packet is sent from one peer to another. When handling a send event, the executor establishes a connection between the sender and the receiver, *e.g.*, by calculating the available bandwidth of the connection. Based on the available bandwidth, the transmission time is calculated. Finally, a receive event whose scheduled time is equal to the current time plus the transmission time is sent to the controller.

▷ Receive event: when a receive event occurs, the executor updates the playback buffer of the receiver, terminates the connection, and notifies the sender that the packet has been received.

▷ Membership/partnership event: periodically, a peer needs to update its neighbor/partner list, *e.g.*, by gossiping. The membership/partnership management mechanism is invoked by the executor when it receives a membership/partnership event.

▷ Join event: the application issues a join event for one peer when its join time is reached (equal to the current simulation time). Peer join time is stored in an input file (trace file) and read by the application in its initiation. The executor allocates memory for variables used by the peer, *e.g.*, playback buffer, and activates the peer by inserting the first membership, partnership, request events for this peer.

▷ Leave event: when a peer leaves the network, a leave event is issued. It should be noted that issuing a leave event for a peer does not mean that the peer sends a leave notification to the system – in practice, a peer can leave at any time without any notification. Therefore, in simulation, handling a leave event includes deleting related variables and events, *e.g.*, active send/receive events of the peer, while other peers can only detect the disconnection by sending requests or executing their membership/partnership management mechanism. It is not expected that peers send a leave notification.

▷ Bandwidth variation event: the available bandwidth of a peer may be changed. Bandwidth variations can be simulated by allowing the application issuing bandwidth variation events to the system. To handle such an event, the executor has to reallocate bandwidth for

all connections affected by the bandwidth variation of one connection, and update the scheduled time of the corresponding receive events of those connection.

The complete list of events with their name (in the simulator), type, source, and the streaming protocols is presented in Table A.1.

Table A.1: Events used in our simulation

Name	Type	Source	Protocol
e_playback	Periodic	Application and Executor	Common
e_join	Non-periodic	Application	Common
e_leave	Non-periodic	Application	Common
e_send	Non-periodic	Executor	Common
e_receive	Non-periodic	Executor	Common
e_member	Periodic	Application and Executor	Common
e_partner	Periodic	Application and Executor	Common
e_bwvariation	Non-periodic	Application	Chameleon
e_request	Non-periodic	Application and Executor	Common

A.4 Max-min Fair Rate Allocation in Simulation

When a network node has more than one connection with other nodes, it needs to allocate its available bandwidth 'fairly' among the connections. In practice, the rate allocation mechanism is defined by the IP protocol, *e.g.*, TCP and UDP. In simulation, allocating node bandwidth to each connection is approximated by a *max-min fair rate* allocation algorithm, firstly described in [96]. Piccolo *et. al.* [97] present an efficient implementation of the algorithm, summarized in Section A.4.1.

A.4.1 Traditional Implementation

They depict a generic node by explicitly indicating its bandwidth capacity, and the generic connection between nodes by means of a single edge linking them. The network is modeled as an undirected graph. If we denote:

- N: the number of nodes in the network. For each node $i \in \{1, 2, ..., N\}$, C_i is its bandwidth capacity (the bandwidth assigned to its access link).
- $A(i, j)$: the adjacency matrix, which expresses connectivity between nodes in the network.

$$A(i,j) = \begin{cases} 1 & \text{if } i \text{ is connected to } j \\ 0 & \text{otherwise} \end{cases}$$

- F_i: the allocated capacity of node i.
- U: the set of nodes with capacity still to be allocated.
- V_i: the set of connections of node i that still require to be allocated.
- n_i: the number of connections of node i still to be considered ($n_i =\mid V_i \mid$).
- r_{ij}: the allocated bandwidth of connection (i, j).

The algorithm computes the rate of each connection through at most N iterations as shown in Algorithm 9.

