ABSTRACT

SO, JUNG KI. Defending against Malicious Behaviors in BitTorrent Systems. (Under the direction of Dr. Douglas S. Reeves.)

Peer-to-Peer (P2P) systems are a promising and cooperative model for the recent Internet applications, such as content (file) sharing, multimedia streaming, and distributed computing. Unlike traditional client server models, P2P systems are based on participants' active cooperation. This cooperative nature is helping to develop various applications, and contributing toward common goals for their participants. P2P systems can accommodate a great number of participants. Despite scalable service, mutual benefits among participants are achievable through cooperation, without dedicated servers in the systems. In particular, BitTorrent systems, a typical P2P approach, maximize peer cooperation with simple and scalable operations. The BitTorrent systems have not only attracted a great number of users through efficient content distribution, but also have contributed a major portion of the Internet traffic by the scalable users.

Unfortunately, the lack of security mechanisms in P2P systems makes compliant peers vulnerable when achieving peer cooperation. From the early stages of P2P systems, there was little concern about security issues guarding against strategic attackers. Most P2P systems including BitTorrent have optimistically expected users to follow the protocol descriptions, and mainly concentrated on various functions of management for applications. As a direct effect of such conditions, selfish users (freeriders) can take advantage of protocol weakness for their own profits violating the regulated standards. Moreover, malicious users can launch various attacks on compliant users, which will eventually lead to system collapse.

In this dissertation, we investigate vulnerabilities of BitTorrent systems, and propose general solutions to address the problems, such as *Sybil attacks*, *Free-riding*, and *Byzantine attacks (cheating attacks)*. These attacks are difficult to prevent due to the typical P2P paradigm with no central authority. Therefore, our solutions are fully decentralized against malicious or selfish participants in P2P systems. Each user makes its own decision without an authority's intervention. Although a peer's decision depends on directly connected neighbors, cooperation with them can improve detection through extension of the local view, and prevention against the malicious attacks. Based on peer cooperation, the proposed approaches defeat selfish behaviors, cheating attacks, and malicious attacks triggered by fake identities. They also help participants achieve performance enhancement in spite of the seriously unfavorable conditions in which various attacks are.

First, we propose *GOLF* with *locality filtering* against threats from Sybil attackers in file-sharing systems. GOLF scheme fosters peer cooperation to exclude potential attackers, and locality filtering tentatively identifies Sybil attackers based on patterns in IP addresses. Under the proposed scheme, the effect of Sybil attackers sharply decreases. GOLF effectively detects and blocks potential attackers,

despite false accusation.

Second, we present *CoVoD*, a novel cooperative P2P VoD streaming approach, to penalize uncooperative peers. CoVoD adopts a referral based neighbor management scheme. Each peer expands its local view through referral information, and rewards the source of the referral. Such referral incentives can achieve direct cooperation among peers with asymmetric interests. Experimental results demonstrate that CoVoD improves service quality over alternatives, and strongly penalizes freeriders (peers who do not contribute resources).

Third, we address a defense mechanism, called *AntiLiar*, against cheating attacks in mesh based streaming systems. It provides participants with a practical authentication and informational integrity based on a secure progress log and a verifiable evidence. A consistent neighbor management improves peer cooperation and blacklists malicious liars. Experimental results show that AntiLiar improves costs to defeat cheating attacks, and achieves a good service quality in a mesh based streaming system.

Defending against Malicious Behaviors in BitTorrent Systems

by
Jung Ki So

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2012

APPROVED BY:

Dr. Peng Ning	Dr. Ting Yu
Dr. Khaled Harfoush	Dr. Douglas S. Reeves Chair of Advisory Committee

DEDICATION

To my beautiful wife, ***GyoungAh Kim****,*
my adorable son, ***GunYoung****,*
my brothers, ***Sun Ki So****,* ***Bumju Kim****, and* ***Minju Kim****,*
and parents for their love and encouragement.

BIOGRAPHY

Jung Ki So was born on March 11, 1974, in South Korea. He received a Bachelor of Engineering degree in the Department of Information Engineering at Korea Military Academy in 1997. After serving various military assignments in Korean Army from 1997 to 2004, he started his graduate study, and earned his Master of Science degree in the Department of Computer Science at Seoul National University in 2006. He continued the graduate study with Ph.D program in the Department of Computer Science at North Carolina State University in 2007. Since he joined the Cyber Defense Lab directed by Dr. Douglas S. Reeves in 2008, he has focused his research on Peer-to-Peer systems and network security. During his doctoral study at North Carolina State University, he served as a teaching and a research assistant, and published his papers in conference proceedings. On May 9, 2012, he successfully defended his Ph.D. dissertation. He is currently working for Defense Acquisition Program Administration (DAPA) for the national defense in Seoul, South Korea.

ACKNOWLEDGEMENTS

I would like to thank all people who have helped and inspired me during my doctoral study. First of all, I would like to express my deepest gratitude to my advisor, Dr. Douglas S. Reeves, for his endless concern, inspiration, commitment, and support. Throughout my doctoral research, he continually encouraged me to develop analytical thinking and research skills, and patiently guided me in the right direction.

I am very grateful for having an exceptional doctoral committee and wish to thank Dr. Peng Ning, Dr. Khaled Harfoush, and Dr. Ting Yu, for their interest in my work and for their valuable comments and encouragement. I especially thank Dr. Peng Ning who introduced me into the advanced security research fields. I would like to thank Dr. Harry Perros and Dr. Rudra Dutta for providing me insightful comments and valuable feedbacks in my oral and written qualifying exam.

I am pleased to thank Dr. Bob Pond, my mentor, for his spiritual guide to the Lord and friendship, during my doctoral study. I realized and learned God's messages through his mouth. I would like to thank my former advisor in Seoul National University, Dr. Heon Young Yeom, for giving kind advice and special encouragements during the preparation of Ph.D program. I also would like thank Korean Army and Defence Acquisition Program Administration for giving me this invaluable chance.

I truly appreciate Korean army officers, Dr. Chongkyung Kil, Dr. Kyuyong Shin, Woopan Lee, Dr. Yongchul Kim, and Kiseok Jang for valuable time we had together at NC State. I cannot forget to mention my friends, Jasan Lee's, YoungHyun Oh's, SungRyong Baek's, HyungJun Kim's family, and Paster KwanSeok Kim. I appreciate all my lab colleagues for their help in various ways.

It has been great to work with Dr. William Enck, Dr. Xuxian Jiang, Dr. Steffen Heber, and my committee members (Dr. Reeves, Dr. Ning, Dr. Yu, and Dr. Harfoush) as a teaching assistant for several semesters, acquiring invaluable teaching experience at NC State. In addition, I would like to thank Ms. Margery Page and Ms. Carol Allen for constant help.

My deepest gratitude goes to my family, GyoungAh and GunYoung. Their endless love, patience, and trust always help me keep moving forward in my life. I really appreciate my extend families for concern, support, and encouragement in South Korea. I especially thank my brothers (Sun Ki So, Bumju Kim, and Minju Kim) and their families for supporting and taking care of my parents.

Finally, but most importantly, thank God for saving my life and always being with me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Peer-to-Peer (P2P) systems are becoming increasingly popular for many applications, including file sharing, multimedia streaming, and distributed computing. One of the significant differences from a client server model is that P2P systems provide an environment where participants become both a supplier and a consumer and dynamically cooperate with each other. Such flexible roles of the participants make P2P systems scalable. Dynamic cooperation among peers enables to provide a robust service in P2P overlay applications. Furthermore, the openness of most P2P applications provides an anonymous identity to a participant for privacy. These properties make P2P systems to be popular. In this environment, peers establish multiple connections and interact with others without being previously known, which allows to achieve their common goals by sharing their respective resources. To this end, voluntary and friendly cooperation among peers is a fundamental premise for the success of P2P systems.

Most P2P systems have optimistically expected users to follow the protocol descriptions, and mainly concentrated on various functions of management for applications. This faith makes the protocols vulnerable, and malicious attackers exploits such security holes. Moreover, P2P systems delegate general decisions to each participant. The absence of central authorities for system management causes various malicious behaviors in the interaction among participants. Since there is no coordinator to monitor and control participants' activities, each peer has to self-organize to overcome the unreliable conditions. However, it is difficult for peers to make a correct decision with a limited view in a fully decentralized environment.

BitTorrent, in particular, is a typical P2P model, and its protocols [1] are used in various P2P applications; for example, file-sharing, VoIP, and multimedia streaming. Due to simple and cooperative operations, BitTorrent has contributed a major portion of the Internet traffic by scalable users. According to the Internet study [2], BitTorrent is the most popular P2P protocol and the dominating protocol followed by HTTP. Unfortunately, the lack of security mechanisms in BitTorrent makes participants vulnerable to various attacks by strategic users. Selfish users (freeriders) tend to exploit protocol weakness for their own profits violating the regulatory standards [3, 4, 5]. Moreover, malicious users can attempt

to launch various attacks on peers, such as cheating [6, 7, 8, 9], file piece pollution [10, 11], connection slot consumption [4, 12], and bandwidth exhaustion [3, 13]. Furthermore, Since BitTorrent offers a newcomer an identity at no charge, attackers can create a great number of new false identities for easy avoidance from detection of participants and continuous interruption of normal operations. Victims cannot differentiate Sybil attackers from legitimate peers. Even if an attacker is detected and penalized, it can avoid punishment after a simple change of an identity. Thus, malicious attacks can continue despite a detection by participants. Such malicious behaviors can eventually lead to system collapse.

The best way to defeat such malicious behaviors is to provide effective defense mechanisms in BitTorrent. These approaches need to be fully decentralized in order to defend against malicious or selfish attackers without central authorities. For a decentralized mechanism, participants should actively cooperate with each other to make good decisions. Effective incentive or reputation mechanisms have to promote repeated cooperation for legitimate peers. Thus, our solutions aim to establish decentralized and cooperative behaviors in BitTorrent systems.

The goal of this dissertation is to make BitTorrent systems robust for defending against existing and potential malicious attacks. To accomplish this goal, we investigate major threats and vulnerabilities in BitTorrent systems, such as *Sybil attacks*, *freeriding*, and *cheating attacks*. As a solution for respective problems, we introduce practical and effective approaches that achieve a decentralized and cooperative behaviors among peers.

1.1 Research Problems & Countermeasures

This section overviews important research problems in BitTorrent systems with a content swarm. The protocol vulnerabilities and existing threats in such systems decrease system availability and data integrity, which causes a significant loss of cooperation derived scalability, robustness, and flexibility in P2P advantages. To address this risk, we propose practical and decentralized methods for improving availability and integrity in BitTorrent systems.

1.1.1 Sybil Attacks in File Sharing Systems

As mentioned, BitTorrent has suffered from various attacks such as fake block attack [10, 11, 14], network resource exhaustion [12, 13], and swarm poisoning [15] due to the lack of security mechanisms. These effects are made worse by the ability of attackers to manufacture a great number of new identities (*Sybil nodes*) at will [16]. The net effect of Sybil nodes and weak security leads to inefficient BitTorrent operation, or collapse. The Sybil attack is a fundamental and pervasive problem in P2P systems. Despite researchers' efforts, the Sybil attack has yet to be successfully prevented in the absence of central authority [17, 18, 19, 20, 21, 22, 23, 24, 25].

In Chpater 4, we propose **GO**od **L**eecher **F**riends (*GOLF*), with *locality filtering* to sharply mitigate

the impacts of Sybil attackers. GOLF is a simple and direct reputation scheme. It detects a local attacker with *block filtering* against the file piece pollution, and cooperates with directly connected neighbors to exclude potential attackers. Locality filtering tentatively identifies Sybil nodes based on patterns in IP addresses. The proposed scheme is fully decentralized, and easy to deploy in the current BitTorrent systems.

Under the proposed scheme, Sybil attackers may still continue malicious behaviors, but their effect sharply decreases. GOLF decreases the downloading of fake blocks to nearly zero, and improves the average completion time. With locality filtering, a seeder cuts down bandwidth consumption by a factor of ten in the presence of Sybil nodes. Comparison to existing reputation models shows detection rates against both potential attackers and false accusation. GOLF effectively detects and blocks potential attackers in the case of false accusation.

1.1.2 Selfish Behaviors in P2P VoD Streaming Systems

Selfish behaviors interrupt voluntarily peers' cooperations, which throws doubt on the system's usefulness. According to the recent measurement studies [26, 27, 28], selfish users (freeriders) are prevalent in P2P systems. Anti-freeriding schemes have been addressed for the cooperation of participants toward the mutual benefits. They are divided into four categories, which are; direct reciprocity [29, 30, 31, 9, 8], reputation schemes [32, 33, 34], currency-based schemes [35, 36, 37], and piece modification schemes [38, 13]. Such efforts have been mainly designed for and focused on file sharing. Although this phenomenon is not much different for P2P VoD streaming applications, solutions developed for file sharing cannot be applied directly. This is because P2P VoD design faces additional challenges for cooperation: (i) designing an effective scheme for cooperation between peers with asymmetric interests, and (ii) guaranteeing streaming quality of service (QoS) under stringent time constraints.

In Chapter 5, we present a novel cooperative P2P VoD streaming approach, called CoVoD scheme, to tackle these challenges. In the proposed scheme, each peer adaptively manages its own neighbors based on referrals among peers with asymmetric interests. The referrals expand the local view of peers, and assist in finding cooperative neighbors with similar interests. The provider of a referral obtains the opportunity to download the desired data from the consumer of the referral even if they have no piece to be bartered. The referral incentive enables peers joining later in the video playback to get bootstrapped quickly, avoiding the cold start problem (i.e., delaying newcomer participation because they have nothing to trade), but without creating opportunities for freeriders to exploit seeding protocols.

Experimental results demonstrate that CoVoD strongly penalizes freeriders and outperforms existing freeriding alternatives for the aspects of fairness and service quality. The proposed scheme accomplishes high uplink utilization and close to optimal fairness despite the asymmetry of interests among peers, and the presence of freeriders. Asymmetric peers achieve more than 95% seamless QoS when 20% of the participants are freeriders, with less than 3% communication overhead.

1.1.3 Cheating Attacks in Mesh based Streaming Systems

Cheating is a powerful strategy that destroys trust and defeats mutual cooperation toward the common goal in distributed systems. BitTorrent is not immune to threats of cheating attacks. Through false gossiping, strategic liars can trick compliant peers, and interrupt them from performing normal actions. A liar can take advantage of ambiguity by a limited interaction between two communication parties. A typical example is piece under(over)-reporting in a BitTorrent-like swarm; a liar generating fake control messages makes peers to waste resources, and disturbs the completion of a file in the swarm [8, 9]. On the other hand, a liar can exploit uncertainty in a mesh based streaming system based on an asynchronous overlay network. When a message is lost or omitted, peers cannot identify whether or not it is a network fault or an intentional omission. The liar rejects and ignores data dissemination to other peers [6, 7]. In the end, compliant peers undergo screen jitters or frozen images on the service time. In addition, a liar can interrupt peer discovery to intensify the impact of cheating. Hence, defeating cheating attacks is the core to making BitTorrent systems more robust.

In Chapter 6, we propose a defense mechanism, *AntiLiar*, against cheating attacks in the context of mesh-based streaming. The goal of AntiLiar provides a practical method to defeat liars' impact for service quality and peer cooperation. AntiLiar uses a secure *progress log* that consists of commitments and one-time signatures for authentication and message integrity. Each peer maintains its own progress log and advertises the most current state of obtained pieces. Peers trace neighbors' state and validate their commitment and signature released. They detect a manipulated report through a verifiable evidence sent from a one-hop neighbor and a consistent neighbor management. They prevent an adverse effect of piece pollution through block filtering. To distinguish a willful omission in an asynchronous network, AntiLiar uses non-repudiation of the verifiable evidence.

Experimental results demonstrate that AntiLiar minimizes required costs, and improves service quality over alternatives in the face of cheating attacks. Compliant peers with AntiLiar experience 95% seamless video service, even when 10% liars in participants exploit whitewashing and collusion. Comparison to alternatives shows AntiLiar significantly improves overall costs, and provides scalability and robustness.

1.2 Contributions & Properties

This dissertation provides security mechanisms for system availability and integrity with a decentralized and practical manner in BitTorrent systems. The major contributions are as follows:

- ***Mitigating Sybil attacks (GOLF with locality filtering).***
  GOLF with locality filtering significantly mitigates the malicious impact from Sybil attackers. To maintain *GO*od *L*eecher *F*riends (GOLF), peers build a trusted relation based on peers' history of previous, mutually-successful exchanges. Locality filtering tentatively identifies Sybil attackers

based on patterns in IP addresses. GOLF improves the average completion time about 4 times when 50% of the participants are Sybil attackers. With locality filtering, a seeder cuts down bandwidth consumption more than 10 times in the presence of Sybil attackers.

- ***Penalizing uncooperative peers (CoVoD).***
  CoVoD provides a novel P2P VoD streaming method that works when a significant percentage of participants are uncooperative. Through referral information, each peer expands its local view, rewards the source of the referral by means of an upload slot. Each peer makes an unchoking decision based on the history of neighbors' contributions. Without referral or piece contribution, uncooperative peers cannot achieve a good quality of experience. Compliant peers with CoVoD achieve more than 95% seamless QoS when 20% of the participants are freeriders, with less than 3% communication overhead.

- ***Defeating cheating attacks (AntiLiar).***
  AntiLiar effectively defeats the impact of liars with information integrity and authentication, through commitment trees and verifiable evidence from one-hop neighbors. With AntiLiar each peer maintains a consistent view to prevent a corrupted swarm view by liars. Although cheating attacks severely impact quality of service, waste resources, and discourage cooperation among peers, the proposed scheme effectively defeats these impacts. Compliant peers with AnitLiar experience greater than 98% quality of service, when the percentage of liars is less than or equal to 30%.

- ***Improving system availability.***
  GOLF with locality filtering improves the system resources capacity. GOLF uses a simple reputation scheme through peer cooperation. As long as each peer cooperates with others, it can protect itself from potential attackers. Locality filtering is a resource testing approach that flags possible Sybil nodes, based on similarities in their IPv4 addresses. With the proposed schemes peers effectively download file pieces, and redistribute their pieces cached locally. GOLF with locality filtering blocks the leakage of system capacity from Sybil attackers.

  CoVoD improves system availability under freeriding in a P2P VoD system. Although asymmetric interest environments reduce peer cooperation and aggravate freeriding among peers, an adaptive neighbor management based on referrals improves service quality and discovery of compatible neighbors. Without cold start problem (newcomer penalty), a referral incentive helps peers to voluntarily share their resources between a file piece and the information of compatible neighbors state.

  AntiLiar improves utilization of system capacity under cheating attacks in a mesh based streaming system. Although liars attempt to deceive and hide their resources, AntiLiar effectively counters their strategies. To defeat the impact of liars it uses a commitment tree and a verifiable evidence

from a one-hop neighbor. Each peer can maintain a consistent view for expanded neighbors. With AntiLiar, peers can effectively discover compatible others, and barter their resources. This process ensures higher resource availability.

- ***Providing integrity.***
  Block filtering is a novel fine-grained integrity checking that provides a simple, distributed, but cost-effective method in BitTorrent-like systems. Once the original seeder generates and adds a block filter in the metainfo file (i.e. *.torrent*), each participant can verify block-level integrity instead of piece-level. Such a general block-level verification can be used in both file-sharing and multicasting of audio and video. In this dissertation, GOLF and AntiLiar use block filtering for anti-pollution of a file piece.

  Progress log is a tamper-evident log that provides a deterministic model in P2P systems. This appended-only log consists of commitments and one-time signatures for informational integrity. Each peer maintains and advertises the most current state of obtained pieces in its own progress log. With shared information, peers trace neighbors' state and validate their released commitment and signature through fast hash operations.

In addition, the proposed methods have desirable common properties as follows:

- ***Decentralized approaches.***
  Our proposed methods (GOLF, CoVoD, and AntiLiar) are a decentralized approach. First, GOLF does not require a central authority for a reputation system. Based on a private bartering history, each peer determines neighbors' trust, and self-organizes them. Second, CoVoD penalizes uncooperative peers through individual decisions. Each peer makes an unchoking decision based on the history of neighbors' contributions. Without referral or piece cooperation, freeriders cannot achieve a good QoS. Third, AntiLiar enables each peer to prevent cheating attacks as a decentralized manner. Each peer verifies its neighbors' state by comparing messages between shared from a one-hop neighbor and obtained from a directly connected neighbor. It keeps a consistent neighbor view without reliance on the tracker.

- ***Practical and cost-effective approaches.***
  The proposed schemes are practical and cost-effective. First, filter based approaches (block filter and locality filter) use Bloom filters that are a space-efficient data structure. Block filter reflects the whole block digests of the shared file in a conventional Bloom filter. Locality filter uses a counting Bloom filter to maintain swarm participants. For updating and verification of filters, it only requires several hash operations. Second, CoVoD discovers potential neighbors' state through a referral, without establishing direct connections. The referral can be easily combined with the existing PEX protocol [39]. It expands a peer's swarm view to a direct neighbor's neighbors (one-hop neighbors). Third, AntiLiar uses a cryptographic hash function for verification

against cheating attacks. It associates a one-time signature and peer ID with hash chaining, which enables a one-time authentication. A digital signature based on a public key is minimally used against fake reporting.

1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows. Chapter 2 introduces terminologies, protocols, and vulnerabilities in BitTorrent systems. Chapter 3 investigates existing approaches related the research problems. Chapter 4 addresses a simple reputation scheme with locality filtering based on resource testing against Sybil attacks in a BitTorrent system. Chapter 5 presents an adaptive neighbor management scheme to prevent selfish peers in a VoD system. Chapter 6 addresses a defense mechanism against cheating attacks in mesh based streaming systems. Finally, Chapter 7 concludes this dissertation and points out some future research directions.

Chapter 2

Background

In this chapter, we give the background knowledge for our schemes. We briefly introduce terminologies, protocols, and vulnerabilities in BitTorrent systems. Then, we briefly describe Bloom filter: a simple space-efficient data structure. Readers are referred to [40] for more details.

2.1 BitTorrent systems

BitTorrent [1] is a P2P content sharing system used to facilitate efficient downloads. It is one of the most common P2P protocols. BitTorrent focuses on efficient content replication, whereas other P2P systems consider an effective search mechanism. BitTorrent can distribute a content file without heavy load on the content provider. Instead of the exclusive provision by the provider, its protocols make peers interested in the content to join an overlay network and to trade the partly downloaded data among the participants. Such cooperations overcome the resource restriction of the provider. Hence, BitTorrent reduces its resource burden sharply and holds scalable peers through their active cooperation.

The current P2P streaming systems are based on BitTorrent protocols due to a practical, cost-effective, and efficient approach for content distribution among participants. There are many BitTorrent-like streaming applications; for example, CoolStreaming [41], PPLive [42], PPStream [43], SopCast [44], and Tribler [45]. Such applications have developed a new web-based television model, called P2PTV, that redistributes video streams in real time on an overlay P2P network. The advent of P2PTV has been significant because it can make any video content available through BitTorrent-based protocols. Since users joining the overlay network simultaneously upload and download stream parts while watching the content, a stream provider can serve scalable users without additional resource costs.

2.1.1 Terminologies

Terminologies are not standardized in BitTorrent systems. For the sake of clarity, we define the terms throughout this dissertation as follows.

- **Torrent :** A *torrent* is a session of transferring a single content. The torrent file is a metadata file of the distributed content file. It contains a file name, a length, SHA-1 values of each piece, and tracker's location information.
- **Tracker :**A *tracker* is the centralized component in BitTorrent systems. The tracker keeps track of all peers in the swarm, helps a peer discover other peers, and helps a newcomer join the swarm.
- **Peer (user) :**The term *peer* is interchangeable to the term *user*. A peer can be either a seeder or a leecher. Each peer is classified by the state of file download. If a peer has an incomplete file, it is a leecher. Otherwise a peer has a complete file, it is a seeder.
- **Seeder :**A *seeder* is a peer that has the complete file and is only uploading to the swarm. The original seeder is the peer that initially offers the file for download.
- **Leecher :** A *leecher* is a peer that is still downloading file pieces in the swarm.
- **Swarm :**A *swarm* is an overlay network for a shared content in which peers share their own portions with other peers. It is composed of the tracker, seeders, and leechers in the virtual network.
- **Piece :** A file is divided into even-sized *pieces* verified by each SHA-1 value in the torrent file.
- **Block :** A piece is divided into even-sized *blocks*. A block is delivered between peers as the transmission unit on the network.
- **Neighbor list :** A peer maintains a list of other peers it establishes TCP connection to. Initially, the tracker informs a newcomer of a randomly selected list, called neighbor list. This is also known as *peer set*.
- **(Un)Interest message :**If peer A has a piece and peer B does not have the piece, peer B sends an *interest* message to peer A. Conversely, when peer B already obtains the piece to the other peer or peer A has a subset of the pieces of peer B, peer B sends an *uninterest* message to peer A.
- **(Un)Choke message :**If peer A decides not to send a piece to peer B, peer A sends a *choke* message to peer B. Oppositely, when peer A decides to send some piece to peer B, peer A sends an *unchoke* message to peer B.
- **Local Rarest First (LRF) :** The local rarest first algorithm is the piece selection strategy. It is also known as the rarest first algorithm.
- **Tit-For-Tat (TFT) :** The TFT scheme is the peer selection strategy. It follows a give and take rule. This is also known as *choking algorithm*.

2.1.2 Overview of BitTorrent protocols

A BitTorrent system consists of a *torrent* website, a *tracker*, *seeders*, and *leechers*; this is collectively referred to as a *swarm*. An original seeder (i.e., a file owner) breaks a file into even-sized pieces (e.g., 256KB per piece), and generates SHA-1 hash values for each piece, for checking data integrity. The torrent website stores information about a file name, its length, the SHA-1, and tracker location information. The tracker is both a bootstrap server and a coordinator informing leechers of potential neighbors. The tracker maintains a list of currently-active leechers in the swarm. Upon an admission request from a newcomer, the tracker provides the newcomer with a list of randomly-selected neighbors. The newcomer becomes a leecher with an incomplete file. The leecher obtains file pieces from other peers in an order determined by the Local Rarest First (LRF) algorithm. Upon completion of file download, the leecher becomes a seeder. Seeders upload file pieces to other peers without requiring reciprocation. To effectively distribute file pieces, a seeder can use a variety of seeding algorithms, such as bandwidth-first, random-based, and round-robin. File pieces are further divided into blocks (e.g., 16KB per block) for exchange purposes.

For fair piece exchange, BitTorrent uses the TFT incentive scheme [29]. In this scheme, a leecher exchanges a message indicating what file pieces each neighbor currently has. If the leecher discovers a neighbor has a piece it requires, it sends an *interest* message to the neighbor having the piece. When two parties both have an interest in pieces the other possesses (i.e., symmetric interests), one of them may select the other as an *unchoke* recipient (allocating an upload slot to upload file pieces) to send the required pieces. A peer repeatedly decides new unchoke peers with the following steps: (i) The peer sets a *choke* state (i.e., does not allocate any upload slot) to all of its neighbors. (ii) The peer sorts interested leechers based on their measured upload rate, and selects the $k-1$ highest neighbors where k is the number of regular unchoke slots. (iii) The peer chokes the lowest uploading leecher of previously-unchoked peers every unchoke period (i.e., 10 seconds), and then unchokes the first interested peer. (iv) The peer randomly selects a new leecher to unchoke from among its neighbors every 30 seconds (i.e., optimistic unchoke).

In BitTorrent, the local rarest first (LRF) algorithm is a piece selection strategy. Each peer maintains a list that contains the number of copies of each piece for local neighbors. A peer sorts numbers of the piece copies in the list as a descending order (the smallest to the largest) except downloaded pieces . Based on the sorted list, the peer selects the next piece to download. If some pieces are the same priority, it randomly picks one of them at the next download selection. The list is updated whenever the number of each piece copy is changed in the local neighbors. In addition, each peer attempts to download the incomplete piece as fast as possible. Since a block is the transmission unit, a piece can be incomplete when an unchoking period is expired. If a peer downloads (requests) at least one block of a piece (i.e., some blocks of a piece are missed), it requests the same piece to the neighbors with the highest priority. Such algorithm helps a peer to minimize the number of partially downloaded pieces.

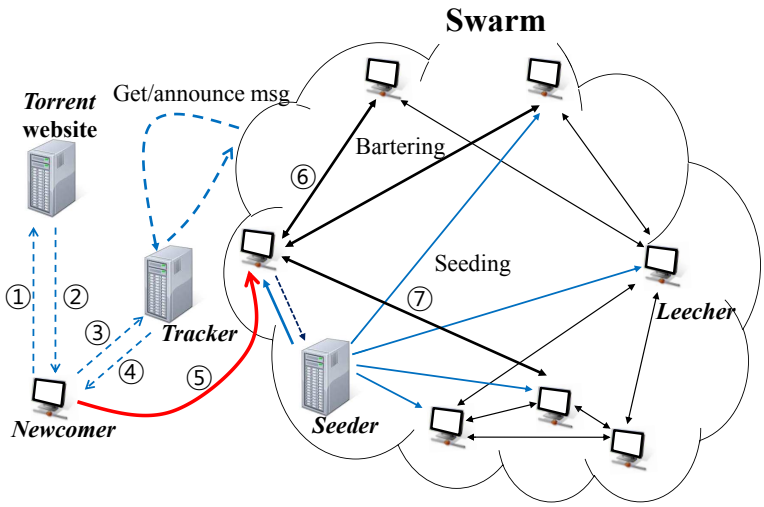


Figure 2.1: Overview of a peer's life cycle in a BitTorrent system

The BitTorrent system has connection management algorithms to handle incoming and outgoing requests. The algorithm for incoming requests is first-come, first-served. A peer continuously listens and accepts a new incoming request. The connection is established if the maximum number of connection slots is not reached. For outgoing connections, the peer finds better candidates according to the TFT scheme.

A peer periodically checks whether its neighbors are dead (no longer responding). If so, the peer removes them from the neighbor list. Each peer queries the tracker to harvest new neighbors in a swarm: (i) when a regular interval expires; (ii) when the neighbor list is insufficient to trade pieces (i.e., typically 30 or fewer); (iii) when the leecher completes the download.

Figure 2.1 shows a peer's life cycle in a BitTorrent system. First, a newcomer requests torrent file of interest from a torrent website (①). The torrent website then replies with the torrent file to the requester (②). Next, the newcomer contacts the tracker using the URL in the torrent file (③). The tracker sends the newcomer a random neighbor list (④). This list approximately contains 30 $\sim$ 80 peers (typically, 50 peers). Afterwards, the newcomer joins the swarm and becomes a leecher (⑤). The leecher establishes TCP connections with local neighbors and requests file pieces. Once getting initial pieces by (optimistical) unchoking from a seeder or other leechers, the leecher can barter the pieces with local neighbors (⑥). Finally, the leecher completes file pieces and becomes a new seeder. Peers periodically contact the tracker to inform their progress and to obtain new neighbors. With these messages, the tracker maintains swarm participants and updates their state.

2.1.3 Vulnerabilities in BitTorrent systems

BitTorrent systems are vulnerable to intentional attackers. Since BitTorrent systems have incorporated few security mechanisms from the early stage, malicious attackers can exploit protocol vulnerabilities to squander for other resources and interfere with normal operations, ultimately leading to the system's collapse. Recent studies [10, 12, 11, 15, 9, 14, 13, 3, 4, 5, 9, 46] have identified various attacks in BitTorrent systems.

BitTorrent does not have a peer authentication mechanism. An attacker can generate a great number of false identities at little cost even though the protocol guides one-to-one relations between a user and an ID. A great number of Sybil nodes with false identities attempt to join in the swarm. Although the tracker is trustworthy, it cannot tell whether or not a joining peer is a Sybil node. Then, the tracker maintains swarm participants compromised by Sybil nodes, and suggests Sybil nodes as potential neighbors whenever a peer requests a new neighbor list [15]. Sybil nodes make it hard to find good bartering peers and to cooperate with other peers.

BitTorrent does not support a message integrity mechanism. A malicious attacker (Byzantine attacker) misleads peers to an equivocal situation with an arbitrary gossip message. If a Byzantine attacker modifies a control message shared with neighbors, they cannot know whether the received message is correct because of no message integrity for gossiping. For example, by using piece under-reporting, an attacker disturbs neighbors from completing a file [9, 8]. Attackers can deceive compliant peers and interrupt them from performing normal operations. The victims hardly manage their goal under continuous Byzantine attacks.

BitTorrent has a weakness of piece-level integrity checking. According to [10, 14, 11], malicious attackers can waste neighbors' download bandwidth and verification (computation) time. An attacker may return some fake blocks to the neighbors who selected the attacker as an unchoking recipient, due to complying with TFT protocol. This bad reward, however, may worsen the impact because of the absence of the defense towards fake blocks. Due to the coarse-grained file piece integrity mechanism (i.e., using SHA1 values of file pieces), verification of fake blocks consumes a non-trivial amount of download bandwidth, reassembly effort, and buffer space, and the victim has to re-download the genuine pieces from other neighbors.

Selfish or malicious peers can aggressively consume peers' network resources (connection slots and upload bandwidth) [10, 12, 3, 4]. They request TCP connections to all peers (especially, seeders) with strategies like the large view exploit [4] and establish connections. The attackers can have more neighbors than compliant peers who have limited connection slots. They exhaust leechers' network resources using rational incentive schemes [3, 5, 9]. If attackers collude with each other or exploit Sybil nodes, they can occupy seeders' connection slots and seeding slots [12]. This attack cost is sending fake control messages and minimal piece messages to hold their interests relation [9], which is trivial. The attacker can make it difficult for peers to connect with other honest traders. The result will be slow

download times and a decrease in cooperation.

2.2 Bloom Filters

Bloom filter [40] is a space-efficient randomized data structure to verify whether an element is in a group. The Bloom filter that maps n elements through k hash functions is a m-bits string. To represent a set $S = \{x_1, x_2, \ldots, x_n\}$, it initially sets all zeroes in the m-bit string. Using k hash functions for n elements, index bits in a filter are marked (set to 1). By doing so, all elements are reflected and the filter abstracts a membership. To verify a member in the group, it is hashed k times, and checked whether k index bits are set to "1". If so, the member must be in the group. Unfortunately, Bloom filters may falsely claim membership for a non-member due to the collision of index values (i.e, *false positives*). Although false positives occur, Bloom filters can hold scalable elements in a small space and verify membership with fast hash functions. Such advantages exceed the weakness when the error rates are controlled fairly low.

In the case of elements in a group that are dynamically changing, a general Bloom filter cannot be applied. Alternatively, counting Bloom filter enables deletion of an element without recreating the filter. Such variation modifies a bit into a $counter$. The index of the counter indicates the number of times a hash function maps to that index. If an element is added in the group, counters of the relevant indexes are added, and if an element is deleted, the relevant counters are subtracted. The additional counters increase the length of Bloom filter. Note that the space overhead can be reduced as in [47].

The drawback of Bloom filters is that it allows false positives even though it does not generate false negatives. The probability of false positives can be derived:

$$FP = (1 - (1 - \frac{1}{n})^{km})^k \approx (1 - e^{\frac{-km}{n}})^k. \tag{2.1}$$

The rate of false positives depends on how many bits are used for indexes, how many elements are in, and how many hash functions are used. To maintain a fixed rate of false positives, the length of bits in the Bloom filter must increase proportional to the number of elements inserted. Given the values of m and n, the rate of false positives in the equation 2.1 can be minimized to $(0.6185)^{m/n}$, when $k = (n/m) \times ln2$. Note that given n elements and a desired false positive rate fp, the required length of bits is

$$m = -\frac{n * ln(fp)}{(ln2)^2}. \tag{2.2}$$

Chapter 3

Survey of Research Problems

In this chapter, we survey existing approaches to address the research problems; Sybil attacks, freeriding, and cheating attacks.

3.1 Sybil Attacks in Distributed Systems

Identity is an important factor to distinguish different users and to trace a user's activities. Most systems have a mechanism to assign a digital identifier to each user. A digital identifier can be established as a unique certificate, email account, log-in account, or randomized strings within the given systems. Based on the sameness between a user and its identity, many security mechanisms are achievable. One of the representative examples is user authentication by a digital signature bound between a public key and an identity. Through user authentication mechanism, malicious behaviors of attackers could be prevented by penalizing the source.

Typically, P2P systems support anonymous communication between peers. Instead of being identified by a central authority, peers are identified by randomized keys such as SHA1 values. For example, DHT-based P2P systems assign peer ID as a hash value and BitTorrent uses peer ID as a 20-byte string through SHA-1 function. BitTorrent currently has no guidelines for creating a new ID. The only requirement for a new identity is to keep the uniqueness.

Anonymous feature in P2P systems can be a double-edged sword. P2P systems do not need a centralized authority for users' identities. Since each peer identity is anonymous or pseudonymous, they can provide the participants with the privacy of activities. Owing to peer anonymity, the interests of P2P systems have dramatically increased throughout the world. Peers can easily share copyrighted files (e.g. music, video, game, and software files) with other peers without revealing a real identity. Such piracies have caused the lawsuit against P2P companies and users. On the other hand, P2P systems cannot provide security mechanisms such as user authentication. Malicious or deliberate attackers can exploit Sybil attack. They can create a great number of false identities and use them to avoid detection

and accusation for their malicious behaviors. Thus, Sybil attackers have a freedom from the penalty of the malicious behaviors in the anonymous P2P systems.

Now, we overview existing approaches against false identities in distributed systems. There are two main categories, which are centralized approaches and decentralized approaches. Both of them have pros and cons against the Sybil attack. Although the centralized approach can prevent Sybil attacks, it does not fit P2P systems. Alternatively, the decentralized approaches can effectively mitigate such attacks.

3.1.1 Centralized Approaches

To exclude Sybil nodes, a central authority can be a solution. Trusted certificates based on public key infrastructures prevent false identities. In addition, monetary based systems require financial expenses to obtain the desired goods, and trace these transactions through a user identity. Such systems can mitigate the nodes with false identities, but cannot prevent the Sybil attack. The standard drawbacks of the centralized approaches, such as overhead, lack of scalability, and reliability, become a huge barrier to be deployed in the P2P systems.

Trusted certificate

Douceur [16] claimed that a trusted certificate is the only solution to eliminate Sybil attacks. A trusted third party can issue trusted certificates for legitimate participants, using public key or identity-based cryptography [48]. The trusted certificate relies on the fact of that each user is uniquely assigned to one identity. The central authority has to ensure the one-to-one relation between a user and an ID. However, it is difficult for a central server to manage the perfect uniqueness of digital identities in practice. The central authority should manages the trusted certificates for all participants, such as granting a new certificate and revoking lost or stolen one. Due to such overheads, system scalability and reliability should be restricted. Even when holding scalable participants, the central authority becomes a bottleneck and a single point of the failure. In addition, this approach sacrifices benefits from anonymous feature in P2P systems.

Monetary-based systems

In monetary based systems, user identity is explicitly used for the payment of desired goods or services. The monetary based approaches additionally adopt a payment mechanism. Such approaches are not susceptible to the Sybil attack because of a digital currency. For example, Karma is a micro currency for P2P applications [49, 35]. Each transaction is monitored by the system and each peer's activities can be controlled based on the amount of virtual money collected by the peer. In addition, Dandelion [37] is another example of a virtual currency in a centralized server. This approach uses symmetric key

cryptography and a trusted server to ensure fair exchanges. However, such systems require a central server or an infrastructure for accounting, billing, or authentication.

3.1.2 Decentralized Approaches

Alternatively, a decentralized approach can detect and mitigate the effect of Sybil attacks. There are three main schemes against false identities in the distributed systems, which are, resource testing, social network, and reputation systems.

Resource testing

Resource testing is based on the fact that a Sybil node has a limited resource. If a user employs multiple Sybil nodes in the system, they have a high probability of fewer resources than a normal user. In order to inspect whether or not a user has an expected resource, this approach includes the examination of computing power, storage capacity, network bandwidth, or a range of IP addresses. However, it may bring about false positives in a environment where nodes have heterogeneous capacities.

Despite inaccurate verification against a Sybil attacker, many studies have been proposed due to a simple and direct validation of the Sybil node in distributed systems. Douceur [16] addressed the issue by using node validation via computation, storage, communication challenges to defeat the Sybil attack. These validations cannot be an ultimate solution but an effective method to mitigate false identities. Newsome et al. [18] investigated Sybil attacks, and addressed several schemes such as a radio resource testing, key verification for random key predistribution, registration, and position verification in sensor network. Although a malicious attacker behaves as if it were a great number of nodes by impersonating other nodes or by claiming false identities, the radio resource testing effectively defeats such unfavorable situations. Rowaihy et al. [19] proposed an admission control to mitigate Sybil attacks by a computation challenge. With client puzzles, a peer attempting to join the network is repeatedly challenged by the peers in the hierarchy. This scheme requires a trusted third party for key-based cryptography and construction overhead of hierarchy tree for the admission control. In addition, Freedman et al. [50] used testing for IP addresses to remove a corrupted domain.

Social networks

The rise of social networks affects daily life, business, and information access in web-based systems. In daily life, many web users get in touch with their online or offline friends via social communities; examples are YouTube, Facebook, Twitter, MySpace, and Cyworld. They often enjoy uploading, watching, commenting, and feedbacking (twittering) about user generated contents and common interests in the web based systems. For effective information access, recommendation services in Google, Amazon, Yahoo, and Netflix offer users improved information based on interests for the better decision. Such exiting networks could be a defense mechanism against Sybil attack. Since it requires human efforts to

establish a trust link, an attacker has few links from sound users to Sybil nodes. If an attacker generates too many false identities, the graph of representing the social network becomes strange.

Recent Sybil defenses, such as SybilGuard [20], SybilLimit [21], and SumUP [22], leverage a social network (trusted network) among the users. Such approaches use a graph property to identify a suspicious link (edge) between a compliant user and a Sybil node. The edge connecting a compliant user and a Sybil node is called an *attack edge*. Social relations are formed by users' interdependence such as friendship, kinship, or common interests (e.g., hobby, dislike, knowledge, beliefs, religion, etc.). Since each edge among users involves a mutual trust, Sybil nodes are unlikely to possess many attack edges in the social network. Removing the attack edges is to disconnect a large number of Sybil attackers. This property is connected to using social network as a central authority against false identities. Unfortunately, approaches based on social networks bring up a cold start problem (i.e. newcomer discrimination) and limits scalability and cooperation.

SybilGuard [20] mitigated corruptive influences of Sybil attacks through a social network among user identities. Edges between two users in the social network indicates a human-established trust relationship. SybilGuard relies on a verifiable random walk in the graph and intersections between such walks to identify attack edges. A small quotient cut (i.e., a small set of attack edges) between the Sybil region and the honest region can remove Sybil nodes. SybilLimit [21] improved the number of Sybil nodes accepted per attack edge compared to the SybilGuard protocol. Such improvement derives from the proposed schemes; using intersection condition on edges, balance condition to deal with escaping tails of the verifier a random route, and benchmarking technique to estimate random routes. SumUp [22] addressed a solution against the corruption in online content voting systems triggered by Sybil nodes. SumUp uses an adaptive maximum flow to limit the number of fake votes cast by Sybil nodes. It assigns and adjusts link capacities in the trust graph to collect the net vote for an object, and leverages user feedback to diminish the corrupted votes of Sybil nodes. Unfortunately, most defenses based on social networks have insufficient mitigation, newcomer discrimination, and scalability problem.

Reputation systems

The reputation system is classified into as peer [32], object [51], and hybrid reputation [52]. Reputation systems form numerical reputation scores of individual peer or object, which is based on pairwise feedbacks from the participants. As peers interact with each other, they record their experiences and develop an opinion for each other. The reputation rating aggregated opinions of the peers helps a user develop trustworthiness of previously unknown peers or objects. For example, a peer can estimate other peers' behaviors with numerical reputation scores (vectors). Well-organized cumulative reputation scores help a peer to make a correct decision despite the distorted reputations of Sybil attackers. Unfortunately, Sybil nodes can mostly counteract such efforts in the reputation systems. Therefore, an effective defense scheme should be considered against possible counteractions of Sybil attackers.

Reputation systems based on peers' transactions and their feedbacks can be an alternative to reduce the impact of Sybil nodes. For example, Marti et al. [24] investigated the efficiency of the reputation approaches either with a central login or with self-managed identities. The reputation with the central login server improves prevention rates but requires a high management cost. On the other hand, self-managed approach provides a higher level of anonymity while preventing identity hijacking, but has a limitation about parameters setting.

SybilProof [23] uses a static graph formulation against false good reputations triggered by Sybil nodes. In the SybilProof scheme, an attacker cannot boost its own reputation by creating a great number of false identities. However, sound peers do not directly affect a Sybil-resilient voting system. Hence, Sybil nodes may have almost as good reputation as the sound peers. Landa et al. [53] proposed a Sybil-proof indirect reciprocity mechanism, called PledgeRoute in P2P networks. PledgeRoute helps peers to contribute resources in the condition of indirect reciprocity among peers. It reduces indirect reciprocity to direct reciprocity, and transfers peer contribution among peers. Sybilproofness is accomplished with an accounting system keeping track of peer contributions and finding maximum bottleneck contributions in a routing graph. It is similar to finding attack edges through maximum flow in social network. DSybil [25] is a defense for reducing the influence of Sybil attackers with the typical voting behaviors of compliant peers. A relatively small number of Sybil nodes can compromise voting results because on average only a small fraction of the compliant peers vote on an object. In order to defend against this effect, DSybil exploits the heavy-tail distribution of the typical voting behavior of the normal peers.

In addition, Piatek et al. [34] attempted to manage persistent incentives across swarms in BitTorrent systems. One-hop reputation uses public/private key pairs for identity, which generates key management overhead and limits scalability and anonymity. Lian et al. [54] evaluated private experience and shared history to achieve the balance of reputation coverage and accuracy. The scheme is vulnerable to a whitewasher (i.e., a kind of Sybil nodes) and imposes penalties on newcomers.

3.2 Freeriding in P2P Systems

Selfish behaviors by *freeriders* (users who obtain resources with little or no contribution) interrupt voluntary cooperations among peers, which is one of well-known problems in P2P systems. According to the recent studies [26, 27, 55, 28], freeriders are prevalent in P2P systems. Zghaibeh et al. [26, 27] showed the population of freeriders reached up to 16.8% in the BitTorrent system. They categorized freeriders into three classes; lucky peers, strategic peers, and cheaters. Strategic peers took the majority of freeriders uploading less than 5% of individual content volume. Although cheaters were 8% of total population of freeriders, they could obtain contents without any contribution. Handurukande et al. [55] observed freeriders in eDonkey were approximately 80% of the total peers. Hughes et al. [28] measured 85% of peers shared no file and 86% only shared 10 or fewer files in Gnutella systems. As a common feature of these measurements, freeriding has been on an increasing trend compared to the previous

studies. Consequently, the burden of non-freeriders will be increased.