Algorithm 9 Traditional Algorithm

while ($U \neq \emptyset$) **do**

% Find the set B of bottleneck nodes

$$B = \{b \mid \frac{C_b - F_b}{n_b} = \min_{i \in U} \frac{C_i - F_i}{n_i}\}$$

% Update capacity

$$U = U - B$$

$$V_i = V_i - \bigcup_{b \in B} V_b$$

$$r_{ij} = r_{ij} + \frac{C_b - F_b}{n_b} \qquad \forall (i,j) \in \bigcup_{b \in B} V_b$$

$$F_b = C_b \qquad \forall b \in B$$

$$F_i = F_i + \sum_{b \in B} A(i,b) \frac{C_b - F_b}{n_b} \qquad i \in U$$

end while

Summarizing, in every iteration, the algorithm identifies the bottleneck nodes, which provide the most restrictive rate allocation because these nodes are required to allocate fairly all their capacity to their connections. Then, those bottleneck nodes and their connections are removed from the graph. The bandwidth of other nodes is updated, and the next iteration is carried on (until all nodes are considered).

During simulation, when the rate of a connection is changed, the above algorithm is invoked to re-compute the allocated rate for all connections in the network. However, such changing events, *e.g.* when a new connection is established or an old connection is ended, may affect only a subset of the existing connections. Therefore, the re-computation should be done on the subset only. In the optimized version [97], first, related connections/nodes whose bandwidth will be changed after the creation or the end of a connection are identified; then, Algorithm 9 is applied to re-computes the allocated bandwidth for the subgraph only.

A.4.2 A More Efficient and Practical Implementation

The traditional implementation (including its optimized version) of the max-min fair rate allocation algorithm can not be applied directly to simulate a P2P system because it models the network as an undirected graph, while a physical network is naturally modeled by a directed graph in which directed edges show the direction of data transmission from senders to receivers. In other words, the network needs to be 're-modeled' before using the algorithm, *e.g.*, converting from the directed graph to the undirected graph. Figure A.2 shows an example of the conversion.

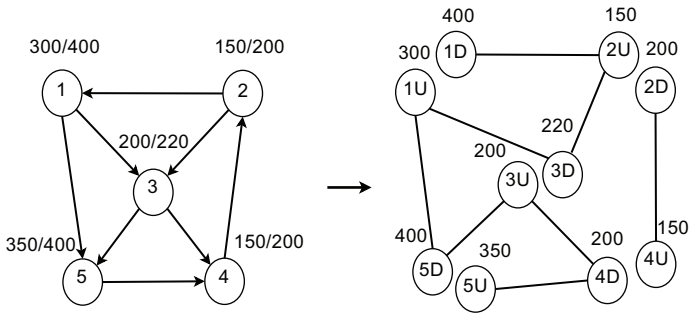


Figure A.2: An example of modeling the network to undirected graph. The directed edges on the left graph represent for the connections from the upload nodes to the download nodes. A physical node *i* is modeled as two distinct nodes *iU* and *iD* for its uplinks and downlinks in the right graph. The number above a node is its bandwidth (displayed as upload/download for the nodes in the left graph).

As observed in Figure A.2, the number of nodes is doubled in the traditional computation. This increases the CPU consumption in browsing nodes and memory to store the network topology. In the proposed approach, the allocated rates on directed graphs are computed by browsing every directed link. By considering every connection, it is not necessary to split a node into two distinct nodes. Since the number of nodes is kept the same as the network size, the CPU and memory are saved. To present the proposed algorithm, the following notations are used:

- UB_i, DB_i: the upload and download capacity of node i.
- AUB_i, ADB_i: the allocated upload and download capacity of node i.
- NUL_i,NDL_i: the number of unallocated uplinks and downlinks of node i.
- r_{ij}: the allocated rate of the connection from i to j, *i.e.*, i is the upload node, and j is the download node.
- $A(i, j)$: the adjacency matrix. However, in this case, $A(i, j) = 1$ means that i is the upload node and j is the download node in the connection.