Now, we briefly summarize the strengths and weaknesses for anti-freeriding. There have been a number of attempts to tackle freeriding in P2P systems. Anti-freeriding schemes fall into four main categories, which are, direct reciprocity [29, 30, 31, 5, 9, 8], reputation schemes [32, 56, 33, 34], currency-based (monetary-based) schemes [35, 36, 37], and piece modification schemes [38, 57, 9, 13, 14]. In addition, many efforts against freeriding have been proposed focusing on file sharing systems. Unfortunately, the solutions developed for file sharing cannot be directly applied since P2P streaming systems face additional challenges including asymmetric interests among peers and stringent time constraints for the quality of service. Anti-freeriding in the P2P streaming systems is more challengeable than file sharing.

3.2.1 Direct Reciprocity

In reciprocity based schemes, each peer maintains past interactions with other peers in a history, and reflects the information of this history for decision making processes. These schemes are classified into direct reciprocity or indirect reciprocity. In direct reciprocity, a peer with past interactions of directly connected neighbors estimates on the contribution level of the neighbors, and selects better peers to trade desired data. On the other hand, in indirect reciprocity, each peer entrusts decision making process to a mediation system, an intermediate peer, or a set of peers. The indirect schemes can be monetary based systems and reputation systems [1].

Direct reciprocity schemes are mainly applicable for content sharing applications. Each peer can maintain a history of neighbors' contribution that provides adequate opportunities for reciprocation while completing a file download. According to individual local history, peers increase the cooperation of bartering file pieces. Thus, freeriders hardly obtain a free service without the contribution to other peers although one-time cheating and optimistic unchoking are possible in direct reciprocity schemes. TFT scheme [29] and its variations [30, 31, 5, 9, 58, 8] fall within these schemes. Such schemes hold the common merit as a practical and simple approach, but also have weaknesses such as newcomer penalty (cold-start problem) and avoidable ways of freeriders.

TFT scheme [29] is a basic incentive mechanism in BitTorrent. A user's download bandwidth is approximately proportional to its upload bandwidth. Since TFT forces leechers (peers who are still downloading file pieces) to cooperate with each other, freeriders have difficulty to obtain file pieces. However, TFT scheme cannot provide any incentive mechanism between a seeder and a leecher. Seeders provide optimistic unchoking for altruistic donation to leechers, which is a weak point. Freeriders take advantage of optimistic unchoking through the large-view exploit [4, 3]. In addition, Bharambe et al. [31] simulated BitTorrent using a synthetically generated distribution of peer upload capacities.

[1] We subdivide indirect reciprocity schemes into currency-based approaches and reputation-based approaches for more detailed analysis of anti-freeriding.

They proposed a peer selection scheme matching peers with similar bandwidth, and a block level TFT scheme. Similar to the TFT scheme, this approach cannot effectively prevent freeriding.

Piatek et al. [5] proposed a design and implementation of BitTyrant in which a leecher carefully select bartering peers and contribution rates to maximize download per unit of upload bandwidth. To improve the download performance, a leecher allocates upload bandwidth to neighbors, only to the point that stimulates direct reciprocity. In BitTyrant, a leecher does not apply to optimistic unchoking against freeriding. Although BitTyrant outperforms a normal BitTorrent for the aspects of fast completion time and effective upload rates, it does not defeat freeriding. Moreover, BitTyrant discriminates a newcomer obtaining an initial piece currency.

Levin et al. [9] addressed an auction-based approach, called *Prop-share* that allocates upload bandwidth to directly connected peers based on their contributions. Prop-share improves a completion time compared to BitTyrant and BitTorrent. Leechers allocate altruistic uploads to discover new neighbors. Seeders altruistically donate their upload resources without reciprocation. Freeriders, thus, can exploit these vulnerabilities.

Give-to-Get (G2G) [58] introduced a ranking-based selection against freeriding in P2P streaming systems. In G2G, peers give data received from a peer to neighbors to get pieces from the peer. It is similar to TFT incentive scheme based on forwarding of pieces to other participants. However, one-time freeriding is still possible at a middleman. In G2G, a peer as a donator neither designates its recipient's recipients like children in a tree structure nor validates their feedbacks, which is susceptible to Sybil attacks or collusion.

In addition, TrInc [8] uses an additional hardware component that provides a peer with a secure evidence to attest neighbors gossiping. Within using trusted hardware, freeriders cannot cheat other peers to take free pieces. However, it has several concerns; additional hardware cost, privacy (discouraging anonymity), and hardware malfunction.

3.2.2 Reputation Schemes

Reputation systems use trust scores (values) for reflecting peers' behaviors. Trust scores of participants are mapped into variable values by direct and indirect interactions. Such systems can evaluate peers' usefulness, and improve the range for interaction among peers. However, they may produce false positives because of relying on indirect observations compromised by attackers. These approaches, thus, require special mechanisms to monitor and evaluate peers' scoring. Reputation systems, in general, are vulnerable to Sybil attacks, whitewashing, or colluding.

EigenTrust [32] is a typical reputation system in P2P systems. The EigenTrust initially was designed to defeat inauthentic files. It provides each peer with a unique global trust value based on the participants' history. Unfortunately, this scheme has some weaknesses. Deploying a reputation server inherits the drawbacks of the centralized model. Moreover, it is hard to implement because of some is-

sues; selecting pre-trust peers, space overhead, and computation overhead. Kung et al. [56] proposed an admission control system using EigenTrust algorithm. It is capable of providing differentiated services to users based on their past reputations. Admission scheme uses a distributed eigenvector-based method to compute peers' reputations. However, this scheme has an issue preventing freeriders. Freeriders can compromise reputation scores by colluders and false identities, and avoid discrimination through whitewashing.

One hop reputation [34] reduces computation and memory requirements to obtain the benefits of indirect observations. As this method restricts the range of reputation scoring to one hop neighbors, it gains the benefits similar to a global reputation system. However, one hop reputation does not evaluate the effects of freeriding. This scheme penalizes a newcomer through long history and requires public key cryptograph for identity.

3.2.3 Currency-based (Monetary-based) Systems

In currency-based (monetary-based) systems [49, 35, 37], peers pay virtual money to get a specific resource from other peers. Currency based schemes include payment infrastructure, such as accounting, billing, micro-money related activities (creation and deletion), monitoring and authentication. These systems can promote peer cooperation by means of providing resources through payment, comparable to the current traditional monetary system. Peers earn virtual moneys by contributing resources and spend part of the moneys to purchase required file pieces. Since each peer records activities (e.g. resource contribution and consumption), currency-based systems enforce rational or selfish peers to contribute resources. However, currency-based systems in general suffer from scalability by trusted infrastructure. Moreover, they must accommodate payment infrastructure as a precondition.

Karma [49, 35] is a general economic framework for combating freeriders. Once a newcomer joins, it is awarded a seed amount of Karma. Karma values for each peer are maintained by a group of peers, called bank-set, who are collectively responsible for Karma values. Whenever a peer consumes or supplies resources, it uses virtual moneys and keeps track of interactions. Unfortunately, Karma generates huge overheads because it manages each peer's transaction (i.e., whenever a karma is used and interaction between peers occurs). If a freerider exploits Sybil attacks (generating fake identities or whitewashing), it can obtain free initial seed money.

Dandelion [37] is another example of a virtual currency facilitated by means of a centralized authority. Dandelion provides a cooperative paid content distribution protocol to encourage uploading other peers. A peer is motivated to upload to its peers even if the peers do not have content that interests the peer. Dandelion uses symmetric key cryptography against peer's cheating, and a trusted central authority to ensure fair exchange for trading data between peers. However, Dandelion inherits the drawbacks of the central server; scalability problem, single point of failure, and performance bottleneck.

3.2.4 Piece Modification Schemes

BitTorrent systems provide newcomers with an initial currency [2] for free. According to [59], this altruism is the target for freeriders to exploit. Piece modification schemes [38, 57, 9, 13, 14] prevent freeriders from obtaining free pieces, although peers additionally require encryption (or encoding) and decryption (or decoding) steps. However, piece modification schemes are little applicable to real time applications due to the delay caused by encoding and decoding (or encryption and decryption) and a central authority (TTP).

Locher et al. [38] proposed a source coding scheme to protect seeders from freeriding. The seeder divides a file into several blocks, and then computes and uploads a linear combination of a small number k of these pieces. Leechers are required to upload to one another to collect a sufficient number of coded pieces to reconstruct the file. However, this approach focuses on a seeder-side defense mechanism.

Treat-before-Trick (TbeT) [13] makes use of the secret sharing scheme against freeriding. Similar to breaking a file in BitTorrent, the key is divided in to n subkeys. A peer obtaining t subkeys can recover the master key and decrypt the encrypted file pieces. In TbeT, peers must upload the encrypted pieces to obtain a subkey, which prevents freeriding. Unless a freerider contributes a piece, it never receives any subkey. In addition, Prop-share [9] (previously classified to a direct reciprocity approach) also proposed a piece encryption mechanism for preventing altruistically seeding. However, they are vulnerable to key disclosure, and do not fit for P2P streaming applications.

For streaming application, some approaches proposed piece modification schemes. Orchard algorithm [57] uses a multiple description coding with dynamic multiple spanning trees against freeriding in P2P streaming applications. Orchard enforces that each peer has to forward a substream for every substream it wants to receive. However, the performance against freeriding is unclear since it does not include the effects of freerider's strategies. This scheme also inherits the drawbacks of tree-based approaches; tree maintenance and biased costs (intermediate peers are more overloaded than leaf peers). MIS scheme [14] modifies shared pieces using HMAC and streaming server in P2P streaming. This scheme aims the prevention of fake block attacks, not freeriding.

3.3 Byzantine Attacks in Distributed Systems

The term "*Byzantine*" comes from the *Byzantine generals' problem* (Byzantine agreement protocol) [60, 61] which illustrates the pitfalls and design challenges of reaching a consensus among participants by communicating over an unreliable network. In [60], Byzantine commanding general wishes to organize a concreted plan of action; whether to attack or to retreat. Unfortunately, since each lieutenant general's corp is geographically separated by the enemy, the only way for the generals to communicate with one another is sending messengers. The Byzantine corps of generals have been infiltrated by traitors. The

[2] A newcomer needs some pieces to barter other other peers, called bootstrapping pieces.

messages between generals can be lost or modified by traitors, and each general cannot identify whether or not a corresponding general is loyal. Despite this, the loyal generals must reach the same conclusion; either to attack or to retreat by sending messages. The *Byzantine agreement protocol* is an algorithm that addresses this problem successfully. To tackle this problem, there are several challenges; unreliable network, message integrity, and user authentication.

Byzantine fault indicates an arbitrary fault in distributed systems. Byzantine faults cause unexpected failures; for example, crash failure, response or request omission, incorrect process, incorrect result, and corrupted state. In general, client server models must resist arbitrary faults in order to furnish reliable services to users for satisfaction and profit. Some services, such as a banking service included in distributed computing and database, must be reliable and available because service failure and downtime would be disastrous. If there is no mechanism to handle Byzantine faults, systems cannot guarantee a reliable service. On the other hand, P2P systems are somewhat tolerable to unintended failures (e.g. node crash, network error, or hardware malfunction), but are vulnerable to deliberate attacks. *Byzantine attackers* (liars) mislead victims to an equivocal situation with an arbitrary gossip message in the P2P systems. False messages can easily distort the information of making a peer's decision. If an attacker attempts to modify or drop messages shared with other peers, victims' information will be unconsciously compromised. Due to no verification mechanism of whether a message received from the sender is correct, the victims cannot achieve their goal.

To cope with such failures, Byzantine fault tolerant (BFT) mechanisms have been addressed in distributed systems. There are two main trends; the one is classic BFT schemes (i.e. *state machine replication* [62, 63, 64, 65, 66, 67, 68]), and the other is practical BFT schemes (trusted hardwares [69, 8] and mounting evidence [70, 71]).

3.3.1 Classic Approaches

The state machine replication schemes [63, 64, 65, 66, 67, 68] are a classic BFT approach. Typically, this scheme is used for a set of servers to keep the same state for data integrity and correct computation in distributed systems. The key idea is keeping a global correct view among replicas (servers). Due to asynchronous environments[3] in distributed systems, servers must hold causal ordering of events ("*happen-before*" relation of events) for the consistency. To accomplish this, each server maintains its own process state. A leader elected in the servers delivers replicas (servers) an event requested by a client. If they accept the event and reach the same result, their state will be changed. The state machine replication schemes are divided into *replica-based approaches* [63, 64, 65, 68] in which replicas use a reliable commit protocol (e.g. three-phase commit protocol) and *quorum-based approaches* [66, 67] in which replicas use a majority rule. Unfortunately, these approaches cannot directly apply P2P systems because of the quadratic cost of replica communication. A great number of peers cause huge communi-

[3]It means nodes are connected by a network that may fail to deliver messages, delay them, duplicate them, or deliver them out of order.

cation and space (memory) overhead to handle events. Dynamic membership changes (churn) makes it more difficult to deploy a classic BFT approach.

Table 3.1 shows properties in Byzantine solutions, in which f indicates the number of fault replicas. Table includes a practical Byzantine fault tolerant (PBFT) approach [64], Query/Update (Q/U) [66], and Hybrid-Quorum (HQ) [67]. To maintain a consistent view PBFT and HQ require $3f$+1 servers, while Q/U requires $5f$+1 servers with respect to the total number of replicas. Theoretically the lower bound for the total number of replicas is $3f$+1. Although Q/U requires more replicas to tolerate a Byzantine fault, it decreases agreement latency. Unlike 3-phase steps in PBFT and HQ, Q/U can achieve a consistent view with 2 steps through majority voting. Note that the number of messages for consensus depends on the number of replicas. To achieve the same view, the communication cost of additional messages among replicas increases quadratically.

Table 3.1: Properties in Byzantine solutions

Property	Practical BFT	Query/Update	Hybrid-Quorum	Lower bound
Total servers	$3f$+1	$5f$+1	$3f$+1	$3f$+1
Agreement latency	3	1	3 (1)	-

Replica-based approaches

Castro et al. [63, 64] proposed a practical three-phase commit algorithm in which nodes (servers) achieve Byzantine agreement in asynchronous environments. It uses an efficient authentication scheme based on message authentication codes (MACs) during normal operation. Public key cryptography is only used when there are faults. The proposed algorithm provides *safety*[4] and *liveness*[5]. The safety property guarantees that all non-faulty replicas agree on a total order for the execution of requests despite failures. The view-change protocol provides liveness by allowing the system to make progress when the primary fails. Yin et al. [65] improved replication costs and privacy by separating agreements that order requests from executions that process requests. Separating agreement from execution for BFT services makes it possible to tolerate faults in up to half of state replicas as well as to protect confidential information. However, this scheme still retains the quadratic communication costs.

Kotla et al. [68] proposed a speculative BFT protocol, called Zyzzyva, that reduces the cost and simplifies the design of BFT state machine replication. For example, a client sends request to the primary server (leader), and then the leader forwards ordered request to replicas. All replicas optimistically

[4]In other words, it is called to linearizability. If a request with sequence number n and history h_n completes, then any request that completes with a higher sequence number $n' \geq n$ has a history $h_{n'}$ that includes h_n as a prefix.

[5]It means that clients eventually receive replicas to their requests.

execute the ordered request, and respond immediately to the client. If the client receives $3f+1$ matching responses (where f is the number of faults ranging zero to the total number of servers), it completes the request. Since replicas respond to a client's request without three-phased commit protocol, they can become temporarily inconsistent. Clients detect such inconsistent states among replicas, help correct replicas converge on a total ordering, and only rely on responses that are consistent with this total order. Unfortunately, if a client is malicious or faulty, Zyzzyva's improvement may be questionable.

Quorum-based approaches

Abd-El-Malek et al. [66] proposed a Query/Update protocol (Q/U) that provides an operation-based interface for a fault-scalable service. Q/U objects export interfaces comprised of deterministic methods; a query does not modify objects but an update does. In Q/U protocol, a client directly contracts replicas and optimistically executes operations. It only requires one round to reach a consistent view when no contention and no failures. However, Q/U has two shortcomings; First, it requires a large number of replicas ($5f + 1$). This number is considerably higher than the theoretical minimum of $3f + 1$. This weakness requires additional replicas' placement and increases the number of possible points of failure. Second, Q/U protocol may perform poorly when there is contention among concurrent write operations, since it requires back-off time to resolve the contention.

Cowling et al. [67] proposed a Hybrid-Quorum replication (HQ) protocol. In the absence of contention of clients' requests, HQ uses a lightweight quorum protocol. To manage a consistent view, reads require one round communication between a client and the replicas, and writes require two round trips. If a contention aries, HQ employs a replica-based approach to agree on a proposed ordering of requests. Although they tried to keep the advantages of two approaches, their scheme increases the system complexity, and such complexity becomes a barrier to adopt in the distributed system.

3.3.2 Practical Approaches

P2P systems are tolerable to crash failure and omission failures because of redundant resources. Unfortunately, P2P systems are vulnerable to malicious attacks in which the systems cannot foster cooperation to achieve common goals by sharing resources. Unlike classic BFT approaches, the goal of these approaches is defeating Byzantine attacks for reliable services of scalable participants rather than keeping the strict requirement of the agreement.

Trusted hardware

An alternative approach is deploying a trusted hardware for attestation against equivocal behaviors [69, 8]. A trusted component improves the previous approach for the aspects of communication overhead and system scalability for Byzantine agreement. This approach delegates a security mechanism against arbitrary faults to a hardware module. Trusted components help each participant to attest whether or

not other nodes are accurate. Although they do not require additional online communication between machines, the precondition of deploying a trusted device can be barriers; for example, user consensus of using a trusted hardware, privacy concerns, and standardization of trusted hardwares.

Chun et al. [69] proposed a trusted module called Attested Append-only Memory (A2M) that is a hardware facility for the improvement of BFT protocols in distributed systems. A2M provides a particular abstraction of a trusted log which leads to protocol designs against equivocation. It has a set of trusted, undeniable, and ordered logs. Each entry in the log consists of a log-specific sequence number and an incremental cryptographic digest. Whenever interaction occurs, A2M creates and appends a signed entry in the log. Equivocal messages generated by Byzantine attackers (malicious servers) can be detected and removed by secure digests. A2M reduces agreement overhead without need for inter-replica communication.

Similar to A2M, Trusted Incrementer (TrInc) [8] is a trusted hardware that prevents Byzantine attackers triggering equivocal behaviors. TrInc provides a primitive for unique one-time attestation in distributed systems. Each participant delegates a security mechanism to a trusted component (TrInc) that holds a non-decreasing counter and a hidden cryptographic key. This combination (monotonic counter and a key) provides a smaller abstraction than the trust log of A2M. TrInc has two attestations binding a message to a counter with a key. Advance attestation only moves to a new state from the previous state whenever appending an entry in the log. Status attestation represents the most current state without advancing a counter. TrInc has several desirable properties; it immediately keeps consistency, it does not need additional on-line assumptions, and it improves communication overhead per protocol message to $O(1)$.

Unfortunately, such hardware-based approaches have several issues. First, an additional hardware requirement is a barrier. To use this approach, all users must equip an additional hardware component. Second, trusted security chip is also vulnerable to attacks [72]. Malfunction of trusted components compromised by an attacker may lead to overall system crash. Third, privacy issue is not free to this approach. Trusted hardware manufacturers may have a privilege that is capable of privacy invasion. For example, iPhone location service against privacy was highly controversial [73].

Mounting evidence

A mounting evidence (*auditing or accountability*) schemes is used for monitoring. This scheme records users' interactions (operations) using a cryptography mechanism. Each user can detect a Byzantine attack and defeat liars' strategies, through a private evidence and a verifiers' assistant. Such approaches, however, have several considerations: costs, detection delay, and monitoring peer (witness or guard).

PeerReview [70] maintains tamper-evident records that provide non-repudiable evidence of node actions. With PeerReview, a correct node can eventually detect faults from a faulty node, thereby defending against false accusation. The purpose of the tampered-evidence is to detect equivocation em-

bedded in messages recorded in a log. Each node only keeps a full copy of its own log. Nodes exchange enough information to determine whether or not another node has a fault. Tamper-evident logs with a commit protocol ensure that each node keeps a consistent log. For the detection of a malicious attacker, each node is associated with a small set of other nodes and acts as a witness node. Witnesses nodes collect evidence about others, check their correctness, and make the results available to other nodes. However, PeerReview does not provide fault tolerance from Byzantine attackers. The detection delay of PeerReview may generate additional space overhead and temporarily create inconsistent tree paths in the log.

Ho et al. [71] address a practical transformation scheme, called Nysiad, that extends a crash tolerant model to a Byzantine fault tolerant model. Nysiad assigns a set of guards (i.e., the same role of witness peers in [70]) to each peer. The guards validate the messages sent by their associated peers, and keep a consistent state through a gossip protocol. This scheme, however, requires a central server (called Olympus) that maintains guards in the systems. Hence, this server can be a single point of failure and a bottleneck as a result of frequent member churn (i.e., dynamic membership) in P2P systems.

3.3.3 Existing Approaches against Cheating Attacks in P2P Systems

For the defense of Byzantine attacks, SecureStream [6] uses an admission control in which each peer monitors other peers and accuses an attacker triggering misbehavior. A streaming source notifies data transmission to neighbors in every round. Whenever obtaining new information, peers send a notification to their neighbors. This gossip method defeats omission attacks. In addition, an auditing mechanism that records peers' contribution prevents freerider strategies. Local auditors periodically evaluate the contribution of each of their neighbors. Unfortunately, this approach requires a special neighbor management mechanism (Fireflies), and mainly considers freeriders' strategies as attack models in live streaming systems.

Li et al. [7] proposed a **B**yzantine-**A**ltruistic-**R**ational (**BAR**) gossip against arbitrarily malicious attacks. BAR gossip relies on a random neighbor selection and a fair exchange mechanism. Peer selection is based on verifiable pseudo-randomness, which is robust to neighbor selection attack, freeriding, collusion, and Sybil attacks. For streaming data propagation, BAR gossip uses two main functions; *balanced exchange* and *optimistic push*. Peers exchange their pieces with balanced exchange, and optimistically push others (optimistic unchoking). This incentive mechanism is very similar to the TFT scheme in the BitTorrent protocol. Unfortunately, BAR gossip requires a centralized authority for identity management, which limits scalability and anonymity. The central server will be the single point and bottleneck. Moreover, this scheme only supports a static membership system and does not support newcomers after streaming.

There are various routing schemes, such as indexing in Napster, flooding in Gnutella, and hashing in DHT-based systems (Kedemlia, Chord, Pastry, and etc.). DHT-based searching is becoming more

common than the others in P2P systems. Each peer has a partial view of contents and participants in DHT-based systems. If peer's views are stale or incorrect by dynamic churn, fake identities, and false updates, peer discovery using a DHT-based approach does not work well. Young et al. [74] proposed a quorum based technique against Byzantine adversary in such systems. The quorums are sets of peers for helping a peer to overcome Byzantine attacks through a majority action. This scheme uses (n, t) threshold cryptography and distributed key generation (DKG) that reduces communication costs and prevents DHT systems from experiencing malicious Byzantine attacks. Instead of TTP, (n,t)-DKG protocol allows a set of n nodes to construct the secret. However, this scheme is vulnerable to Sybil attacks. Adversary can break a quorum-based scheme using scalable Sybil identities.