Algorithm 10 Proposed Algorithm

$n\_l \leftarrow 0$: number of allocated links
NL: the total number of links
while ($n\_l < NL$) **do**
  % Computes the rate of each connection
  for $i = 1, N$ **do**
    for $j = 1, N$ **do**
      if ($A(i,j) = 1$) **then**
        $r_{ij} \leftarrow Min(UB_i/NUL_i, DB_j/NDL_j)$;
        $AUB_i \leftarrow AUB_i + r_{ij}$;
        $ADB_j \leftarrow ADB_j + r_{ij}$
      end if
    end for
  end for
  % Removes allocated connections and updates bandwidth of other nodes/connections
  for $i = 1, N$ **do**
    % bottleneck at the upload of node i
    if ($AUB_i = UB_i$) **then**
      for $j = 1, N$ **do**
        if ($A(i,j) = 1$) **then**
          $A(i,j) \leftarrow 0$;
          $DB_j \leftarrow DB_j - r_{ij}$; $UB_i \leftarrow UB_i - r_{ij}$;
          $NUL_i \leftarrow NUL_i - 1$; $NDL_j \leftarrow NDL_i - 1$;
          $n_l \leftarrow n_l + 1$;
        end if
      end for
    end if
    % bottleneck at the download of node i
    if ($ADB_i = DB_i$) **then**
      for $j = 1, N$ **do**
        if ($A(j,i) = 1$) **then**
          $A(j,i) \leftarrow 0$;
          $DB_i \leftarrow DB_i - r_{ji}$; $UB_j \leftarrow UB_j - r_{ji}$;
          $NUL_j \leftarrow NUL_j - 1$; $NDL_i \leftarrow NUL_j - 1$;
          $n_l \leftarrow n_l + 1$;
        end if
      end for
    end if
  end for
end while

The pseudo code of the algorithm is presented in Algorithm 10. It includes iterations, in each of which three steps are carried out:

- Step 1: Computes the rate of each connection as follows (line 5-13):

$$r_{ij} = Min(\frac{UB_i}{NUL_i}, \frac{DB_j}{NDL_j})$$

- Step 2: Since there is always a bottleneck at one extreme of any connection, we can identify connections from or to bottleneck nodes by checking if a node allocates all its bandwidth to its connections, ***i.e.*** the aggregate allocated rate of its connections is equal to its bandwidth capacity (line 17, 28). Those connections are called *saturated connections*. Saturated connections are marked and removed from the graph (line 20, 31).

- Step 3: The other algorithm parameters are updated.

The key difference between our approach and the traditional one is that, in each iteration, we identify saturated connections, while the traditional algorithm identifies bottleneck nodes. An example of the proposed computation for the network in Figure A.2 is depicted in Figure A.3. In the first iteration, step 1 computes the rate of every connection (Figure A.3(b)). Then, step 2 identifies saturated connections by comparing the rates calculated in step 1 with the bandwidth capacity of each node, ***e.g.*** the connections (2,1) and (2,3) are saturated because the sum of their rate is equal to the upload capacity of node *2*. The saturated connections are marked, and the bandwidth of every node is updated (Figure A.3(c)). After the first iteration, six – out of eight – connections are marked. The second iteration is carried out, and the result is shown in Figure A.3(d). Since, all connections are saturated, the computation ends.

A.4.3 Performance Evaluation

The proposed solution and the traditional one are implemented in C++. To compare their performance, the *clock()* function of the *ctime* library is used to calculate the relevant CPU consumption time. In the experiments, we consider a P2P network with a number of peers N. The upload and download capacity of each node is generated randomly within the range of [300, 1200] Kbps and [360, 2500] Kbps, respectively. Links are generated randomly among those peers but it is required that peers maintain an average number of connections AC to other peers. Experiments are run on a desktop PC with 1.0 GHz AMD Athlon 64 X2 Dual Core Processor $4800+$ with 2 GB RAM, and running GNOME $2.16.0$. For each experiment, the algorithms are run 10 times, and the average outcome is recorded.

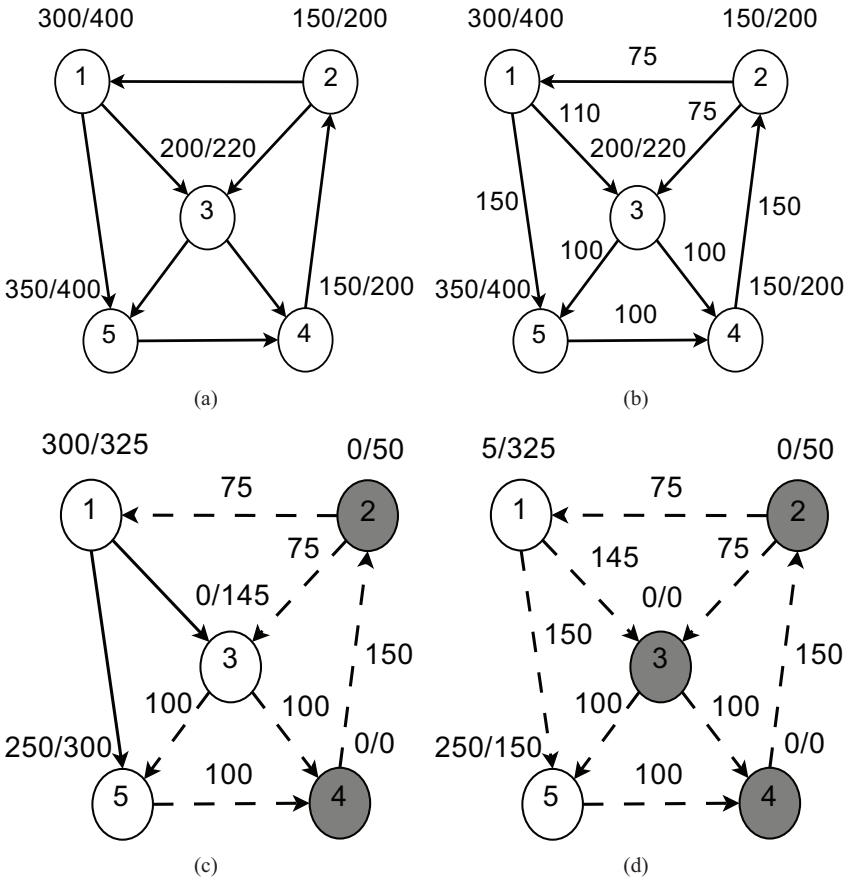


Figure A.3: An example of the computation of the proposed algorithm. Dashed links are saturated connections.

In the first experiment, the algorithms are compared on different network sizes ranging from 2000 to $10,000$ peers. The average number of per-peer connections is set to 10. We calculate the CPU consumption time each algorithm uses to allocate bandwidth for all connections in the network and plot the results in Figure A.4. As shown in Figure A.4, the proposed computation outperforms the traditional approach. In all cases, the proposed algorithm consumes only $18 - 22\%$ of the CPU time that is spent for the traditional algorithm. The correctness of the proposed algorithm is easily assessed by verifying that it always yields the same results as the traditional implementation. The method proposed in [97] is also applied to only re-compute the rate of related links when a connection is established or terminated, or when a bandwidth variation occurs. This further optimizes the proposed algorithm in terms of computational cost as it can be observed from Figure A.4 that the CPU consumption time is increased almost linearly in our solution with the network size.

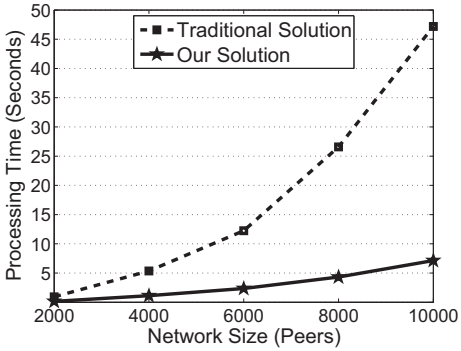


Figure A.4: The performance of the algorithms with different network sizes

In the next experiment, we would like to see how the algorithms perform when the average number of connections AC changes. The value of AC is varied from 5 to 20. The network size is fixed to 5000 peers. The results are plotted in Figure A.5. When AC increases, the CPU consumption time also increases because there are more connections to be computed in the network. However, the proposed approach still consumes less CPU time than the traditional computation.