Chapter 4

Defending Against Sybil Attacks in BitTorrent Systems

4.1 Introduction

Peer-to-Peer (P2P) systems account for a major portion of Internet traffic in recent years. The P2P paradigm enables a wide range of applications to operate as scalable network services; examples are file sharing, VoIP, and media streaming. The BitTorrent protocol [1], is one of the most popular approaches to P2P file-sharing. This protocol encourages maximum peer cooperation to distribute files. BitTorrent-like systems, such as Vuze (Azurues), uTorrent, BitComet, Tribler, and PPLive, contributed more than 50% of all P2P traffic, and roughly one third of all Internet traffic, in 2008/2009 [2].

P2P systems in general are quite robust to failures, and adapt readily to rapidly-changing conditions. Unfortunately, systems based on BitTorrent are vulnerable to deliberate attacks by determined adversaries [15, 10, 14, 9]. This is because BitTorrent incorporates few security mechanisms, or mechanisms that are only partly effective. For instance, although the BitTorrent protocol includes coarse-grained data integrity checking (i.e., a SHA-1 hash image per piece), it is highly vulnerable to contamination by fine-grained data pollution (uploading of fake blocks) [14]. Dhungel et al. [11] showed that even one polluter in a channel can degrade a streaming service severely in PPLive (i.e., a BitTorrent-like streaming application). As another example, attackers are able to hinder a compliant peer from exchanging data with potential neighbors by fake control messages [9]. In addition, attackers can exhaust legitimate peer's upload bandwidth [12, 13].

Defending against attacks on P2P systems is made more difficult by the fact that one attacker can generate a great number of false identities at little cost; this is known as the Sybil attack [16]. The Sybil attack is a fundamental and pervasive problem in P2P systems. Attackers can use these identities to avoid detection, and to avoid repercussions for their malicious behavior. Since victims cannot differentiate Sybil attackers (Sybil nodes) from legitimate peers, it is difficult for a peer to avoid the above-mentioned

attacks. Therefore, prevention or mitigation of Sybil attacks is key to making systems such as BitTorrent more robust.

The goal of this chapter is to mitigate the impact of the malicious behaviors of Sybils in a BitTorrent system. Sybil nodes aggressively attempt to compromise the swarm, disseminate polluted (corrupted) file pieces, and exhaust peer resources. To address these problems, we propose a light-weight reputation scheme, called good leecher friends (*GOLF*), combined with locality filtering. GOLF detects polluted file blocks through a light-weight, fine-grained integrity check. Peers using GOLF share information with each other about attackers. This information is weighted by their history of previous, mutually-successful exchanges. So, peers can learn about and avoid attackers. Locality filtering flags possible Sybil nodes, based on similarities in their IPv4 addresses. The BitTorrent tracker maintains a *locality filter* that classifies participants. This filter is updated when a peer joins or leaves the swarm, and is distributed to seeders by the tracker.

The primary aim of this chapter is to mitigate the malicious impact from Sybil nodes through peer cooperation, in a way that is lightweight, and easily integrated with BitTorrent. We have implemented the proposed scheme, which sharply reduces the impact from Sybil nodes. For example, the bandwidth cost is reduced by a factor of 10 in the presence of Sybil nodes. Comparison to other reputation schemes shows GOLF has an effective detection under false accusation.

We summarize the properties of the proposed scheme below.

- *Defending Sybil attacks:* The proposed scheme significantly mitigates the malicious impact from Sybil nodes. To penalize active attackers, GOLF uses a simple reputation scheme through peer cooperation. It detects a local attacker using block-level integrity checks, and delivers the warning to bartering friends. As long as each peer cooperates with others, it can protect itself from potential attackers. Locality filtering tentatively identifies Sybil nodes based on patterns in IP addresses. A locality filter detects abnormal membership, which helps an effective distribution and a good neighbor selection.

- *Practical approach:* The proposed scheme is technically feasible to implement in a BitTorrent system. To deploy the proposed scheme, the original seeder and the tracker additionally manage filters (i.e., block and locality filter). The original seeder generates a block filter, and adds it in torrent metadata. Once receiving the torrent metadata, each peer can verify block-level integrity. The tracker updates a locality filter for swarm membership. The locality filter fills participants in bins based on their IP address ranges, and selects them. Unlike Blacklisting scheme [75], it has neither startup delay to import blacklists nor discrimination of benign peers in blacklist IP ranges.

- *Simple cooperative reputation scheme:* GOLF is a decentralized approach. It does not require a central authority for a reputation system (e.g., EigenTrust [32]). According to a private bartering history, each peer determines neighbors' trust, and self-organizes them. Although a peer has a limited view, peer cooperation expands its view, and mitigates the impact from Sybil attackers.

Similar to TFT incentive [29], GOLF scheme reinforces a cooperative environment based on direct good transactions between peers.

- *Cost-effective scheme:* The proposed scheme is cost-effective. Since the tracker is a centralized server for swarm membership, it maintains a locality filter that has a space-efficient data structure. The tracker does not require a costly audit to judge peer behaviors. Each peer with a block filter can check a fake block through hash operations.

The remainder of this chapter is organized as follows. Section 4.2 briefly compares and describes existing methods of Sybil attack prevention. Section 4.3 describes our assumptions, and relevant attacks by Sybil nodes. Section 4.4 details GOLF scheme and locality filtering. Section 4.5 presents the results of simulating the proposed scheme. Section 4.6 analyzes the overhead and security issues, and section 4.7 summarizes this chapter.

4.2 Related Work

Douceur [16] introduced the Sybil attack in distributed systems. To exclude Sybil nodes, a central authority can be a solution. A trusted third party (TTP) can issue certificates for authorized participants, using public key or identity-based cryptography [48]. This approach has the standard drawbacks of a centralized infrastructure (overhead, lack of scalability and reliability), as well as a requiring a sacrifice of anonymity. A system that charges for IDs can mitigate (but not prevent) the Sybil attack. The drawback is that barriers to entry discourage wide participation and cooperation.

Decentralized approaches, such as resource testing [18, 19], trusted networks [20, 21, 22], and reputation [23, 24, 25] are alternative defenses against the Sybil attack. Resource testing based on the fact a Sybil node has a limited resource may bring about false positives in a environment where nodes have heterogeneous capacities. Yu et al. [20] showed that a trusted network (i.e., a social network) can mitigate the effects of Sybil nodes. Use of a trusted network may however incur cold start problems (i.e. newcomer discrimination), increase reliance on a separate infrastructure, and limit scalability. In contrast, GOLF adds little overhead and has no cold start problem. Sybilproof [23] considers Sybil strategies where a user is only concerned with raising his own reputation. GOLF, by comparison, reduces the impact of Sybil nodes engaging in "badmouthing" (i.e., false accusation).

Our threat models reflect existing and potential attacks by Sybil nodes. Konarth et al. [15] described the ability of liars (Sybil nodes) to attack a swarm by false announcements. Dhungel et al. [11] measured the impact of the fake block attack in PPLive. Levin et al. [9] introduced a strategic behavior of increasing interest periods (i.e. under-reporting) to consume connection slots. In [12], attackers could exhaust seeder's bandwidth and connection slots.

Piatek et al. [34] attempted to achieve persistent incentives *across swarms* in BitTorrent systems. This one-hop reputation uses public/private key pairs for identity, which generates key management

overhead as well as limits scalability and anonymity. Instead of a public key infrastructure, the GOLF scheme uses a direct reputation based on bartering volume *in a swarm*. The proposed scheme leverages an anonymous identity and scalable users. Lian et al. [54] evaluated private experience and shared history to achieve a balance of reputation coverage and accuracy compared to EigenTrust [32]. Since such schemes are vulnerable to whitewashing (a type of Sybil attack) and collusion, GOLF addresses such concerns.

Rowaihy et al. [19] reduced Sybil attacks with an admission control scheme that uses a client puzzle with public key cryptography in a DHT. Their scheme requires a trusted third party, artificial barriers to entry, and the overhead of constructing a hierarchy. Our scheme imposes no such barriers, adds only a small overhead to BitTorrent, and does not require a trusted third party. Sun et al. [76] investigated the effect of using Sybil nodes as a freeriding strategy. GOLF handles various malicious behaviors by Sybil nodes. MIS scheme [14] detects a fake block (pollution) attack through HMAC in P2P streaming. Although GOLF has the similarity of a block-level detection, our scheme use a filter-based detection through blocks image, and does not require server's intervention to identify a real polluter.

The blacklisting approach [75] excludes IP address ranges of the attackers. SafePeer plugin, a blacklist approach, requires a delay of between 2 and 20 minutes to import a database of blacklisted IP addresses [77]. This drawback has limited usage of the SafePeer plugin. Also, it may mistakenly reject some benign peers in blacklisted IP address ranges. Different from blacklisting, locality filtering uses an adaptive filter to keep track of suspicious IP ranges, and has no startup delay. The locality filtering does not block newcomers, even if they happen to occur in a suspicious IP range.

The rest of this chapter describes a fully distributed scheme for dealing effectively with content pollution and Sybil attacks. There is no penalty for newcomers (cold-start problem), and no sacrifice of anonymity. There is no reliance on a public key infrastructure, or on a trusted third party (other than the use of a tracker, which is a standard part of the BitTorrent protocol). There is no startup delay. The scheme uses direct reputation evidence based on bartering volume in a swarm, and the effects of bad-mouthing and collusion are considered. Careful attention is given to the use of space-and communication-efficient encoding of information.

4.3 Assumptions & Threat Models

In this section, we describe our assumptions and threat models. Sybil nodes can engage in several malicious behaviors, including connection slot, bandwidth, fake block, and swarm poison attacks.

4.3.1 Assumptions

We consider basic BitTorrent components. We rely upon the tracker and the torrent website to provide correct information, and to be available (methods of fail-over and redundancy are known and used).

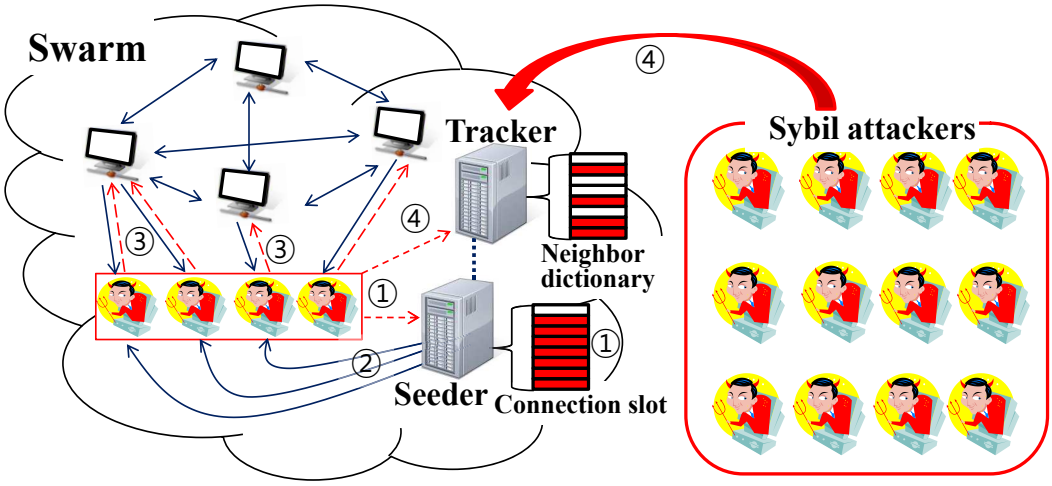


Figure 4.1: Overview of malicious behaviors from Sybil nodes.

There is no central authority or trusted third party for peer authentication. Therefore, no peer can tell whether a peer identity has been faked, and all participants are assumed to be legitimate (non-malicious). A seeder can adapt different seeding algorithms to distribute file pieces to leechers. Each leecher follows the rate-based TFT unchoking and LRF piece selection schemes.

We assume that malicious nodes can act individually, or together (in collusion with one another). An individual node has limited resources but is able to generate fake identities. A determined adversary can create a large number of Sybil nodes and effectively control them. Adversaries can easily use a NAT on a single network to create lots of virtual addresses. We believe it is considerably easier to create effective Sybil nodes in limited address ranges. Note that in [75], determined attackers are located in small network ranges, and our measurement study supports such property in 4.5.2.

4.3.2 Threat Models

Leechers may experience the effects of malicious behavior by Sybil nodes during piece exchange [10, 11, 9]. Malicious peers will cheat the seeder and the tracker [15, 12]. Figure 4.1 shows Sybil nodes can annoy participants with the following attacks.

Connection slot attack (①)

Sybil nodes can aggressively request TCP connections to consume limited connection slots. Once established, the Sybil node can send its neighbors (seeders and leechers) fake control messages to hold their interest state. Although the cost of the control messages sent to neighbors is trivial, the attack can make it difficult for non-malicious peers to connect with other benign neighbors. The result will be slow download times and a decrease in cooperation.

Bandwidth attack (②)

Sybil nodes may attempt to greedily consume the upload bandwidth of a seeder. In the event that Sybil nodes occupy most of the unchoke slots of the seeder, benign leechers may be starved (unable to download file pieces from the seeder). In addition, a Sybil node connecting with a benign peer may receive a considerable portion of the upload bandwidth of that peer.

Fake block attack (③)

Sybil nodes may send fake blocks to neighbors, to waste their download bandwidth and verification (computation) time. A Sybil node may initially appear to be complying with the TFT protocol. Due to the coarse-grained file piece integrity mechanism (i.e., using SHA1 values of file pieces), verification of fake blocks consumes a non-trivial amount of download bandwidth, reassembly effort, and buffer space, and the victim has to re-download the genuine pieces from other neighbors.

Swarm poisoning (④)

Malicious nodes create fake (Sybil) IDs and attempt to join a swarm. While the tracker may be trustworthy, it cannot discriminate whether a joining peer is malicious without attack evidence. The tracker may therefore suggest Sybil nodes as potential neighbors whenever it is requested to provide neighbor lists.

4.4 GOLF scheme & Locality Filtering

In this section, we present a simple reputation scheme, GOod Leecher Friends (GOLF), with locality filtering. The ultimate aim is to mitigate malicious attacks from Sybil nodes. Leechers cooperates with direct neighbors to combat Sybil nodes. The tracker and seeders minimizes the impact through filtering Sybil nodes. The GOLF scheme enables a leecher to detect potential attackers by sharing its experiences with direct neighbors. Locality filtering helps the tracker and seeders to discriminate against Sybil nodes, using an efficient data structure for the purpose. Our filter based schemes (block filtering and locality filtering) are shown by the Figure 4.3.

4.4.1 GOLF Scheme

The goal of GOLF is diminishing the effect of attackers. GOLF relies upon cooperation among leechers. To identify the possible Sybil nodes, each leecher uses a filter-based detection mechanism. GOLF expands the local view of attackers to immediate neighbors by exchanging information about past behavior. The local trust value is based on previous TFT volume, and the detection of corrupted blocks.

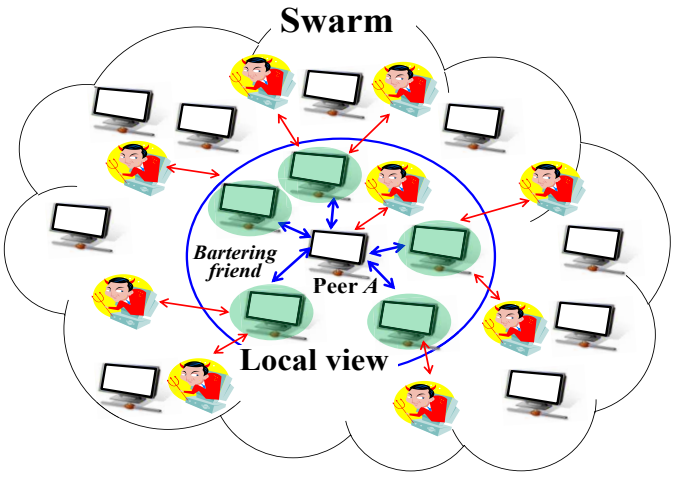


Figure 4.2: Overview of GOLF scheme

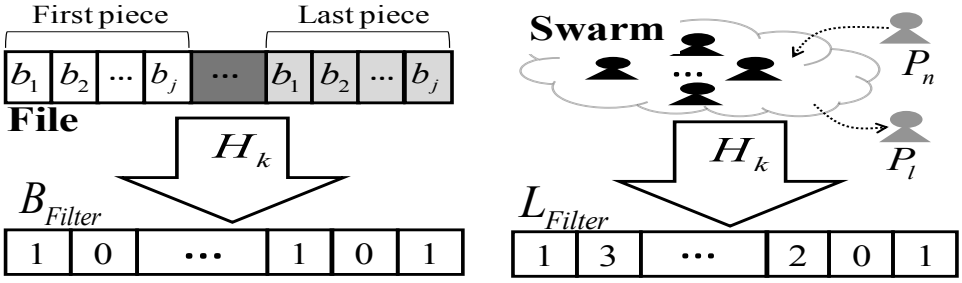
Figure 4.2 shows the overview of GOLF scheme. Once triggered a fake block attack from a Sybil attacker, peer A identifies the ID of the local Sybil attacker. The peer A disconnects the connection of the attacker, and records it in the attacker history. Then, A propagates the information to its good local neighbors that are properly bartered in the past and are currently establishing a connection. To exclude them as a potential neighbor candidate, the bartering friends mark it into the attacker history according to GOLF reputation. A also updates a potential attacker information from reputation values by bartering friends. Consequently, each leecher can exclude potential attackers from a neighbor set.

GOLF protocol

GOLF is based on good interactions, or exchanges of legitimate (non-corrupted) blocks between neighbors. If a neighbor interacts successfully and properly, the leecher regards the neighbor as a "friend". Otherwise, the leecher records the neighbor's ID and misbehavior in its attack history. The leecher will refuse connection requests from previously-misbehaving peers. The leecher propagates information about attackers to its direct neighbors, who can use that information in making their own decisions. Consequently, the gossip between friends can exclude potential attackers from connecting.

Block filter against fake block attack

Sybil attackers can directly annoy leechers through fake blocks. Checking data integrity by SHA-1 signature of each piece prevents leechers from accepting a corrupted pieces. Nevertheless, Sybil attackers can easily make them squander their resources by fake blocks. The fact of a piece is additionally broken into blocks gives a weak point to leechers. For instance, Sybil attackers may upload only a fake block instead a whole piece, or fake blocks across pieces in rewards for TFT. Before downloading residual



(a) Block filter creation (b) Locality filter update

Figure 4.3: The original seeder creates a block filter B_{Filter} with all blocks in the file and the tracker updates a locality filter L_{Filter} with swarm participants. In (a), each piece is divided into even-size blocks (b_1, b_2, ..., b_j). Each block is hashed k times, and reflected to B_{Filter}. In (b), peer P_n indicates a newcomer and peer P_l indicates a leaver.

blocks, a leecher keeps the corrupted block. If the leecher downloads the rest of blocks from other peers, it will not know which peer(s) provided false blocks.

To tackle this problem, a *block filter* (B_{Filter}) based on Bloom filtering [40] is used. The block filter is a summary of all blocks in the shared file. Figure 4.3(a) shows the creation steps for B_{Filter}. The original seeder hashes each block in the file with k hash functions, and marks the corresponding k bits in the filter. After processing all blocks, the seeder adds this B_{Filter} to the *torrent* metadata. After obtaining the *torrent* file, leechers do not need to download it again when they rejoin the swarm. Although the size of B_{Filter} in the torrent metadata depends on the number of blocks and the expected rate of false positives, it is very small relative to the size of most files being shared; detailed overhead costs are analyzed in 4.6. Additionally, unlike MIS scheme through HMAC and server's intervention [14], filter-based detection enables each leecher to directly identify a real attacker (polluter).

Attacker detection using Block filter

GOLF uses B_{Filter} to counter the fake block attack. Upon obtaining B_{Filter}, leechers can check block integrity. Verification of a block involves repeating the hash functions and checking that the expected k bits in the filter are set. Integrity checking can then be done on individual blocks, rather than solely at the file piece level. Failure to be verified by the block filter indicates the block is corrupted, while successful verification means that the entire piece must still be downloaded and verified (via the SHA-1 hash). A leecher receiving a fake block, or a corrupted file piece, can set a flag indicating this neighbor is unreliable (assumed malicious). Each leecher independently maintains a history of attacks or misbehavior, based on its own direct interactions with other peers. Naturally, each leecher will prefer to cooperate with good leecher friends.

Countering False Accusations

Sybil nodes may provide false information to their neighbors concerning their experiences. This has to be considered in the choice of information to use in assessing potential attackers. A Sybil node may falsely accuse a benign peer of malicious behavior. In order to reduce the effect of false accusations, trust is first based on individual (private) experience. The notation for this computation is summarized in Table 4.1.

Table 4.1: Notations for GOLF scheme

N	the number of local neighbors
D_i^t	total downloaded volume of genuine blocks from peer i until # of rechoke times t
U_i^t	total uploaded volume to peer i until # of rechoke times t
C_i^t	contribution value of peer i until # of rechoke times t
B_i^t	bartering fraction of peer i in N neighbors until # of rechoke times t
I_i^t	interaction value of peer i until # of rechoke times t
T_i^t	trust value of peer i until # of rechoke times t
$\Delta_{i \to j}^t$	suspicion value reported by peer i about peer j at rechoke period t
S_k^t	suspicion value of other peers k until # of rechoke times t

A peer records the *contribution value* of each of its local neighbors until the number of rechoke times t, denoted by C_i^t, where $0 \leq C_i^t \leq 1$ at every rechoke interval. The peer respectively computes the contribution values of each of directly-connected neighbors i as

$$C_i^t \ = \ \frac{D_i^t}{U_i^t + D_i^t} \tag{4.1}$$

Note that symmetric exchange between peers has a contribution value of .5. For asymmetric exchange, the contribution value will be adjusted as volume uploaded and downloaded to directly-connected neighobrs.

The bartering fraction of a neighbor i of a peer having N neighbors is simply

$$B_i^t \ = \ \frac{D_i^t}{\sum_{j=1}^{N} D_j^t} \ * \ C_i^t \tag{4.2}$$

A peer computes the *interaction value* I_i^t of each of its neighbors i as the product of its bartering fraction and contribution value, i.e.,

$$I_i^t \ = \ \frac{D_i^t}{\sum_{j=1}^{N} D_j^t} \ * \ C_i^t \tag{4.3}$$

The interaction value can range from 0 (minimum interaction) to 1 (maximum interaction), and represents the importance of a neighbor. A neighbor uploading only a small amount of the total of received blocks, or downloading much more than uploading will have a small interaction value.

The *trust value* of a neighbor i, denoted as T_i^t, is computed as

$$T_i^t = \frac{I_i^t}{\sum_{j=1}^{N} I_j^t} \tag{4.4}$$

The trust values represent the opinion of a peer about the neighbors with which it has directly bartered file pieces.

A peer will compute a *suspicion value* S_k^t for other peers k based on the history of its direct interactions, and information reported by other peers. This value ranges from 0 (not suspected of being malicious) to 1 (known to be malicious). If a peer has directly experienced an attack by neighbor i at rechoke period t, $S_i^{t'}$ will be set to 1 for all $t' \geq t$.

Peers exchange their suspicion values with each other, and use this reputation information to update their own suspicion values. A suspicion value reported by peer i about peer j at rechoke period t is denoted as $\Delta_{i \to j}^t$. Upon receiving this reported suspicion value, a peer updates its own suspicion value S_j^t as

$$S_j^t = [\frac{S_j^{t-1} \times (t-1) + T_i^t \times \Delta_{i \to j}^t}{t}] - T_j^t \tag{4.5}$$

The term inside the square brackets in equation 4.5 represents the average degree of suspicion for peer j, while T_j^t reduces this according to the trust directly earned by j. The suspicion value is calculated for neighbors and for peers for which Δ values are received.

A peer makes an independent judgement about other peers, based on the received suspicion values, and stored trust values earned by successful interactions with its neighbors. Since the number of neighbors decides possible bartering ranges for the swarm, the threshold for the suspicion value is set as a fraction of the number of connection slots. A high trust value based on direct experience diminishes the effect of other peers' prejudices against a neighbor. Each peer suspends the decision about whether to suspect a neighbor (to reduce a hasty judgement) until the provider of suspicion information has correctly bartered at least some minimum number of pieces. A malicious attacker will attempt to influence the suspicion value of a benign peer. False accusations correspond to inaccurate high suspicion values. In the following section, the impact of strategic Sybil nodes that attempt to compromise reputation information is considered.

4.4.2 Locality Filtering

Locality filtering reduces network resource exhaustion and swarm poisoning through IP *binning*. In this approach, a bin represents peers who share the same IPv4 /24 IP address prefix (e.g., 10.9.8.6 and

10.9.8.7 share the same /24 prefix, 10.9.8.6 and 10.9.5.6 do not). The tracker groups participants with the same IP /24 prefix using a *locality filter*. A locality-tracking algorithm helps a peer avoid Sybil nodes, thereby preserving connection slots for benign leechers.

The likelihood of attackers within a small network range is relatively high [75]. The measurement study in 4.5.2 details a quantitative analysis. Based on the study, the locality filtering determines and suggests a threshold value.

Tracking Algorithm

The tracker is charged with monitoring membership / participation in the swarm. The L_{Filter} is an implementation of a counting Bloom filter [47]. As shown in Figure 4.3(b), the tracker maintains a L_{Filter} that reflects a snapshot of current participants. The set of participants can be (and usually is) frequently changing; the tracker updates the L_{Filter} by Algorithm 1 whenever a peer joins or leaves. Each peer reports its state to the tracker at regular intervals in the normal BitTorrent protocol. For example, when a newcomer joins, the tracker hashes its IP /24 prefix using k hash functions, and adds 1 to each resulting index value (counter). Conversely, if a known peer leaves the swarm, the tracker decreases the corresponding k index values. The tracker shares L_{Filter} with seeders at regular intervals.

Algorithm 1 Join and leave event in locality filter

L_{Filter} : Locality filter clustering IP/24 prefix
$H_1, H_2, ..., H_k$: k hash functions

1: $IP_{prefix} \leftarrow$ IP/24 prefix of a newcomer (leaver);
2: $Index_i = H_i(IP_{Prefix})$; where i = 1, ..., k
3: $L_{Filter}[Index_i] \pm = 1;$ (newcomer: $+$, leaver: $-$)

Algorithm 2 shows the neighbor selection method in GOLF. Locality tracking algorithm uses the locality filter to select neighbors in different IP /24 address ranges. The tracker provides the requestor with suggestions for neighbors until it has sent a sufficient number. The tracker randomly selects a candidate neighbor. The tracker checks the /24 prefix of this candidate using the locality filter. If the number of peers in the swarm having the same prefix exceeds the "threshold" of suspicion, and one of them has already been suggested as a neighbor, the tracker will discard the candidate. Otherwise, the tracker adds the candidate to a new neighbor list, and sends the list when it has been filled to the new leecher.

Algorithm 2 Locality-based Tracking Algorithm

δ_I : # of informing neighbors (default: 50)
τ : Threshold for suspicious IP prefix
L_{ns} : List of randomly selected neighbors
CPs : Currently active Peers in the swarm
SP : Selected active Peer in the swarm
$N_{IP/24}$: # of peers in an IP/24

1: **if** $|CPs| < \delta_I$ **then**
2: $L_{ns} = L_{ns} \bigcup \{CPs\}$
3: **else**
4: **while** $\delta_I > 0$ **do**
5: $SP \leftarrow RandomSelect(CPs)$;
6: $IP_{prefix} \leftarrow$ IP/24 prefix of SP;
7: $Index_i = H_i(IP_{Prefix})$; where i = 1, ..., k
8: $N_{IP/24} \leftarrow MIN(L_{Filter}[Index_i])$;
9: **if** $N_{IP/24} > \tau$ and $IP_{Prefix} \in L_{ns}$ **then**
10: $Continue$;
11: **else**
12: δ_I -= 1;
13: $L_{ns} = L_{ns} \bigcup \{SP\}$;
14: **end if**
15: **end while**
16: **end if**

Seeding Algorithm

In order to alleviate network resource exhaustion from Sybil nodes, a seeder uses L_{Filter} for effective unchoke allocation. If Sybil nodes take a majority of unchoke slots, benign leechers will potentially suffer data starvation. Locality seeding is helpful in reducing the abnormal selection of Sybil nodes. Such seeding operates similarly to locality tracking. Requesting peers are sorted by some metric, such as download rate, random selection, or service priority. In this order, the seeder checks the requesting peer's IP /24 prefix against the L_{Filter}. If the count for this address prefix is less than a threshold value, the seeder will assign the next unchoke slot to the requesting peer. Otherwise, the seeder will move on (in order) to the next candidate.

4.5 Evaluation

This section presents a trace measurement and the results of applying the GOLF algorithms. The goal is to understand the performance of the proposed scheme against malicious behavior by Sybil nodes. The experimental setup is first described, followed by the results. Note that in order not to bother real BitTorrent systems, we reflect the results of simulations. This is because the above-mentioned various attacks can make troubles in BitTorrent systems.

4.5.1 Experimental Setup

We developed two BitTorrent simulators; one is a faithful implementation of the BitTorrent protocol, while the other modifies BitTorrent by adding GOLF, as described above. The simulators are event-driven, and include events such as joins and leaves, bartering pieces, unchoking (including optimistic unchoking), and exchange of piece messages. TFT and LRF policies were implemented. Sybil actions such as sending fake blocks, discarding received data from leechers, consuming seeders' bandwidth, and making false accusations were also implemented.

In both simulators, some fractions of the nodes were assumed to be Sybil nodes; the exact fraction is described for each experiment. Peer addresses, except for Sybil nodes, were for the most part located in different /24 address ranges. This assumption is also consistent with measurements described in 4.5.2. A random delay was added when sending a piece to all peers. According to [78], the volume of the control messages in BitTorrent is negligible compared to piece messages. Thus, we do not reflect the delays due to control messages. To reduce simulation complexity, the network was assumed to have no bottlenecks or transmission errors. Each peer had an asymmetrical bandwidth capacity that reflects the ADSL standard models [79]. Every peer had between 500Kbps and 1.3Mbps for an upload rate. The original seeder had 5Mbps as its upload rate.

The locality tracking algorithm was implemented in the tracker module. Although the simulator included three different seeding algorithms for leecher selection by the seeders, the results showed

Table 4.2: Number of peers per IP/24

# Peers	Day 1		Day 2		Day 3		Day 4		Day 5	
1	13,306	96.2%	5,049	96.0%	3,451	97.0%	2,624	97.1%	2,230	97.4%
2	439	3.2%	184	3.4%	81	2.3%	60	2.2%	38	1.7%
3	46	0.3%	18	0.3%	11	0.3%	8	0.3%	1	0.0%
4	16	0.1%	3	0.1%	0	0.0%	3	0.1%	1	0.0%
≥ 5	31	0.2%	8	0.2%	13	0.4%	6	0.2%	19	0.8%

similar patterns. To make clear, the evaluation results based on round-robin (RR) seeding algorithm only detailed in this section. RR seeding algorithm sorted leechers based on their service priority (i.e., leechers receiving the least were given the highest priority). The seeding algorithm with locality filtering is denoted in CRR.

The number of peers was limited to 1,000 nodes, based on a previous measurement study [80]. Each simulation started with one seeder and one tracker. They served all participants in the swarm throughout the simulation. Peers joined the swarm based on an arrival process derived from a real BitTorrent trace log [81]. Once a leecher downloaded the entire file, it became a seeder until it left the swarm. To explore malicious attacks from Sybil nodes, the fraction of Sybil nodes was varied from 5% to 50%. File sizes were set between 5 MB to 500MB (exact values reported in each experiment). These sizes mimic typical files in BitTorrent. A simulation run finished when all benign peers completed the file download. Each simulation was run 30 times to compute 95% confidence intervals.

4.5.2 Measurement Study with RedHat9

We analyzed the distribution of IPv4 addresses of leechers in a RedHat9 (1.77GB) trace [81]. The trace reflects downloads over a period of 5 months, and has all events from the tracker of the torrent. The log contains report time, IP address, port number, peer ID, upload and download size, and events. Results are presented for the distribution of IPv4 addresses during the first 5 days of flash crowd events, which are particularly challenging for file sharing systems.

Table 4.2 shows the number of peers per /24 prefix. At least 96% of leechers were in a /24 address range with no other leechers present. Address ranges with 4 or fewer leechers present accounted for 99.6% of all leechers. Accordingly, in the following a suspicious threshold of 5 was used to identify potential Sybil nodes.

The Figure 4.4 shows a cumulative distribution function for peer duration. To download RedHat9 image, the average completion time required 8.1 hours for single session and 19.2 hours for multi-sessions. Leechers trend to have more dynamic memberships than seeders. Due to seeders' long durations Sybil nodes easily seek seeders, and are likely to exhaust the resources of them.

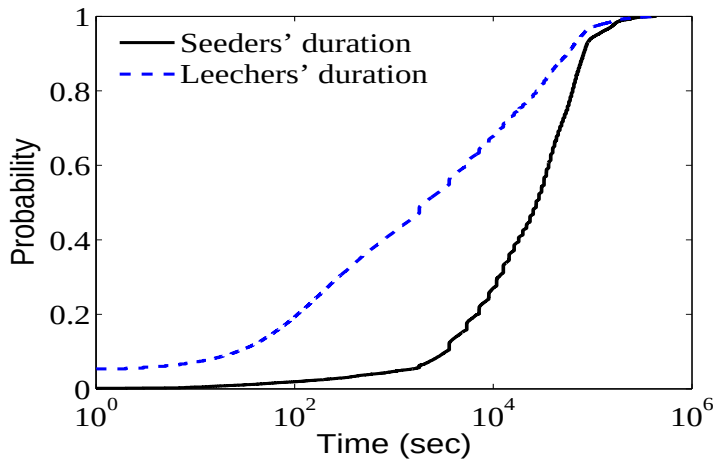


Figure 4.4: CDF of peers' duration for RedHat9 trace during the first 5 days. X-axis is in log scale.

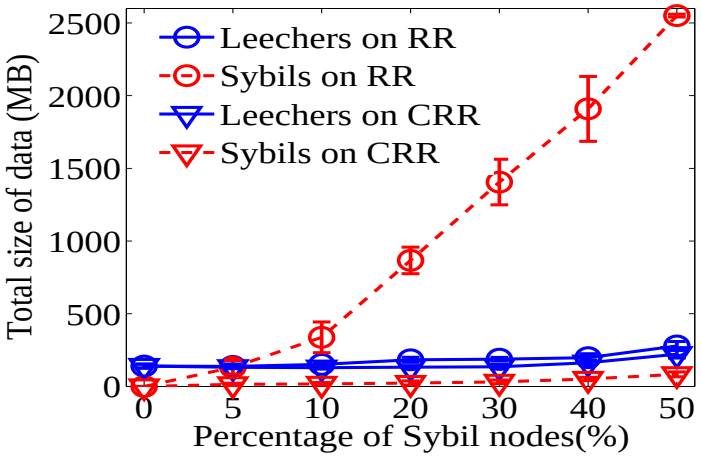


Figure 4.5: Seeder impact of bandwidth attack, which represents the total amount of data sent by the seeder. The total number of nodes is 1,000 and the content size is 5MB. *X*-axis indicates Sybils' percentage out of total nodes. I-shaped lines indicate 95% confidence. Round-robin (RR) seeding algorithm is used, and the countermeasure of RR (CRR) indicates with locality filtering.

4.5.3 Experimental Results

Seeder Impact

In the first experiment, the seeder was required to distribute 5MB of content to all benign users. The total bandwidth required in order to achieve this included bandwidth that was wasted on malicious (Sybil) nodes. Figure 4.5 shows the total amount of data sent by the seeder for RR seeding policy. Performance was measured without and with locality filtering.

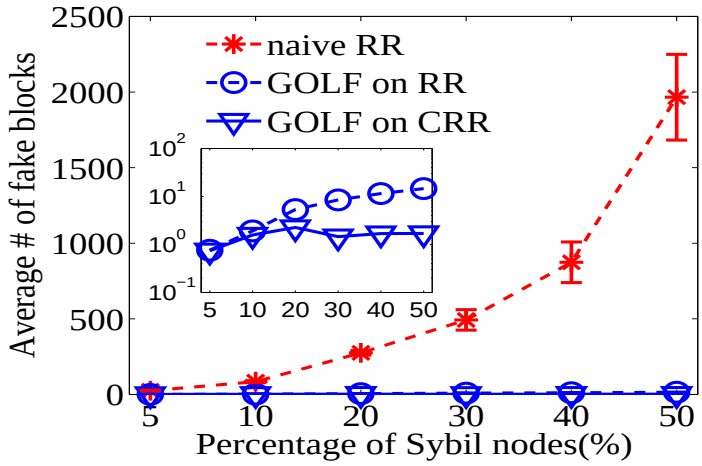


Figure 4.6: Impact of fake block attack for 100MB, which shows the average number of downloaded fake blocks per leecher until file completion. Inner graph details the result of GOLF with(out) locality filtering by Y-log scale.

Comparing the figure, it may be seen that locality filtering greatly reduces the impact of Sybil nodes. The bandwidth consumed by Sybil nodes is decreased substantially by locality filtering, by a factor of 10 or greater if the Sybil node percentage exceeds 10%. This is because the filter helps the seeder allot most unchoke slots to benign leechers, not Sybil nodes.

Benign User Impact

The second experiment evaluated the average rate of downloading fake blocks per leecher, in a swarm sharing a file of size 100MB. Figure 4.6 shows the results. The RR seeding (without locality filtering) exponentially increases the average access rate of fake blocks with increasing Sybil node percentage. However, the proposed scheme (locality filtering + GOLF) decreased the downloading of fake blocks to almost zero. This is because each leecher discriminates against direct and reported attackers using GOLF.

In this experiment, each leecher connected to random neighbors and unchoked them according to the TFT policy. Figure 4.7 shows the average percentage of connecting and selection (unchoking), according to the portion that are Sybil nodes. With no attackers, on average each leecher, over the course of its download, had 115 neighbors and exchanged file pieces with 36 of these neighbors. Sybil nodes were successful in consuming a large portion of the unchoke slots for the normal BitTorrent operation and the RR seeding policy. Using control messages related to piece exchange (i.e., Bitmap and Have message), Sybil nodes kept the interest state to benign leechers. They deprived leechers of the chance to connect and barter benign neighbors. The use of GOLF and locality filtering decreased the fraction of unchoke slots allocated to Sybil nodes by as much as a factor of 10, when Sybil nodes were

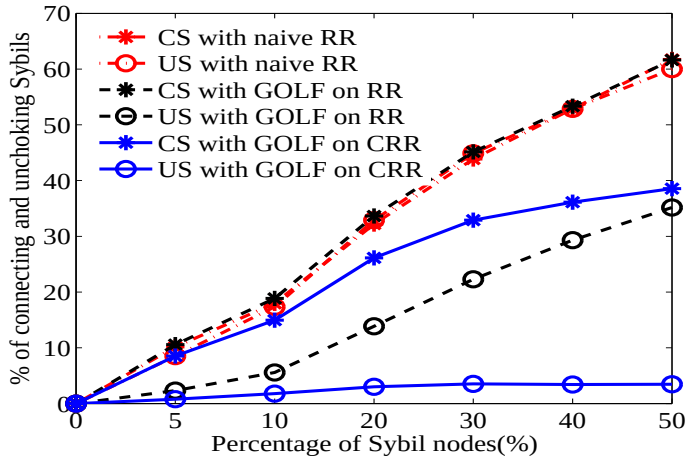


Figure 4.7: Average percentage of connecting Sybil nodes (CS) and unchoking Sybil nodes (US) for 100MB.

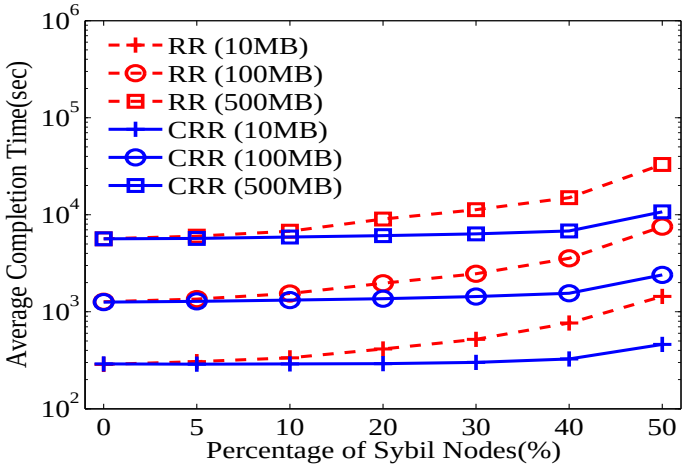


Figure 4.8: Impact on completion time, which shows the average completion time for benign leechers to download the file for different sizes. *Y*-axis is in log scale.

50% of the population of leechers.

Completion time

The next experiment investigated the average completion time for benign leechers to download the entire file, for varying file sizes. The results are shown in Figure 4.8. BitTorrent without locality filtering showed exponential increases as the percent of Sybil nodes increased. This is because Sybils occupy unchoke slots of benign peers, reducing the opportunities for benign peers to exchange file pieces with

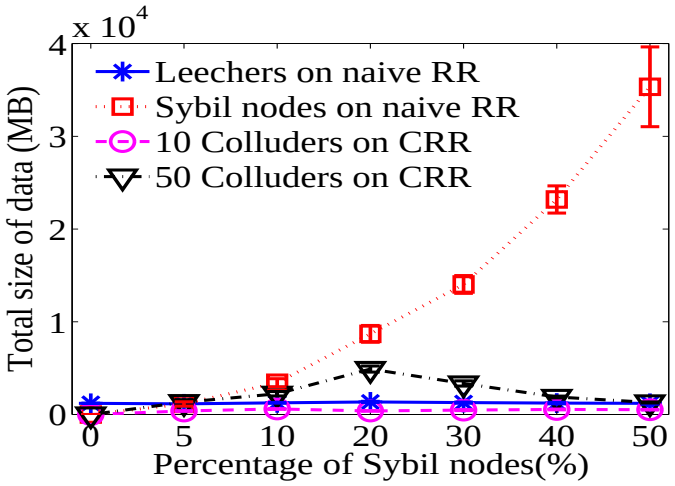


Figure 4.9: Collusion effect, which shows the total size of data sent by the seeder for 100MB. Each colluder who controls a group of Sybil nodes is located in distinct IP/24 prefixes.

one another. In contrast, the use of locality filtering resulted in near constant download completion times, regardless of the fraction of Sybil nodes. GOLF and locality filtering improved the average completion time about 6 times in 50% Sybil nodes.

Collusion effect

Another experiment investigated the impact of collusion among attackers. In this scenario, Sybil nodes were distributed among multiple IP /24 prefixes. The number of distinct prefixes is referred to here as the number of colluders, and was varied. The percentage of Sybil nodes was also varied.

The occurrence of collusion had little impact on the download completion time of benign users, and is such, is not shown. The seeder, however, was affected by the number of colluders. Figure 4.9 shows these results. The seeder upload bandwidth ("size of data sent" in the figure) increased exponentially as a function of the percent of Sybil nodes without the use of locality filtering. Collusion also affected BitTorrent with locality filtering, until the number of Sybil nodes per /24 address range exceeded the suspicion threshold. Thereafter locality filtering greatly reduced the waste of seeder bandwidth (by a factor of 30 or greater for 50% Sybil nodes). An attacker who is able to spread their Sybil nodes throughout the network will obviously have more impact, but at a significantly higher cost.

Attacker Detection Coverage of GOLF scheme

BitTorrent with TFT is limited in its view. GOLF is intended to disseminate knowledge of attackers slightly more widely, but with limited overhead (no non-neighbor communication or global coordination required). In the next experiment, the effectiveness of GOLF in identifying Sybil nodes was measured.

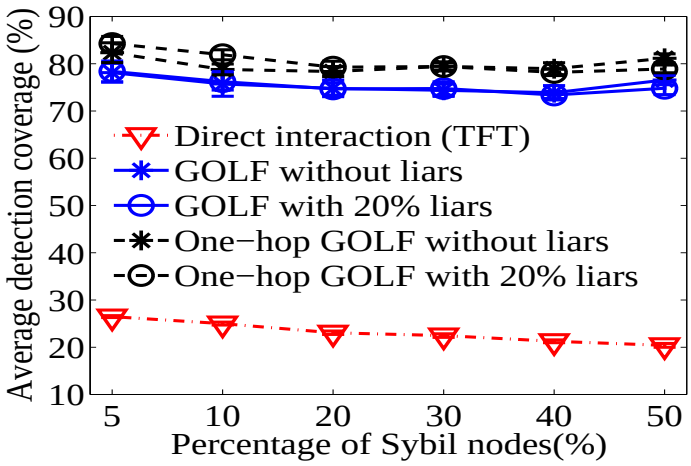


Figure 4.10: Attacker Detection Coverage, which shows the comparison of probability of detecting local Sybil nodes for 100MB. Direct interaction is through TFT neighbors [29], GOLF is through local neighbors, and one-hop GOLF is through local neighbors and their neighbors [34].

The results are shown in Figure 4.10 as the probability of (correctly) detecting attackers. In this figure, three cases are considered: (1) attackers are detected only by direct experience (i.e. TFT) [29], (2) attackers are detected based by direct experience or information provided by immediate neighbors, or (3) attackers are detected based on direct experience, immediate neighbors, and their neighbors (i.e., one-hop neighbors) [34].

The use of information from immediate neighbors, weighted by their suspiciousness, results in a three-fold increase in the likelihood of detecting Sybil nodes, from about 25% to over 75%. The use of information from One-hop neighbors provided additional benefits in this experiment. On the contrary, one-hop GOLF incurs uncertainty and complexity about one-hop neighbors' trust. In addition, the liars compromised peers' attacker history. Some peers mistakenly rejected connection requests from benign peers. Therefore, at most 6.76% peers never completed downloading the file because of false information. The impact of liars is described in more detail in the following section.