A.5 Summary

This chapter presents the packet-level discrete-event simulator in detail. In addition to a more efficient and realistic bandwidth allocation model, the simulator considers both peer dynamic and bandwidth variations. It is believed that such a model is sufficient to evaluate important

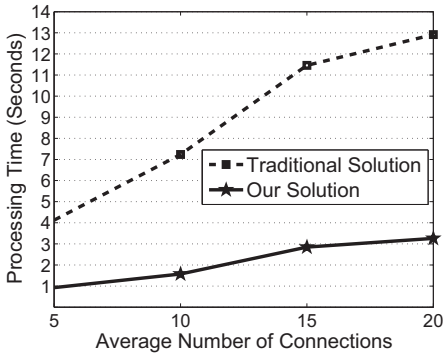


Figure A.5: The performance of the algorithms with different values of AC

features of a P2P streaming system. The simulator was used to evaluate the proposed P2P streaming protocols: Chameleon, Stir, and Chameleon++.

Bibliography

[1] R. Steinmetz and K. Wehrle, *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[2] I. Multimedia Research Group. (2010) IPTV Global Forecast. [Online]. Available: http://www.mrgco.com/iptv/gf0610.html

[3] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Trans. on Multimedia*, vol. 9, no. 8, Dec. 2007.

[4] X. Zhang, J. Liu, B. Li, and T.-S. Yum, "CoolStreaming/DONet: a Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming," in *Proc. of IEEE INFOCOM*, vol. 3, March 2005, pp. 2102–2111.

[5] TVAnts. [Online]. Available: http://tvants.en.softonic.com/

[6] U. Inc. UUSee. [Online]. Available: http://www.uusee.com/

[7] R. Schafer and T. Sikora, "Digital Video Coding Standards and Their Role in Video Communications," *Proceedings of the IEEE*, vol. 83, no. 6, pp. 907–924, June 1995.

[8] I.-T. R. H.264, I.-T. ISO/IEC 14496-10 (MPEG-4 AVC), and V. . ISO/IEC JTC. (2007, July) Advanced Video Coding for Generic Audiovisual Services.

[9] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.

[10] M. Wien, H. Schwarz, and T. Oelbaum, "Performance Analysis of SVC," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1194–1203, Sep. 2007.

[11] J. Chakareski, S. Han, and B. Girod, "Layered Coding vs. Multiple Descriptions for Video Streaming Over Multiple Paths," in *Proc. of ACM Multimedia*, 2003, pp. 422–431.

[12] S. Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," *IEEE Trans. Info. Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[13] M. Wang and B. Li, "R^2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1655–1666, Dec. 2007.

[14] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Trans. Info. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul 2000.

[15] L. Ramaswamy and L. Liu, "Free Riding: A New Challenge to Peer-to-Peer File Sharing Systems, year=2003, month=Jan, volume=, number=, pages=,."

[16] Facebook. [Online]. Available: http://www.facebook.com/

[17] Alexa Top 500 Global Sites. [Online]. Available: http://www.alexa.com/topsites

[18] Facebook Statistics. [Online]. Available: http://www.facebook.com/press/info.php?statistics

[19] 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," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 2, 2008.

[20] K.-J. Lin, C.-P. Wang, C.-F. Chou, and L. Golubchik, "SocioNet: A Social-Based Multimedia Access System for Unstructured P2P Networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 7, Jul 2010.

[21] Z. Liu, H. Hu, Y. Liu, K. Ross, Y. Wang, and M. Mobius, "P2P Trading in Social Networks: The Value of Staying Connected," in *Proc. of IEEE INFOCOM*, 2010.

[22] R. Kumar, Y. Liu, and K. Ross, "Stochastic Fluid Theory for P2P Streaming Systems," in *Proc. of IEEE INFOCOM*, 2007.