Comparison to EigenTrust for false accusation

Generally, reputation systems are vulnerable to false information. To compare GOLF scheme, EigenTrust [32] was used from an aspect of attacker detection as the change from inauthentic files to Sybil nodes. Although EigenTrust was proposed to discourage inauthentic files in unstructured P2P network, the term of *a file* in EigenTrust could be representative of a block (i.e., a transmission unit in BitTorrent) or a peer. Trust in EigenTrust reflects global and local matrix multiplication. The global vector is liable to be compromised by badmouthing from malicious attackers. Although a local trust value was clean, a peer might mistakenly block a connection request from a benign peer. Similarly, Sybil attackers could

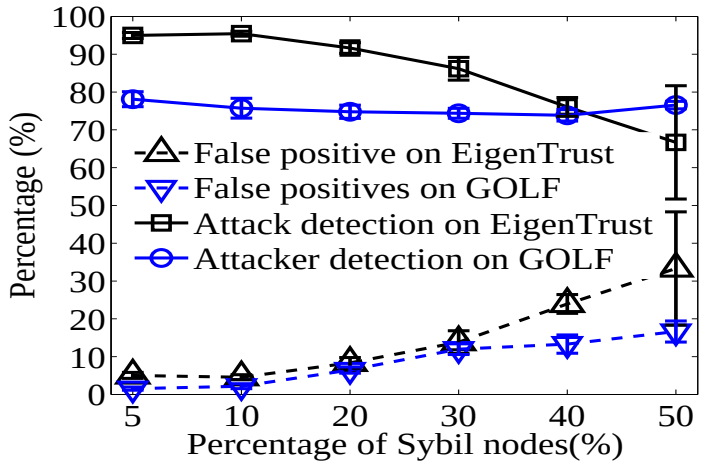


Figure 4.11: GOLF vs. EigenTrust [32], which shows the comparison of the probability of detecting Sybil attackers and falsely accusing benign peers for 100MB.

make falsely accusations about other peers in GOLF scheme.

For the comparison, we set simulation parameters and a scenario. The parameter settings were adjusted into the identical conditions between GOLF and EigenTrust. The fraction of attackers in total 1,000 peers was varied from 5% to 50%, and the number of blocks for 100MB file was 6,400. A peer attempts to request required blocks from other peers according to the trust by the matrix computation for global and local values. A provider (i.e., a peer who is a local neighbor, and is to unchoke to the requester) supplies the requester with a block as long as available rechoke periods. The process from an actual request to a feedback is called as a transaction. After a transaction, a peer evaluates the block obtained and the provider's trust. If it is a fake block, the peer will clean it up, and feedback a negative value to its neighbors for the block owner.

The last experiment compared to EigenTrust about detection rates and false positive under false accusations. Figure 4.11 shows the result. For the aspect of false accusation, the false positives of GOLF scheme are the lower than EigenTrust. The percentage of falsely rejected peers out of the total peers ranged from 1.5% (for 5% sybil nodes) to 16% (for 50% sybil nodes). For attacker detection coverage, EigenTrust is the better. To detect attackers, EignTrust requires pre-trusted peers and triggers communication as well as computation overhead. Whereas, GOLF scheme uses a simple computation by empirical piece interactions in a distributed manner.

4.6 Discussion

In this section, we analyze the overhead for deploying GOLF with locality filtering. After that, we discuss adversary reactions to GOLF, and the issue of false positives.

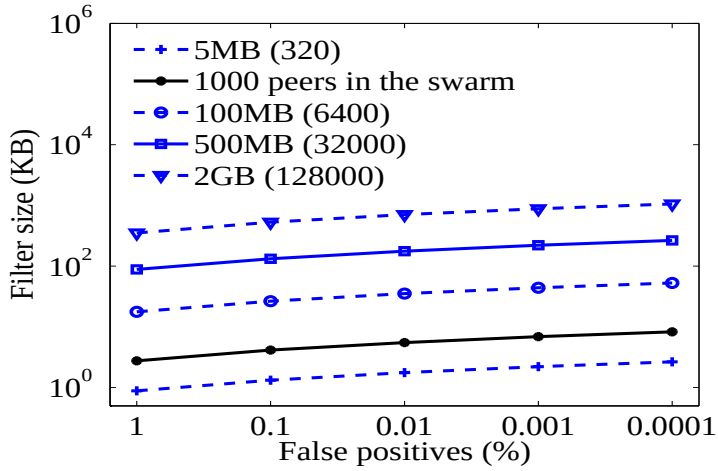


Figure 4.12: Space overhead of block filter and locality filter by the number of elements. The numbers in parentheses indicates elements.

4.6.1 Deployment Overhead

To deploy the proposed scheme, additional costs are incurred. The size of the block filter depends on file size and the size of locality filter depends on the number of participants. We propose the use of Bloom filters, which are well-known space-efficient data structures that are easily updated. The computation overhead requires multiple hash operations to compare the values. In Figure 4.12, the block filter for 2GB content uses 1 MB and the locality filter for 1,000 peers in the swarm uses 8KB. As participants in the swarm change, the tracker updates a counting filter to track locality. The information sent to the seeder needs not be the entire counting filter. To reduce size overhead, the tracker can inform the seeder of locality violations using a much small Bloom filter.

Communication overhead is due to the need to share the block filter, the locality filter, and attacker information. The block filter shared among peers can be included in the torrent file that is already downloaded. The tracker can inform a seeder of a locality filter at a regular interval, which does not require additional communications. Attacker reports can be combined with existing control messages.

4.6.2 Adversary Strategies against GOLF

Generally, reputation systems are vulnerable to counter strategies. For example, Sybil nodes may be liars (make false accusations about other peers), traitors (engage in productive exchanges before providing false information about other peers), or whitewashers (in case of accusation, leave and rejoin with a new identity). The effect of false accusations is mitigated by the weighting by trust (inversely, suspiciousness). This is counteracted by a Sybil node acting as a traitor. Whitewashers with the same IP can be blocked, which is also counteracted by IP anonymization [82, 50].

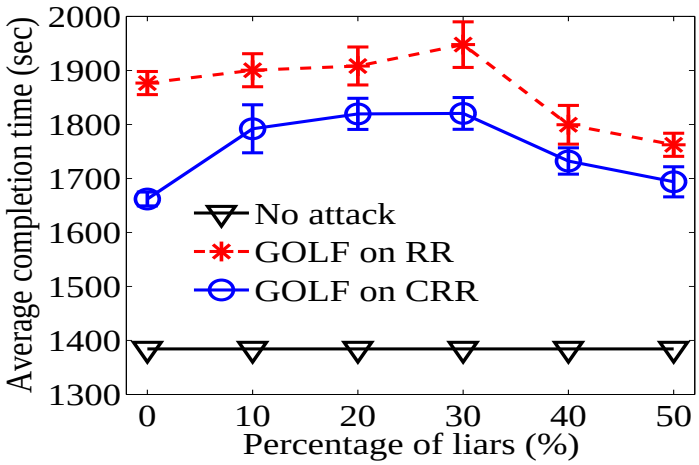


Figure 4.13: Impact of false information, showing the average completion time of benign users, for a 100MB file, as a function of the fraction of Sybil nodes that lie. In this experiment, 20% of the nodes are Sybil nodes.

Figure 4.13 shows the average completion time of benign users as a function of the number of neighbors that lie. The completion time increases about 500 seconds compared to the no attack case. This is because the reports of liars are reflected to benign neighbors and are propagated to their friends. Adjusting the computation of trust to reduce the effects of liars and traitors remains as future work.

Sybil nodes can silently consume resources of benign peers. Such attacks [9, 4, 3] take advantage of cheating to other peers. Sybil nodes connect the more number of peers than normal, send false control messages to create interests, and keep choking state to them. Since Sybil nodes do not trigger direct damage, their intentions are little revealed. In such situations, benign peers cannot prune their connections without awareness. Alternatively, an anti-freeriding scheme [13] can be a solution.

4.6.3 False positives

False positives may occur because of the innocent existence of benign peers in the same /24 address range. For instance, a number of benign peers behind NATs are falsely identified as a Sybil node. They may experience very slow download speeds because of the discrimination of locality seeding. In spite of the delay of getting initial currency (i.e., 4 file pieces), locality tracking algorithm can help the peers overcome seeder's discrimination.

Figure 4.14 shows the effect of false positives by locality filtering. It compares to the average completion times on CRR model for 100MB, based on setting a priori the detection category of each peer. In this figure, a benign node's completion time is not affected much, regardless of whether it is suspected or not. The false positives (i.e., benign peers behind NATs) delay around two minutes on the average.

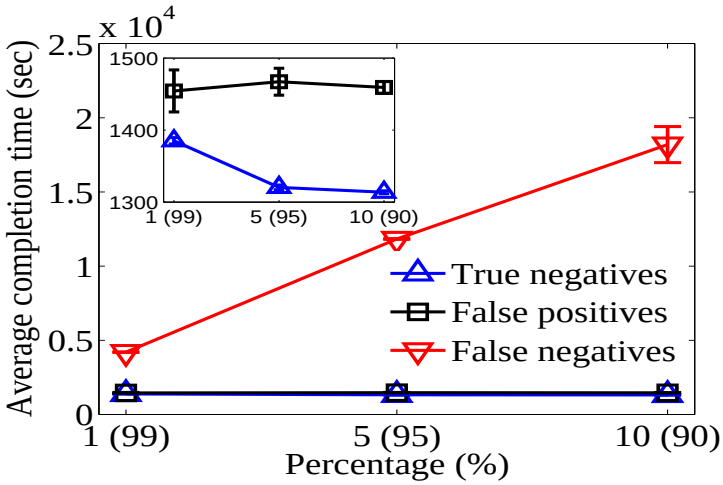


Figure 4.14: Effect of false positives by locality filtering. It shows the average completion times on CRR model for 100MB. The fraction of nodes who were benign, and not suspected of being Sybil nodes (true negatives), was varied from 99% to 95% to 90% . The fraction of nodes who were benign, but (incorrectly) suspected of being Sybil nodes (false positives), was correspondingly varied from 1% to 5% to 10%. The fraction of nodes who were Sybil nodes, but not suspected (false negatives), was also varied from 1% to 5% to 10%.

4.7 Summary of GOLF with Locality Filtering

This section proposes the GOLF scheme with locality filtering to mitigate Sybil attacks in a BitTorrent system. The GOLF scheme uses cooperation between directly-connected peers to spread information about detected attackers. Each leecher learns potential attackers from neighbors with whom it exchanges file pieces. The input from neighbors is weighted by their past positive behavior. Locality filtering helps a seeder evade traffic exhaustion by Sybil nodes, and helps the tracker guide leechers to good neighbors in the swarm. To facilitate locality filtering, computation overhead requires O(1) for hash operations and the space overhead requires 8KB for 1,000 peers in the swarm.

The simulation results show the benefits of our scheme. Whereas Sybil nodes devastate the performance of the standard BitTorrent, the proposed scheme defeats malicious behaviors of Sybil nodes. By virtue of locality filtering and the GOLF scheme, the expected download completion time of non-malicious nodes is affected very little by the Sybil attack. The traffic triggered from a seeder reduces more than 10 times in the presence of Sybil nodes.

In the next chapter, we present an adaptive neighbor management scheme that uses a novel referral based incentive in order to prevent selfish users in P2P VoD streaming systems.

Chapter 5

Defending Against Selfish Peers for Cooperative P2P VoD Streaming

5.1 Introduction

On-demand streaming in the Internet has become a popular service in recent years. A limited number of companies supply the bulk of streaming content, and the demand for improved responsiveness, more sources, fewer delays or interruptions, more video formats, and lower costs has increased. The tremendous popularity of User-Generated Contents (UGC) and Internet TV stretches the normal client-server model. Instead of a dedicated collection of servers, the P2P approach makes it possible to sustain a great number of users at low cost. Huang et al. [83] showed content server overhead could be reduced by 95% as long as peers assist the server.

P2P Video-on-Demand (VoD) systems have several unique characteristics. One challenge is that peers have stringent time constraints to meet playback deadlines, once the video is started. This problem has been addressed in [84, 85], which showed how to effectively find the required data on time, in a cooperative manner. In such a system, a *junior* peer starts playback later than a *senior* peer. Junior and senior peers thus have asynchronous interests, with junior peers needing data from senior peers, but not the reverse. Reliance on a direct reciprocity scheme, such as Tit-For-Tat (TFT, used by BitTorrent [29]), is not effective in such circumstances [86].

Peers responding only to their own self-interest create challenges. Selfish peers (i.e., freeriders) who do not contribute upload bandwidth or storage space try to obtain streaming data for free. This increases the load on compliant peers who do contribute upload bandwidth and storage space, affecting streaming quality of service (QoS) and video availability. In essence, a *rational attack* [87] takes advantage of a protocol weakness for profit, causing degraded QoS. In this chapter, the goal of an incentive scheme is to motivate peers to contribute resources for their own benefit, and for the overall QoS of VoD.

According to the multicast overlay structure, P2P multicast can be divided into tree-based and mesh-

based approaches. Tree-based multicasting [88, 89, 90, 91, 57] establishes a virtual pipeline from the root (content owner) to leaf peers. Since such approaches push streaming data into the pipeline, they fit well the requirements of live video streaming. However, tree-based multicasting is not well-suited for on-demand streaming. Maintaining tree structures is also hard, and sensitive to rapid changes in membership. Instead, mesh-based approaches [86, 58, 84, 85] are robust in the presence of peer churn. To achieve competent QoS, they pull streaming data from neighbors. In such multicasting, the core problem for QoS is adaptively finding and selecting useful neighbors as well as collaborating with other peers under asymmetric interests. In addition, a hybrid approach [92] can inherit a good QoS and robustness, but require the further costs to support them.

Asymmetric peer interests complicate the task of incentivizing cooperation. A peer in mesh-based streaming has a partial view of the participants. Peers with uncooperative neighbors cannot always use their uplink capacity effectively. Peers need to find neighbors who reciprocate for their downloads, who are interested in the same content, and who are roughly at the same point of playback (seniority). Local view expansion, via the large view exploit [4] and Peer EXchange protocol (PEX) [39], can incur significant overhead. Finding and peering with compatible neighbors is therefore a fundamental problem in minimizing the problem of uncooperative peers for P2P VoD streaming.

In this chapter, we present *CoVoD*, a P2P VoD streaming method that works when a significant percentage of peers are uncooperative. CoVoD adopts a *referral* based neighbor management scheme. Each peer expands its local view through referral information, and rewards the source of the referral (by means of an unchoke, or upload, slot). This referral incentive enables peers joining later in the video playback to also benefit in asymmetric interest environments. CoVoD helps peers find potential neighbors adaptively, encourage cooperation, and ensure resources utilization. To evaluate the effectiveness, we applied CoVoD to BitTorrent-like streaming, and the results of experimental evaluation are exhibited:

- CoVoD maintains high QoS of compliant peers. It achieves 95% or better on-time delivery of video to cooperative peers when 20% of the participants are freeriders. The on-time delivery of video to freeriders, in contrast, never exceeds 30%, regardless of the fraction of freeriders.

- CoVoD achieves high utilization and fairness simultaneously, for any percentage of freeriders. Freeriders are excluded from neighbor relationships by the referral system, and have little impact on fairness.

- CoVoD expands the knowledge of other peers in the system, without reliance on the tracker, despite the actions of uncooperative peers. The referral system helps each peer to avoid freeriders and to find new neighbors.

- CoVoD maintains a high QoS for cooperative peers, even if a streaming server crashes, or a number of peers join the swarm simultaneously (i.e., a flash crowd occurs).

We summarize several desirable properties of CoVoD scheme:

- It is a decentralized approach to penalize uncooperative peers. Each peer makes an unchoking decision based on the history of neighbors' contributions. It does not require a central server to block freeriders. Without referral or piece cooperation, uncooperative peers cannot achieve a good QoS.

- It provides peer cooperation in asymmetric interest environments. Peers with different positions in video playback use referrals to obtain file pieces; peers receiving referrals benefit from improved neighbor selection. It does not discriminate against newcomers. A referral incentive provides peers with file pieces or potential neighbor, regardless of peer' seniority. The referral can be considered payment for pieces.

- It is a practical approach. To adopt CoVoD scheme, it requires referral creation and computation for referral incentive. The referral can be easily combined with the existing PEX protocol. Apart from TFT costs, additional referral overhead is trivial.

This chapter is organized as follows. Section 5.2 briefly compares related approaches. Section 5.3 summarizes neighbor management schemes in BitTorrent. Section 5.4 details referral based neighbor management scheme. Section 5.5 evaluates the results. Section 5.6 analyzes the overhead and discusses some issues. Section 5.7 summarizes this chapter.

5.2 Related Work

P2P multicast systems are classified into two categories based on overlay networks; tree-based and mesh-based approaches. Tree-based approaches [88, 90, 93] use the traditional IP multicast and application level multicast through push scheme. They can supply streaming data from the root (content owner) to leaf peers, as if in a pipeline. If an intermediate peer has a relatively small bandwidth, that peer will be a bottleneck and cause a degraded QoS to peers below it in the distribution tree. Typically, intermediate peers are more overloaded than leaf peers. To mitigate this problem, multi-tree approaches were proposed [89, 91]. Unfortunately, tree-based approaches are still expensive to maintain if the membership of the group changes frequently (i.e., has high churn) [94].

On the other hand, mesh-based approaches [84, 85, 94, 95, 86] have adopted the BitTorrent protocol for efficient file dissemination. They dynamically connect to other peers to obtain streaming data by pull method. In spite of the robustness against churn (i.e., peers dynamically leaving and joining the content network), streaming QoS can be unstable with these approaches. To improve the service quality, BiToS [86] (i.e., BitTorrent for streaming) uses a piece selection algorithm assigning priorities. Yang et al. [84] suggests load balancing for peers based on deadline-aware piece scheduling. iPass [85] uses

adaptive incentives with tracker assistance to improve server costs. Feng et al. [96] proposes a prefetching method between peers to reduce server bandwidth costs. Piatek et al. [42] recommends an incentive scheme by contracts among peers to improve the limitation of symmetric reciprocation. Additionally, AirVoD [92] has a tree-based structure as well as mesh-based functions. It requires management overhead of both streaming servers and super nodes.

All such efforts were designed for cooperative environments, in which peers comply with the protocol. However, peers are not always cooperative in P2P systems. For instance, some peers (i.e. freeriders) seek free service, taking advantage of cooperative peers and protocol weaknesses [28, 4]. Although this phenomenon is not much different for streaming applications, solutions developed for file-sharing cannot be applied directly. For instance, a micro-payment system [35, 37] requires a complicated infrastructure for transaction accounting and banking. A reputation approach [34] depends on indirect observations, and should maintain information about neighbors, across swarms, for extended periods of time. One-hop reputation penalizes a newcomer through long history, and peers may restrict to repeat more than one TFT behavior out of the swarm. If one-hop reputation is used in one swarm, it can help each peer expand its own view. However, it does not provide additional incentives to exchange data pieces. This is because streaming typically has a unidirectional flow between peers. Finally, piece modification schemes [13, 38] make use of encryption or coding of pieces to prevent freeriding. These methods address free-riding but do not address the special requirements of VoD streaming.

Orchard [57] creates dynamic multiple spanning trees to prevent freeriding. However, this scheme inherits the drawbacks of tree-based approaches. Give-to-Get (G2G) [58] introduces a ranking-based selection against freeriding in P2P VoD. In G2G, peers give data received from a peer to neighbors to get pieces from the peer. However, one-time freeriding is still possible at a middleman, which degrades service quality. In addition, in [57, 58] each peer makes a decision based on the history of its neighbors' contributions. This decision is limited by the local view of neighbors.

The method proposed in this chapter, entitled *CoVoD*, uses reciprocity for mesh based VoD streaming. Unlike G2G or other mesh-based schemes, CoVoD fosters cooperation among peers with asymmetric interests (different positions in video playback), by means of referrals. Referrals broaden the set of potential, cooperative neighbors. To the best of our knowledge, this chapter presents the first direct cooperative scheme between peers with asymmetric interests for VoD streaming, using adaptive neighbor management to achieve high QoS and fairness despite the presence of freeriders.

5.3 Neighbor Management Protocols

The BitTorrent systems have connection management algorithms to handle incoming and outgoing requests. Each peer establishes TCP connections with peers informed by the tracker. Due to dynamic peer membership, a peer typically has open slots that are filled by other peers to which the peer is connected. Such algorithm for incoming requests is first-come, first-served. Each peer continuously listens and

accepts a new incoming request. The connection is established if the maximum number of connection slots is not reached. For outgoing connections, the peer finds better candidates according to the TFT scheme.

For peer discovery, a general approach is based on the tracker (i.e., a centralized server). Once obtaining initial neighbors and connecting them, a peer periodically checks whether or not its neighbors are dead (no longer responding). If so, the peer removes them from the neighbor list. Each peer queries the tracker to harvest new neighbors in a swarm: (i) when a regular interval expires; (ii) when the neighbor list is insufficient to trade pieces; (iii) when the leecher completes the download.

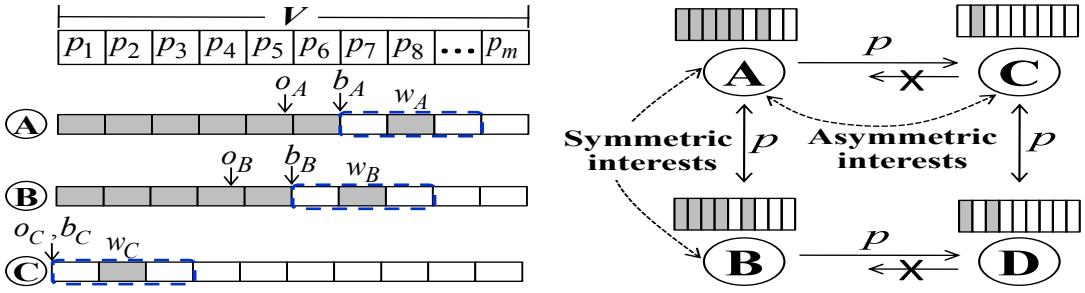
Additionally, BitTorrent systems have decentralized peer discovery schemes; that is, Distributed Hash Tables (DHTs) [97] and Peer Exchange protocol [39]. DHT based on Kademlia [98] can provide peer discovery service to peers. For peer lookup, each peer stores hash tables (i.e., distributed databases for tracking swarm participants) with the format of $(key, value)$ pairs, and retrieves the value associated with a given key. Peers can expand their swarm view through the DHT interface. However, peers' churns may make DHT maintenance overhead, and strategic attackers can compromise DHTs by fake identities (i.e., index poisoning attack [99] and Sybil attack [16]). The excessive lookup queries cause high communication cost, which can trigger a DDoS attack.

On the other hand, PEX reduces dependency from the tracker for peer discovery through a direct exchange between two communication parties. Most BitTorrent systems, such as Vuze (Azuerues), BitComet, and *u*Torrent, have adopted the PEX protocol. PEX allows each peer to directly exchange currently-active neighbor' neighbor list. To probe for better pairings, a peer using PEX shares information about active neighbors' IDs (i.e., peer's ID, IP address, and port number) with currently-connected neighbors, respectively. However, a peer cannot know a potential neighbor's state until a connection is established. After establishing the TCP connection and sharing a *bitfield* message (i.e., a control message representing the pieces that have been successfully obtained), the peer will determine whether the new neighbor connection is useful.

A good neighbor selection is connected to high piece diversity. Since peers use a LRF algorithm to determine which piece to request from the uploading peer, they maintain piece diversities with neighbors having symmetric interests in BitTorrent file-sharing. Streaming service has a stringent time constraint, which argues for different piece selection policies. BiToS [86] adapted a piece selection scheme based on playback windows (i.e., a high priority set and a remaining piece set). In recent work, a deadline aware approach [84] tried to improve playback progress through load-balancing.

5.4 CoVoD: Cooperative P2P VoD Streaming

This section presents the design of CoVoD, which uses referrals to improve neighbor management. The ultimate aim is to increase effective peer cooperation in P2P VoD streaming. Peers make use of referral information to obtain streaming data, by adaptively selecting useful neighbors. The notations of the



(a) Streaming structure & notations (b) Interests between peers

Figure 5.1: The video content being shared by a swarm includes m streaming pieces. The circle represents a peer in the swarm. A and B have symmetric interests, and are senior to peer C and D.

proposed scheme are summarized in Table 5.1.

Table 5.1: Summary of notation

V	video content being shared by a swarm
m	# of streaming pieces ($V = \{p_1, p_2, \ldots, p_m\}$)
N_A	neighbor set of peer A
I_A	identification of peer A
w_A	the size of buffer window of peer A
b_A	pre-buffering position of peer A
o_A	playback position of peer A
d	piece difference in an interested range
$R^A_{N_B}$	referral sent to peer A from peer B using N_B
$n_{A \leftarrow x}$	# of referrals from peer x to peer A
U^r_x	# of referral unchoking to peer x
S^r_x	referral score of peer x
S^p_x	piece contribution score of peer x
F_A	fairness factor achived by peer A
C_A	overall communication cost of peer A

5.4.1 Design Overview

Mutual interests are normally the basis of voluntary cooperation among peers. This is often accomplished by bartering, or trading, of file pieces. However, it is harder to identify mutual interests in VoD streaming. Peers with asynchronous start and stop behavior are at different stages of playback, which causes a directional data flow between peers.

The normal purpose of neighboring is to cluster peers with symmetric interests for a long-term bartering relationship. Typically, BitTorrent-like streaming employs the tracker that is both a bootstrap server, and a coordinator informing peers of potential neighbors. It can be a central server to select neighbors with mutual interests. However, additional tracking of participants' playback progress not only imposes communication overhead, but represents a significant bottleneck for large systems. Unlike such a centralized approach, CoVoD seeks a decentralized approach.

According to playback progress, peers are divided into senior (i.e., a peer playing the video for a longer time) and junior (i.e., a peer playing the video for a shorter time) categories. A senior peer has little interest in junior peers, preventing file-piece exchanges. Neighboring with juniors not only incurs a waste of connection slots, but degrades service quality. To achieve peer cooperation under asymmetric interests, the primary change of CoVoD incentive is for unchoking and seeding. The referral mechanism modifies which peers are selected for uploading, and how referral scores are computed by each peer.

CoVoD scheme provides peers with piece and referral cooperation to share streaming pieces. The piece cooperation indicates piece exchange. Each peer determines an interest on the basis of whether or not a neighbor holds a required piece. If mutual interests are established, peers share their "pieces of interest" with each other. Otherwise, a referral arranges a condition of direct cooperation between peers with asymmetric interests. They do not passively wait optimistic unchoking. They actively utilize resources to make a referral and offer each referral to neighbors. So, selfish or uncooperative peers have little gains without effort. As trading referral information provides an opportunity of being unchoked, referrals increase piece exchanges as well as enhancing neighbor selection.

5.4.2 Referral based Neighbor Management

Referral based neighbor management is a *decentralized* approach to dynamically identify peers with mutual interests. A referral contains the piece difference with a direct neighbor's neighbors (i.e., "one-hop" neighbors [34]), to determine their compatibility of interest. To discover bartering candidates, each peer expands its local view to one-hop neighbors in the swarm, by means of referral messages. Referrals are not the same as reputation or history; referrals are reports based on direct knowledge of the current state of neighboring peers.

Referrals are used during a peer's overall states. When establishing TCP connections, a newcomer exchanges bitfield messages[1] with its neighbors. The newcomer can then create referral information about its local neighbors, which it exchanges with those neighbors. Through such referrals the neighbors have chances both to expand a view of the current swarm and to discover candidates with symmetric interests. A peer having a relatively junior state will contribute referrals to its senior neighbors.

[1] In a bitfield message, each piece is encoded as one bit. The size of bitfield is determined by the total number of pieces (m).

Referrals

As shown in Figure 5.1(b), Peer C interested in file pieces held by neighbor A, but which does not have any file pieces of interest to A, is said to have asymmetric interests. In the normal BitTorrent protocol, C is unable to download many file pieces from A, because C has nothing to offer A in return (the solution in BitTorrent is optimistic unchoking, which has multiple problems). To counteract this, CoVoD proposes that C offer something else to A, namely, a referral. Referrals are sent by C every unchoke period to all of its neighbors with asymmetric interests.

A referral from a peer C to its neighbor A contains the IDs of all of C's neighbors. For each such neighbor x of C, the referral also includes the piece differences between x and A, the playback position (i.e., currently playing piece position) of x, and the pre-buffering position (i.e., the first piece position in the buffer window) of x.

As video playback proceeds, the number of neighbors with asymmetric interests is likely to decrease, but referrals will continuously be sent as long as there is any neighbor with asymmetric interests. The information sent by C to its neighbors with asymmetric interests is therefore kept up-to-date.

Selection of Neighbors

Each peer has 35 connection slots to connect to participants for streaming content. 30 slots are filled by random neighbors, and the remaining slots are reserved for referrals. Initially, a newcomer requests TCP connections to candidate neighbors provided by the tracker. If a candidate neighbor permits, the newcomer establishes the connection with that peer, and enters into a neighbor relation. The remaining open slots will be filled through referrals, and connection requests from other peers. As referrals become available, if there are open connection slots, the top scoring referral (explained below) will be used to fill that slot. In addition, each peer asks the tracker for additional candidate neighbors if the number of connections drops below 20.

In iPass [85], the tracker designates candidate neighbors with similar joining times to the requesting peer. Such biased selection tends to cluster peers with overlapping interest ranges. This unfortunately means neighbor diversity is reduced, and competition for unchoke slots is increased. Instead, CoVoD uses the normal random neighbor selection of the tracker, combined with decentralized referrals. Referrals help select neighbors with pieces that are useful. Through piece difference and pre-buffering position, each peer can estimate whether a candidate neighbor meets the conditions for reciprocal piece barter. Although the PEX protocol [39] increases the number of neighbors, it cannot provide neighbors' state until the connection is established.

Replacement of Neighbors

A referral-based neighbor scheme adaptively replaces neighbors. A peer dynamically decides to replace a neighbor if any of the following conditions are met: (i) If a connection is broken by the other end,

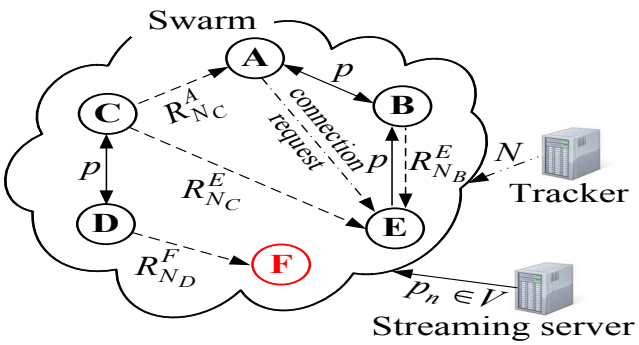


Figure 5.2: Example of CoVoD scheme

then that neighbor is replaced. A broken connection may be an active termination, or may be the lack of a response to a heartbeat message. (ii) If a neighbor fails to cooperate for 5 or more consecutive unchoke periods, that neighbor is replaced. Noncooperation is defined as providing neither referrals nor streaming pieces. (iii) If a neighbor finishes downloading the remaining streaming pieces that are needed after the current playback position, connections with other seeders are broken, and those neighbors are replaced.

After replacement, peer A requests a TCP connection to candidate neighbors, sorted by the following steps. Each peer sorts potential neighbors through referrals every unchoke period. To select a new neighbor from referrals, the compatibility of interests is considered. Once peer A obtains referrals, candidate neighbors are ranked on piece difference and pre-buffering position as follows: (i) If a candidate is a seeder, it will be ranked the highest. (ii) Peers with mutual interests are considered next. Peer A computes (from referrals) the difference in the pre-buffering position between A and candidate x, ($|b_x - b_A|$). Peers with overlapping interest ranges have high priority, where $0 \leq |b_x - b_A| \leq w_A$. Peer A sorts these candidates based on their piece differences in A's interest range (i.e., A's pre-buffering window, denoted $[b_A^i, b_A^i + w_A]$). (iii) Candidates with $|b_x - b_A| > w_A$ have the next priority. Peer A regards such candidates as having asymmetric interests. The larger the difference in playback position, the lower will be the priority.

Exchange and Use of Referrals

Referral neighbor management increases cooperation among peers, as shown in Figure 5.2. Peer A and B with similar video playback progress have symmetric interests, and exchange required pieces each other. On the other hand, a junior peer C at an earlier playback position has an interest in video data held by senior peer A, but A does not have an interest in data held by C. Under this asymmetry of interests, peer C provides referrals to A to obtain an unchoke slot from peer A.

A *piece difference* d_{AE} between peers A and E is the set of pieces in A's interested range, not yet obtained by A, but held by E. By means of the normal BitTorrent protocol, peer C knows the pieces held by each of its neighbors, and can compute the piece difference between A and each of the other neighbors in the set N_C. After doing so, peer C sends peer A a referral $R^A_{N_C}$ =

$$< I_D \mid b_D \mid o_D \mid d_{AD} > \mid < I_E \mid b_E \mid o_E \mid d_{AE} > \mid \ldots$$

Upon receiving the referral $R^A_{N_C}$, peer A can select a new, potentially useful peer with which to barter, and establish a TCP connection to it. In addition, peer A will unchoke C, allowing it to download a file piece, as a reward.

Participants cooperate with each other by means of referrals or streaming pieces. Peer D discovering that neighbor F does not exchange file pieces or provide referrals, will prune that neighbor from its neighbor set. Unlike optimistic unchoking, exchange of keep-alive messages is not sufficient to obtain streaming pieces.

5.4.3 CoVoD Incentive Schemes

To give a chance to pull file pieces, CoVoD has two unchoking options; TFT (file piece reciprocity) and referral slots. Each peer has a total of five unchoke slots. Typically, the unchoke slots are comprised of three for TFT, and two for referrals. The streaming server and seeders operate five unchoke slots, all for referrals. TFT slots are assigned by the bartering history. TFT incentive not only provides fairness to peers for piece bartering but also maximizes uplink utilization for piece cooperation among peers with mutual interests. Referral incentives foster peer cooperation in a decentralized manner without tracker assistance. The referral slots are determined by the sorted referral scores of each peer, as explained below.

$$S^r_x = \tau_A \times (n_{A \leftarrow x} - UC^r_x), \tag{5.1}$$

where τ_A is a *piece deadline factor*, $n_{x \leftarrow A}$ is the cumulative number of referrals received from A, and U^r_A is the cumulative number of referral unchoke slots given to peer A. The values of $n_{x \leftarrow A}$ and U^r_x monotonically increase, and all values are updated each unchoke period.

The piece deadline factor τ_A is determined by the current pieces held by A. If A is in need of pieces for the next unchoke slot (i.e. 5 seconds), τ_A has the value 1. Otherwise, τ_A has the value 0.7, since A has enough pieces buffered for the next unchoke slot. This adjustment helps peer A that has an imminent deadline obtain required pieces on time. This mechanism is based on [84], which uses deadline-aware scheduling for the next unchoking selection.

To use referral unchoking, peer x sorts the resulting referral scores $S^r_{A \in N_x}$ for neighbors with asymmetric interests. Among neighbors whose referral scores are the highest, referral unchoking follows a

round-robin policy, as long as neighbors continue to send referral messages.

Once it has obtained pieces from the streaming server or neighbors by means of referral unchoking, the peer A can trade the pieces it has downloaded, using TFT unchoking. For each unchoke period, peer A updates the piece contribution score S_x^p for each of its neighbors $x \in N_A$. The piece contribution score S_x^p equals the count of pieces downloaded in response to the TFT policy (i.e., pieces downloaded in return for referrals are not counted). Peer A sorts the piece contribution scores S_x^p for each neighbor $x \in N_A$, and selects the top scoring neighbors for TFT unchoking. Note that the piece contribution score reflects the volume of uploaded pieces:

$$S_x^p = \delta_{A \leftarrow x} * \overline{d}_{A \leftarrow x}, \tag{5.2}$$

where $\delta_{A \leftarrow x}$ and $\overline{d}_{A \leftarrow x}$ define the download duration and the average downlink rate from neighbor x, respectively. Each peer determines peers as unchoke recipients based on those with maximum sorted contribution scores.

The fairness achieved by peer A represents the ratio of total uploaded and downloaded volumes before it finished obtaining the video and becomes a seeder, denoted by F_A, computed as

$$F_A = \sum_x S_{x \leftarrow A}^p / \sum_x S_{A \leftarrow x}^p, \tag{5.3}$$

for all peers x in the swarm. In a perfectly fair system, F_A = 1 for all peers, which means they have the same amount of volumes for downloading and uploading. It is difficult to achieve perfect fairness in VoD systems because of the differences in playback startup times of peers. To improve fairness, CoVoD combines TFT and referral unchoking. TFT unchoking leads to fair exchange between peers with mutual interests. A peer receives streaming pieces from seniors, and offers pieces to juniors, in return for referrals.

5.5 Performance Evaluation

This section evaluates the proposed method. Since it is difficult to arrange for a range of user behaviors in an actual deployment, performance results are shown by simulation. We do not claim optimality or guarantee convergence under all circumstances for the referral unchoking mechanism. The experimental results illustrate performance under realistic conditions. The experimental setup is first described, followed by the results.

5.5.1 Experimental Setup

Simulation settings

We developed an event-driven simulator of the BitTorrent protocol, modified for VoD streaming as described above. The events include joins, leaves, (TFT and referral) unchoking, playback, referrals, and piece exchanges. According to [78], the volume of the control messages in BitTorrent is negligible compared to the size of file pieces that are exchanged. Thus, the simulator does not account for the delays due to control messages. To reduce simulation complexity, the network was assumed to have no bottlenecks or transmission errors [58]. The same simulation parameters are used for all methods, including the inter-arrival rate, file size, bandwidth capacity, number of streaming servers, and the startup delay.

Table 5.2: Simulation Parameters

Content encoding rate	500Kbps
Piece size	64KB (512 Kbit)
Content size	38 MB (10 min, 600 piece)
Total # of peers	3,000
Average inter-arrival time	4 sec & flash-crowd
Startup delay	10 sec
Neighbor (peer) set	35
Max # upload slots	5
Link capacity (up / down)	600 / 2000 Kbps
# of streaming server	1
Server capacity (up / down)	5000 / 5000 Kbps

Table 5.2 shows the simulation parameters. Many parameter settings are taken from [86, 58, 84]. Each experiment streamed a single 500Kbps video [84] for 10 minutes [86]. The content was divided into 64KB pieces, and had 600 pieces. The total number of peers was limited to 3,000. Since most BitTorrent-like streaming has fewer than 300 peers in a swarm [100] at any one time, this swarm size was emulated. Each simulation started with one streaming server, and one tracker [58, 85]. The server and tracker served all participants in the swarm throughout the simulation. Peers joined the content swarm based on a Poisson arrival rate, with an inter-arrival time (λ) set to 4 seconds (as in [85]).

We investigated the impact of freeriders in the experiments. Freeriders are assumed to use the large-view exploit [4]. They do not contribute any piece to the neighbors. The result of freeriders sending, or not sending, referrals was investigated, and little difference was found; accordingly, only results for freeriders not sending referrals are shown.

A newcomer started playback of the video after a startup delay of 10 seconds [84]. Once a peer fin-

ished downloading the content, it became a seeder (i.e. a streaming server). A peer for which playback completed was assumed to leave the swarm. Based on an ADSL standard model, the upload bandwidth of all peers was set to 600Kbps. The bandwidth capacity of the streaming server was 5Mbps. Every unchoke period was 5 seconds in length, which was the same for unchoke cycles. The neighbor set size of each peers was limited to 35. The unchoking slots were set to 5, where 3 slots were allocated for TFT unchoking, and 2 slots were allocated for referral unchoking[2]. Each simulation was repeated 10 times to compute 95% confidence intervals.

Performance metrics

The proposed method was compared to two different models, BiToS and G2G [3]. BiToS is a "baseline" method, based on BitTorrent, for video distribution. G2G was proposed as a method of combatting freeriders in VoD streaming.

Acceptable service quality is the main goal of a VoD system. The *continuity index (CI)*, introduced in [41], is a suitable metric for the quality of video service in VoD systems. The CI is defined by the ratio of the number of pieces arriving before the playback deadline to the total number of pieces in the content.

$$CI = \frac{\#\ of\ received\ pieces\ before\ deadline}{total\ \#\ of\ pieces\ in\ the\ content} \tag{5.4}$$

A peer having a CI of 1 means that it has experienced seamless (uninterrupted) video service during its playback. If the CI is less than 1, the peer has experienced some video jitter or quality loss. To evaluate the overall QoS of a system, the average CI for all participants is used.

A perception of *fairness* encourages peers to work cooperatively, where fairness means a peer enjoys as many benefits as it offers. As mentioned in the section 5.4.3, the normalized fairness of each peer for VoD distribution corresponds to the ratio of sent to received file pieces. If the fairness factor of a peer is more than 1, it has uploaded more than it has downloaded, and if it is less than 1, the reverse is true. Selfish peers seek to achieve a fairness as close to zero as possible, while cooperative peers expect a fairness close to 1.

A BitTorrent-like system has an optimal *utilization* if it achieves maximum uplink utilization for each peer, and has perfect fairness if every participant downloads as much as it uploads. If both (i.e., optimal utilization as well as perfect fairness) are achieved, the system is referred to as being optimal [31].

In addition, the reduction of load on a VoD server is one of the metrics. In a traditional client-server model, the server unicasts every client separately. So, total cost is equal to the content size multiplies the number of clients. However, the server reduces its resource cost as much as sharing content size among peers.

[2]This is based on [101], which has shown the trade-off of performance and fairness with the number of allocated unchoke slots.

[3]The original G2G simulator [102] was used and BiToS simulator was developed. Their simulation parameters were set to the same as those of CoVoD.

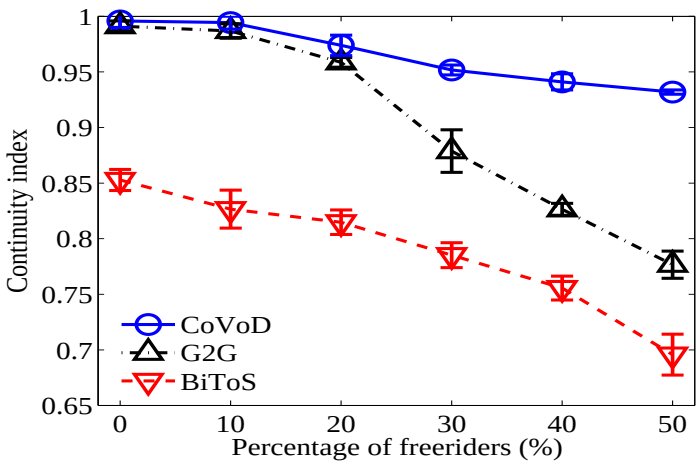


Figure 5.3: Comparison to QoS, which shows the average continuity index for peers on three methods.

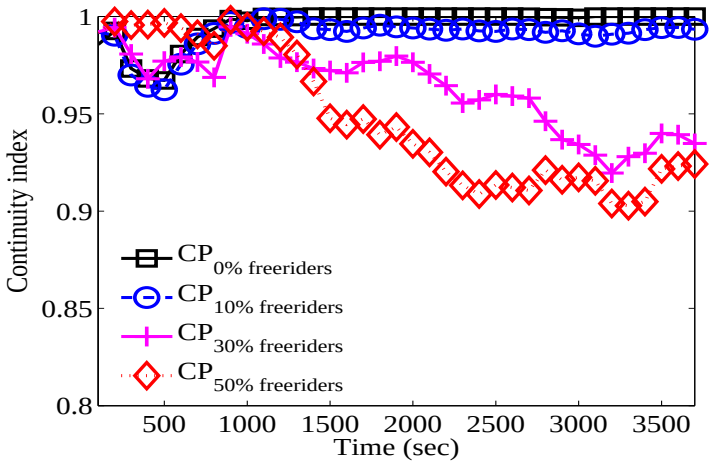
5.5.2 Experimental Results

Service quality

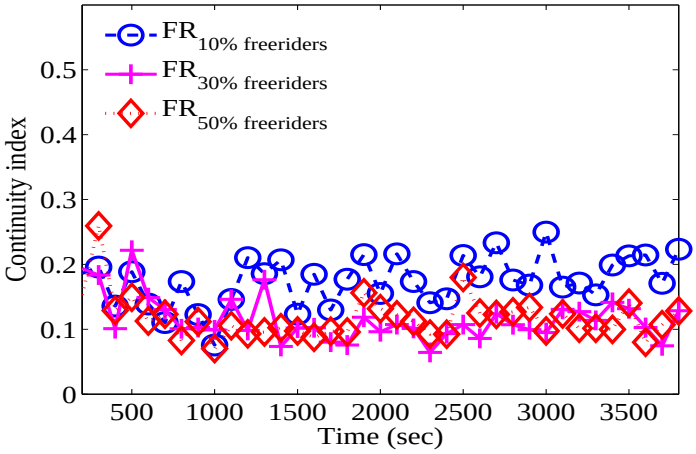
The first experiment investigated the average CI as a function of the percentage of freeriders in the system. This was measured and averaged over the duration of the simulation, with 95% confidence intervals indicated.

Figure 6.3 shows the result for the proposed method, BiToS, and G2G. CI degrades for all as the percentage of freeriders increases. BiToS shows the worst results, and CoVoD the best, particularly as the percentage of freeriders exceeds 30%. The overall trends show that peers experienced the worse service quality as the quantity of freeriders. BiToS simply modifies its piece picking scheme, based on a streaming window, and does not try to avoid freeriders. With G2G, a peer cannot directly tell whether an intermediary is a freerider. Freeriders acting as intermediaries will make beneficiaries (downstream from them) miss streaming piece deadlines. CoVoD is less impacted by uncooperative peers. This is because the file pieces are only uploaded in response to other pieces, or referrals.

Figures 6.4 show the change in the CI of CoVoD over time as a function of the percentage of freeriders, measured separately for compliant peers, and for freeriders. The CI of compliant peers was above .95 when freeriding percentages were less than or equal to 20%. In contrast, the CI of freeriders never exceeded .3, for any percentage of freeriders. Referrals are effective at discriminating between compliant peers and freeriders. In Figure 6.4(a), a dip at the start is caused by a cold start. The streaming server only supported five peers based on referrals every unchoke period, which triggered a deadline miss of some peers at the beginning.



(a) QoS of compliant peers in CoVoD.



(b) QoS of freeriders in CoVoD.

Figure 5.4: Transition of service quality for two types of peers under CoVoD scheme. In Fig. 6.4(a), CP indicates compliant peers, and FR means freeriders in Fig. 5.4(b).

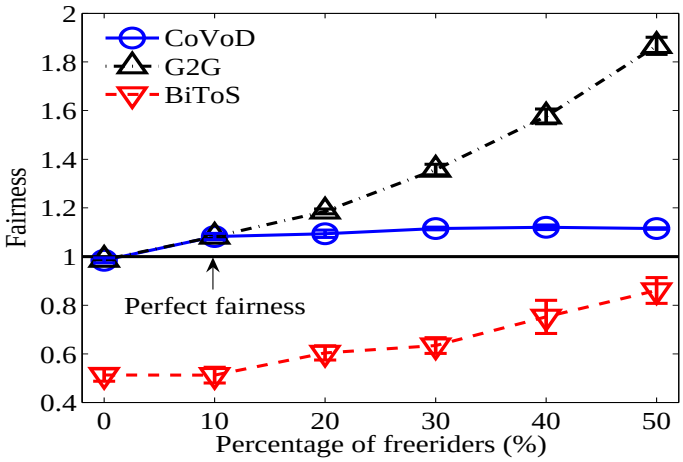


Figure 5.5: Comparison to the average fairness for peers, which shows the fraction of a quantity of sent and received pieces by peers on three methods.