[23] L. Vu, I. Gupta, K. Nahrstedt, and J. Liang, "Understanding the Overlay Characteristics of a Large-scale Peer-to-Peer IPTV System," *ACM Trans. on Multimedia Computing, Communications and Applications*, Feb 2011.

[24] Flickr. [Online]. Available: http://www.flickr.com/

[25] C. R. Kothari, *Research Methodology: Method and Techniques*, 2nd ed. New Delhi, India: New Age International (P) Ltd., 1990.

[26] D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young, "Computing as a Discipline," *Communications of the ACM*, vol. 32, pp. 9–23, January 1989.

[27] A. T. Nguyen, B. Li, and F. Eliassen, "Chameleon: Adaptive Peer-to-Peer Streaming with Network Coding," in *Proc. of IEEE INFOCOM*, March 2010.

[28] ——, "Quality- and Conext-aware Neighbor Selection for Layered Peer-to-Peer Streaming," in *Proc. of IEEE ICC*, May 2010.

[29] A. T. Nguyen, F. Eliassen, and M. Welzl, "Quality-aware Membership Management for Layered Peer-to-Peer Streaming," in *Proc. of IEEE CCNC*, Jan 2011.

[30] A. Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[31] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang. (2008) Network Coding Theory. [Online]. Available: http://iest2.ie.cuhk.edu.hk/~whyeung/publications/tutorial.pdf

[32] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *Proc. of International Symposium on Information Theory (ISIT 2003)*, 2003.

[33] P. Chou, Y. Wu, and K. Jain, "Practical Network Coding," in *Proc. of the 41st Allerton Conference on Communication, Control, and Computing*, Oct 2003.

[34] 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, Jan 2005.

[35] K. Joohee, R. Mersereau, and Y. Altunbasak, "Distributed Video Streaming Using Multiple Description Coding and Unequal Error Protection," *Image Processing, IEEE Transactions on*, vol. 14, no. 7, pp. 849–861, July 2005.

[36] S. Wenger, Y.-K. Wang, and T. Schierl, "Transport and Signaling of SVC in IP Networks," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1164–1173, Sep. 2007.

[37] Y. Wang, M. Hannuksela, S. Pateux, A. Eleftheriadis, and S. Wenger, "System and Transport Interface of SVC," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1149–1163, Sep. 2007.

[38] SopCast. (2010) SopCast - Free P2P Internet TV. [Online]. Available: http://www.sopcast.org/

[39] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. K., J. W. O'Toole, Jr., M. Frans, and K. James, "Overcast: Reliable Multicasting with an Overlay Network," in *Usenix OSDI Symposium*, 2000, pp. 197–212.

[40] P. C. J. Li and C. Zhang, "Mutualcast: An Efficient Mechanism for One-to-Many Content Distribution," in *Proc. of the 1st ACM SIGCOMM Asia Workshop (SIGCOMM ASIA 2005)*, 2005.

[41] M. B. H. Deshpande and H. Garcia-Molina, "Streaming Live Media Over Peer-to-Peer Network," Standford University, Tech. Rep., 2001.

[42] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of ACM SIGCOMM 2002*, 2002.

[43] D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming," in *Proc. of IEEE INFOCOM 2003*, 2003.

[44] J. A. D. Kostic, A. Rodriguez and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, 2003.

[45] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous Unstructured Tree-based Peer to Peer Multicast," in *Proc. of the 14th IEEE International Conference on Network Protocols (ICNP 2006)*, 2006.

[46] V. Venkataraman, P. Francis, and J. Calandrino, "Chunkyspread: Multitree Unstructured Peer-to-Peer Multicast," in *Proc. of the 5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[47] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-Based Streaming," in *Proc. of IEEE INFOCOM*, May 2007, pp. 1415–1423.

[48] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang, "A Peer-to-Peer Network for Live Media Streaming: Using a Push-Pull Approach," in *Proc. of ACM Multimedia 2005*, 2005.

[49] F. Wang, Y. Xiong, and J. Liu, "mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast," in *Proc. of IEEE ICDCS 2007*, June 2007.

[50] J. D. Mol, D. H. P. Epema, and H. J. Sips, "The Orchard Algorithm: Building Multicast Trees for P2P Video Multicasting Without Free-Riding," *IEEE Trans. on Multimedia*, vol. 9, no. 8, pp. 1593–1604, Dec. 2007.

[51] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, 2003.

[52] V. Padmanabhan, H.Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2002)*, 2002.

[53] Y. Cui and K. Nahrstedt, "Layered Peer-to-Peer Streaming," in *Proc. of NOSSDAV*, Jun. 2003, pp. 162–171.

[54] R. Rejaie and A. Ortega, "PALS: Peer-to-Peer Adaptive Layered Streaming," in *Proc. of NOSSDAV*, Jun. 2003, pp. 153–161.

[55] Y. Liu, W. Dou, and Z. Liu, "Layer Allocation Algorithms in Layered Peer-to-Peer Streaming," in *Proc. of IFIP international conference on network and parallel computing (NPC)*, Oct. 2004, pp. 167–174.

[56] N. Magharei and R. Rejaie, "Adaptive Receiver-Driven Streaming from Multiple Senders," *Multimedia Systems*, vol. 11, no. 6, pp. 550–567, Jun. 2006.

[57] M. Mushtaq and T. Ahmed, "Smooth Video Delivery for SVC Based Media Streaming Over P2P Networks," Jan 2008, pp. 447–451.

[58] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," pp. 1424–1432, May 2007.

[59] J. Seibert, D. Zage, S. Fahmy, and C. Nita-Rotaru, "Experimental Comparison of Peer-to-Peer Streaming Overlays: An Application Perspective," in *LCN 2008*, Oct. 2008, pp. 20–27.

[60] A. Ganesh, A.-M. Kermarrec, and L. Massoulie, "Peer-to-Peer Membership Management for Gossip-Based Protocols," *IEEE Trans. on Computers*, vol. 52, no. 2, pp. 139–149, Feb. 2003.

[61] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-based Implementations," in *Proc. of the 5th ACM/IFIP/USENIX International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 79–98.

[62] J. Zhao, F. Yang, Q. Zhang, Z. Zhang, and F. Zhang, "LION: Layered Overlay Multicast With Network Coding," *IEEE Trans. on Multimedia*, vol. 8, no. 5, pp. 1021–1032, Oct. 2006.

[63] X. Xiao, Y. Shi, B. Zhang, and Y. Gao, "OCals: A Novel Overlay Construction Approach for Layered Streaming," May 2008, pp. 1807–1812.

[64] J. Sacha, B. Biskupski, D. Dahlem, R. Cunningham, R. Meier, J. Dowling, and M. Haahr, "Decentralising a Service-Oriented Architecture," *Peer-to-Peer Networking and Applications*, vol. 3, pp. 323–350, 2010.

[65] A. Payberah, J. Dowling, F. Rahimian, and S. Haridi, "*gradienTv*: Market-Based P2P Live Media Streaming on the Gradient Overlay," in *Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6115, pp. 212–225.

[66] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. of IEEE INFOCOM 2005*, 2005.

[67] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P Content Distribution System with Network Coding," in *Proc. of the 5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[68] BitTorrent. [Online]. Available: http://www.bittorrent.com/

[69] D. Qiu and R. Srikant, "Modeling and Performance Analysis of Bittorrent-Like Peer-to-Peer Networks," in *Proc. of ACM SIGCOMM*, 2004.

[70] D. W. Y. Tian and K. W. Ng, "Modeling, Analysis and Improvement for Bittorrent-Like File Sharing Networks," in *Proc. of IEEE INFOCOM 2006*, 2006.

[71] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. R. Rodriguez, "Is High-Quality VoD Feasible using P2P Swarming?" in *Proc. of the 16th international Conference on World Wide Web (WWW)*, Aug. 2007, pp. 903–912.

[72] M. Wang and B. Li, "Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming," in *Proc. of IEEE INFOCOM*, May 2007, pp. 1082–1090.

[73] H. Shojania and B. Li, "Parallelized Progressive Network Coding With Hardware Acceleration," in *Proc. of Fifteenth IEEE International Workshop on Quality of Service*, June 2007, pp. 47–55.

[74] D. Stutzbach and R. Rejaie, "Understanding Churn in Peer-to-Peer Networks," in *Proc. of the 6th ACM SIGCOMM conference on Internet measurement (IMC)*. New York, NY, USA: ACM, 2006, pp. 189–202.

[75] ITU-T and I. JTC1. (2008) JSVM Software version JSVM 9.17. [Online]. Available: http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm

[76] PeerSim: A Peer-to-Peer Simulator. [Online]. Available: http://peersim.sourceforge.net/

[77] S. Voulgaris, D. Gavidia, and M. V. Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays," *Journal of Network and Systems Management*, vol. 13, 2005.

[78] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a Feather: Homophily in Social Networks," *Annual Review of Sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[79] P. Singla and M. Richardson, "Yes, There is a Correlation - From Social Networks to Personal Behavior on the Web," in *Proc. of ACM WWW*, 2008, pp. 655–664.

[80] Z. Xiao, L. Guo, and J. Tracey, "Understanding Instant Messaging Traffic Characteristics," in *Proc. of IEEE ICDCS*, 2007.

[81] J. Leskovec and E. Horvitz, "Planetary-Scale Views on a Large Instant-Messaging Network," in *Proc. of ACM WWW*, 2008, pp. 915–924.

[82] B. A. Nardi, S. Whittaker, and E. Bradner, "Interaction and Outeraction: Instant Messaging in Action," in *Proc. of ACM CSCW*, 2000.

[83] I. Menken, *Cloud Computing - The Complete Cornerstone Guide to Cloud Computing Best Practices*. London, UK, UK: Emereo Pty Ltd, 2008.

[84] Android Cloud to Device Messaging Framework. [Online]. Available: http://code.google.com/android/c2dm/index.html

[85] NodeXL. [Online]. Available: http://nodexl.codeplex.com/

[86] PPLive Traces. [Online]. Available: http://dprg.cs.uiuc.edu/downloads/

[87] E. Sgarbi and D. L. Borges, "Structure in Soccer Videos: Detecting and Classifying Highlights for Automatic Summarization," *Lecture Notes in Computer Science*, vol. 3773/2005, pp. 691–700, 2005.

[88] PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services. [Online]. Available: http://www.planet-lab.org/

[89] S. Robinson, *Simulation : The Practice of Model Development and Use*. John Wiley & Sons, 2004.

[90] C. Kiddle, R. Simmonds, C. Williamson, and B. Unger, "Hybrid Packet/Fluid Flow Network Simulation," in *Proc. of the seventeenth workshop on Parallel and distributed simulation. PADS 2003*, June 2003, pp. 143–152.

[91] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers, "The state of peer-to-peer simulators and simulation," in *Proc. of ACM SIGCOMM*, vol. 37, no. 2, New York, NY, USA, 2007, pp. 95–98.

[92] The Network Simulator ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/

[93] M. Baker and T. Guili, "Narses: A scalable flow-based network simulator," in *Technical report, Stanford University*, 2002. [Online]. Available: http://arxiv.org/abs/cs/0211024v1

[94] The Overlay Simulation Framework. [Online]. Available: http://www.oversim.org/

[95] OMNeT++ – Network Simulation Framework. [Online]. Available: http://www.omnetpp.org/

[96] Y. Bertsekas and R. Gallager, *Data Networks*. New York: Prentic Hall, Englewood Cliffs, 1987.

[97] F. L. Piccolo, G. Bianchi, and S. Cassella, "QRP03-4: Efficient Simulation of Bandwidth Allocation Dynamics in P2P Networks," in *Proc. of IEEE GLOBECOM 2006*, 1 December 2006, pp. 1–6.