Fairness & Uplink utilization

The second experiment investigated the average fairness (with confidence intervals indicated) for peers. The result is shown in Figure 5.5. In G2G, a contributor optimistically believes an intermediary delivers all received pieces to its neighbors. Since a peer occasionally selects freeriders as intermediaries, such freeriders obtain streaming data for free, in spite of being excluded at the next unchoke period. In BiToS, by contrast, peers downloaded more than they uploaded, on average. This is somewhat surprising, since BiToS uses TFT incentive mechanism. The problem is that peers have difficulty finding suitable neighbors due to asymmetric interests. Peers could not therefore fill regular unchoke slots, although they receive streaming pieces by optimistic unchoking.

Both BiToS and G2G show marked lack of fairness. CoVoD, however, shows notably better fairness (closer to 1), regardless of the percentage of freeriders. Freeriders are excluded from neighbor relationships by the referral system, and thus have little impact on fairness.

To further compare the above three methods, the average uplink utilization was also investigated. Figure 6.5 shows uplink utilization as a function of the percentage of freeriders. CoVoD and G2G uniformly achieve a high utilization, despite asymmetry of interests in a VoD system, while BiToS utilization is much lower. From these two figures, only CoVoD achieves high utilization and very good fairness simultaneously, for all percentages of freeriders.

In this experiment, the average uplink utilization for junior and senior peers was examined. As mentioned in 5.4.1, senior peers join a VoD session early and have many opportunities for sharing, while junior peers join later and have fewer opportunities. This is partially offset by the ability of junior peers to download file pieces in response to their willingness to provide referrals. The junior peers

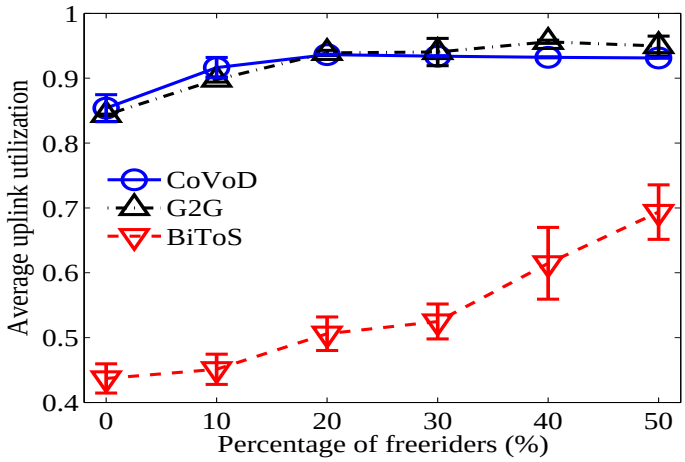


Figure 5.6: Comparison to the average uplink utilization for peers, which shows the fraction of used and maximum uplink rate by peers on three methods.

overall show somewhat lower uplink utilization, as shown in Figure 5.7.

Swarm coverage

CoVoD encourages peer cooperation by referral incentives, despite the presence of uncooperative peers. To investigate this, the *swarm coverage* for compliant peers was measured. The swarm coverage is the average, over all peers, of the ratio of known peers to currently-active swarm participants in Figure 5.8. With no freeriders, on average each compliant peer knew close to 80% participants, while each peer knew more than 98% of the other participants when freeriding increased to 30%. This is because the presence of freeriders encourages pruning of freerider neighbors, and searching for other (non-freerider) neighbors. Hence, CoVoD has an adaptive neighbor management against freeriders. Note that the result of non-freeriders is compatible with the coverage of M^2 friends in [54].

Server failure

Another experiment investigated the effect of failures of streaming servers. This scenario assumes there are sufficient participants in the swarm to maintain the availability of the video before the failure of a streaming server by an unexpected cause. We set the crash of the streaming server after 1,000 seconds of starting a streaming service. After that time, there was no piece source except the peers themselves, which emulated a pure P2P streaming system.

Figure 5.9 shows the average CIs of compliant peers, while varying freerider percentage from 0% to 50%. Despite the loss of the server, the peers in the swarm effectively obtained streaming pieces, and maintained a high CI. Referral unchoking is valuable in this case, as well.

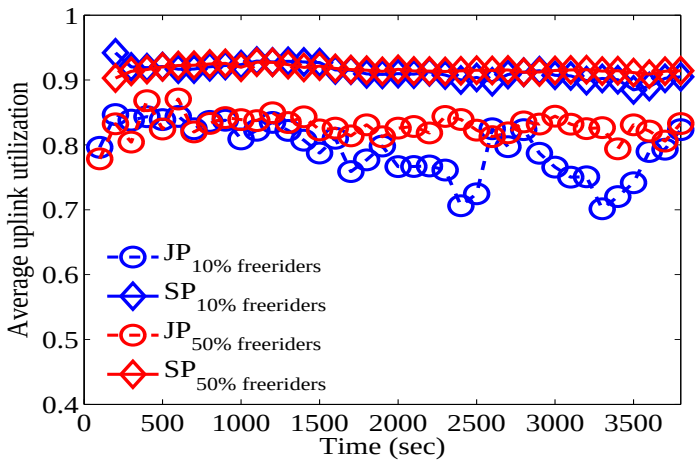


Figure 5.7: Transition of the average uplink utilization between junior peers (JP) and senior peers (SP) with CoVoD.

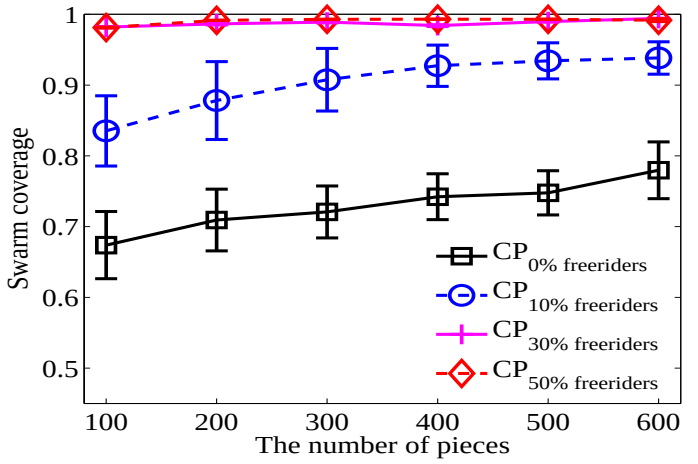


Figure 5.8: Swarm coverage as the number of pieces received, which shows the average of the ratio of known peers to currently-active swarm participants by each peer.

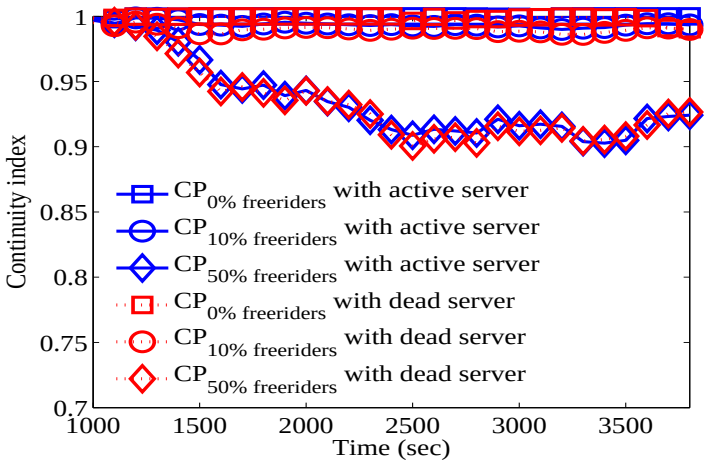


Figure 5.9: QoS variation under a server failure model, which shows the average CIs of each compliant peer as varying from 0% to 50% freeriders.

Flash crowd

P2P VoD systems have a flash crowd property, which makes it difficult to support stable QoS to peers. To explore this, the service quality by a flash crowd model was measured in the last experiment. In this scenario, 200 peers joined the swarm during 10 seconds after 1,000 seconds. The percentage of freeriders was also varied. Typically, resources under the flash crowd model should be distributed for newcomers to keep high CIs.

The results over time as a function of the percentage of freeriders are shown in Figure 5.10. The CIs of compliant peers maintained around 98% CIs, when the freeriding percentage was less than or equal to 10%, excepting some ranges after 25 minutes. This is because bulk peers may compete for file pieces against other peers with close playback progresses. Hence, some peers missed streaming deadlines like a cold start problem.

5.6 Discussion

In this section, we analyze the cost for deploying CoVoD scheme in on-demanding streaming. After that, we discuss the limitations of CoVoD scheme and possible solutions to those limitations.

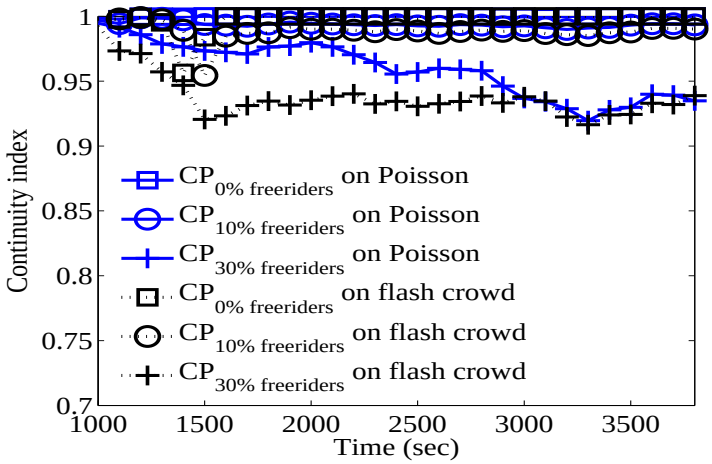


Figure 5.10: CIs under a flash crowd arrival model, which shows the average CIs, according to the portion of freeriders. The inter-arrival rate follows 4 seconds for the Poisson model, and the flash crowd model reflects peers joining the swarm in bulk during 10 seconds after 1,000 seconds of starting a streaming service.

5.6.1 CoVoD Cost

To deploy the proposed scheme in a BitTorrent-like streaming, each peer has additional communication costs by referrals. The overall communication cost of peer A is derived as

$$C_A = \alpha \sum_{x}^{N_A} S^p_{x \leftarrow A} + \rho \sum_{x}^{N_A} n_{x \leftarrow A}, \tag{5.5}$$

where α and ρ are weighting factors for piece upload costs and referral costs for peer A. The first term in this expression represents the total volume of uploads from A to the neighbors x in its neighbor set N_A. This is the cost for TFT unchoking. The second term represents the total number of referrals to neighbors x in N_A. The referral cost is an additional factor compared to a basic BitTorrent file-sharing application.

Since the size of piece is 64KB, α has the same value. The size of a referral message depends on the number of neighbors. The size for an ID has 26 Bytes (20 bytes for ID (SHA-1), 4bytes for IP address, and 2bytes for port number). The size of playback position, pre-buffering position, and piece difference can be set by the total piece number. In the experiment we set 2 bytes for each, so ρ is set to 1120 bytes (=32 bytes * the maximum number of neighbors).

Figure 5.11 shows the average cost of each compliant peer until the completion of watching the content, for varying the percentage of freeriders. The referral cost held under 2MB regardless of freeriders, and the piece cost slightly increased as a function of percent of freeriders. The communication overhead

by referrals occupies around 3% out of the total costs.

5.6.2 Limitations & Possible Solutions

Generally, freeriding is possible in the case of no incentive for donation and vulnerability to bypass a defending scheme. In BitTorrent protocol, optimistic unchoking is a well-known weakness. The passive attack strategy, such as large view exploit (i.e., maximizing the number of neighbors to obtain optimistic unchoke slots), can be easily attempted for free service. In contrast, CoVoD adopts referral unchoking slots instead of an optimistic unchoking slot. Hence, the passive attack strategy will not provide much benefit to freeriders.

Active attack strategies may threaten the CoVoD scheme. Similar to piece pollution in PPLive (i.e., a BitTorrent-like streaming) application [11], active polluters may degrade the QoS, squander resources, and compel to miss the deadline of playback progress. A possible solution can use fine-grained integrity checking of blocks, in addition to pieces.

Another threat is liars [15, 9]. There are two liar strategies to compromise information; the one is false referrals and the other is false reporting for pieces. To defend the first attack, a possible solution is to use an additional trust reputation [32]. The purpose of the latter is to lead falsely symmetric interests for one-time TFT unchoking. This piece under-reporting [9] may consume connection slots and gain TFT unchoking. Neighbor pruning scheme in CoVoD can cut them off in a neighbor set, although one-time cheating is admitted. Additionally, a possible defense uses a trusted hardware against lying [8]. However, such approach can impose peers on hardware costs, and make a barrier to use the systems.

The Sybil attack [16] is a fundamental problem in distributed systems. BitTorrent-like streaming is also vulnerable to Sybil attackers (white-washers) [16]. Since BitTorrent methods do not require verifiable identities, adversaries can manufacture fake identities to interfere with TFT and referrals, and collude to obtain pieces without contributing anything in return. There are many proposed solutions to the Sybil attack [18, 53, 25].

5.7 Summary of CoVoD

This chapter proposes a method of distributing, or multicasting, Video-on-demand (VoD) by means of peer-to-peer (mesh-based) distribution. The method achieves high uplink utilization and close to optimal fairness despite the asymmetry of interests among peers, and the presence of freeriders. The key is adaptive neighbor management by means of referrals. Referrals expand the local view of peers, and assist in finding cooperative neighbors with similar interests. In return, the provider of a referral is unchoked, thereby receiving additional pieces of the desired video file. In the next chapter, we present a defense mechanism against cheating attacks in mesh based streaming systems.

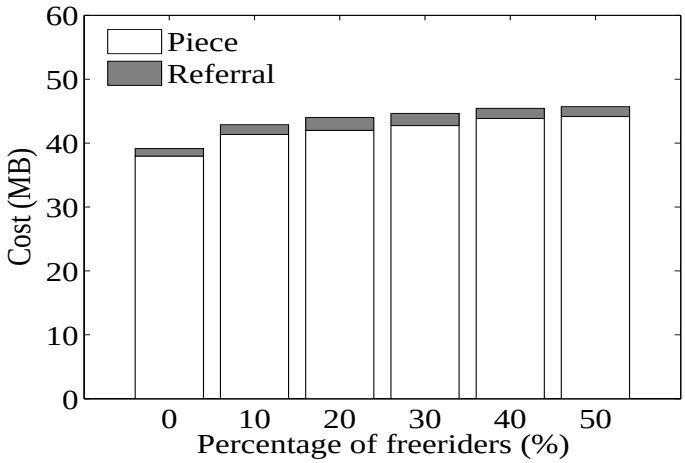


Figure 5.11: The average cost of each peer until the completion of watching the content.

Chapter 6

Defending against Cheating Attacks in Mesh based Streaming

6.1 Introduction

Peer-to-Peer (P2P) systems are based on autonomous peers and their cooperation. Unlike a traditional client-server system, each peer has the role of both a supplier and a consumer. Such systems have advantages of reliability, simplicity, scalability, and rapid adaptation to changing conditions. Peers divide work loads, and achieve their goals through self-active cooperation with each other. Typically, these properties make each peer cooperative and dependable to other participants in a rapidly-changing overlay network. For example, in mesh-based streaming systems based on BitTorrent, peers share content availability by gossiping, and redistribute data that is cached locally [41, 86, 103, 85, 84, 100]. Such systems have accommodated millions of users and offer thousands of media streams to watch [104, 105].

Each peer in mesh based streaming must cooperate with a limited set of other participants in order to accomplish mutual benefits. Such cooperation, based on searching, contracting for, or exchanging information, offers participants chances to supplement their resources. Unfortunately, a malicious attacker (i.e., a *liar*) can generate arbitrary false or fake messages, and compromise neighbors' information [70, 6, 7, 8, 9, 11, 71, 74]. These messages can mislead neighbors, causing them to waste their resources. In the end, this type of *cheating* deprives participants of the benefits of cooperation.

In P2P overlay streaming applications, there are two main trends: mesh based, and tree based approaches. In particular, the mesh based approach is promising for Internet based streaming services (IPTV) utilizing participants' resources. Recent studies have also indicated that mesh based streaming performs better than multi-tree based streaming [94]. The proliferation of mesh based streaming systems raises a question about how to defend against cheating attacks. A liar can easily join a content swarm, exploit security holes by gossiping, and disturb sharing mechanisms.

Lying to others can foster suspicion and impede cooperation. A strategic attacker sending misleading

information can easily deceive compliant peers and interfere with normal activities. For example, after obtaining useful streaming data from other peers, a liar can simply deny receipt of the data [6, 7]. As another example, a liar can retard the completion of downloading for compliant peers by sending a fake report to the BitTorrent swarm [8, 9]. Hence, defending against cheating attacks is key to making BitTorrent-like systems reliable.

Consensus protocols have been proposed to protect compliant participants from attackers or liars; seminal early work is Lamport's introduction of the Byzantine agreement problem in distributed systems [60]. Such approaches use state machine replication based on a reliable commit protocol with causality, or quorum-based approaches using majority voting among replicas. Such approaches, while provably strong, typically have high costs of replication and communication, and may be slow. They are thus unsuitable for the extreme scale and dynamic nature of most P2P systems.

Recent work has addressed Byzantine attacks (cheating) through deploying trusted hardware for attestation against equivocal behaviors [69, 8]. Although a trusted component improves communication overhead and scalability, it raises several concerns. The additional hardware requirement is a barrier, it raises a privacy problem, and malfunction of a trusted component by attacking brings about a new security hole.

Another approach uses accumulating evidence (*auditing or accountability*) of transactions as a practical solution [70, 71] in P2P systems. Auditing with a secure log can provide a peer with a means of detecting liars. However, these methods can still be expensive and somewhat slow. Since such an approach in general records all interactions and validates their causality, peers deal with additional responsibility.

In this chapter, we propose a defense, called *AntiLiar*, against cheating attacks in mesh-based streaming. AntiLiar is a practical method to reduce the impact of liars on service quality and peer cooperation. AntiLiar uses a secure *progress log* that consists of commitments and one-time signatures for authentication and message integrity. Each peer maintains its own progress log, and advertises the most current state of obtained pieces. Peers trace neighbors' state, and validate their commitments. They detect false reports through verifiable evidence sent from one-hop neighbors, and consistent neighbor management, and detect content pollution through block filtering. To detect willful omissions in an asynchronous way, AntiLiar makes use of non-repudiation of the verifiable evidence.

AntiLiar has been implemented, and shown to successfully provides stable quality of service in an efficient way. Experimental results show that compliant peers with AntiLiar experience at least 95% seamless video service under realistic conditions, even when 10% of the participants are liars who also exploit whitewashing and collusion. Comparison with alternatives shows AntiLiar significantly reduces overhead, and provides scalability and robustness.

AntiLiar has several desirable properties:

- *Defending cheating attacks:* It effectively defeats the impact of liars with informational integrity and authentication through commitment trees, and verifiable evidence from one-hop neighbors.

With AntiLiar each peer can maintain a consistent view for expanded neighbors, and blacklist malicious liars.

- *Cost-effective approach:* AntiLiar uses a cryptographic hash function for verification against cheating attacks. Each record in a progress log is committed in a secure way. AntiLiar associates a one-time signature and peer ID with hash chaining, which enables efficient authentication.

- *Decentralized approach:* AntiLiar enables each peer to independently detect cheating attacks. Each peer verifies its neighbors' state by comparing messages from one-hop and directly-connected neighbors, without relying on the tracker.

This chapter is organized as follows. Section 6.2 summarize gossip protocols. Section 6.3 describes our assumptions, and relevant attacks by liars. Section 6.4 details referral based neighbor management scheme. Section 6.5 evaluates the results. Section 6.6 briefly compares and describes existing methods of cheating prevention. Section 6.7 summary the proposed scheme.

6.2 Gossip Protocols

In this section, we briefly summarize the purposes of gossip protocols, and introduce the vulnerabilities of gossiping in mesh based streaming systems.

6.2.1 Definition & Functions

A gossip protocol is a communication protocol inspired by the form of gossip seen in social networks [7, 106]. P2P systems often use a gossip protocol due to its scalability, efficiency, simplicity, and applicability to varied conditions. Mesh-based streaming systems have scalability and no fixed underlying structure. In such systems, peers use a gossip protocol to find desirable neighbors and desired data. Gossiping tolerates message losses or node failures by randomly selecting peers. Unfortunately, it can be easily compromised by malicious participants (i.e. *liars*) [107, 7]. This is because gossiping has no verification mechanism for data that is exchanged.

Gossip protocols are used for peer discovery and data exchange in mesh-based streaming systems. In BitTorrent systems [29], the tracker is initially relied upon for peer discovery, providing a list of randomly-selected potential neighbors on request. After establishing connections to some other participants, each peer adaptively discovers better candidates to barter streaming pieces. A gossip protocol (e.g., PEX protocol [39]) is used for this purpose. To discover candidate neighbors for improved QoS, peers exchange information about other peers with their directly connected neighbors.

Data exchange is determined by how a peer selects pieces of information to download from local neighbors. A peer should find the desired streaming data before its playback deadline. Gossiping with

neighbors is an effective way to accomplish this. Each peer independently finds and requests the data it needs, based on gossiped piece information.

6.2.2 Vulnerabilities in Mesh based Streaming Systems

Cheating attacks come from internal attackers since peers rely on gossiping among other peers. Once a newcomer joins in the streaming overlay network, it is assumed to be legitimate. The newcomer, thus, inherits all privileges to utilize protocols for the streaming service even though it is malicious. Malicious attackers easily interrupt ways of redistributing and searching content data from other compliant peers through spreading false gossip.

For content distribution, cheating attacks are divided into three malicious behaviors; fake reporting, omission (repudiation) attack, and fake block attack. Fake reporting (e.g. under(over)-reporting [8, 9]) is an attempt to modify a gossip message, breaking the condition of data integrity. Omission attack (repudiation attack) [6] is an attempt to deny the receipt of data. Fake block attacks occur when an attacker replaces genuine blocks with fake (bogus) ones, and releases them [11]. As a result, the peers obtaining the fake blocks experience considerably degraded QoS.

From the aspect of searching, if malicious attackers are involved in the participants in a BitTorrent-like swarm, it remains questionable whether to achieve an effective neighbor selection. In PEX [39] based on random walks (e.g. neighbor gossiping) to extend a local view, searching participants does not offer a view over one-hop neighbors. Liars can easily deceive an independent peer. For example, a liar invents PEX message containing malicious attackers, and shares with its neighbors. The victims unconsciously attempt to establish available connections with malicious participants. This problem is even worse if the membership server (e.g. the tracker) is itself attacked. In addition, membership protocol can be compromised; one example is an eclipse attack [108] in DHT based systems. Attackers control a portion of the overlay routing, eclipse compliant peers by dropping or re-routing any messages.

6.3 System & Threat Models

6.3.1 System Model

We consider a mesh-based streaming system with a BitTorrent-like swarm. An overlay network forms a *swarm* that consists of a streaming server, a tracker, and peers. The streaming server breaks the content (video file) into even-sized pieces, and creates a hash for each using the SHA-1 function. These hash values are recorded in the *metainfo* file, and are used for ensuring data integrity. The tracker is responsible for bootstrapping newcomers, tracking the progress of neighbors, and coordinating the initial neighbor selection. The tracker maintains a list of the currently-active participants in the swarm, and on request provides peers (e.g., a newcomer and a peer who has insufficient number of neighbors)

with a list of randomly-selected neighbors. Such servers are trusted, and are therefore assumed to provide correct data.

We assume that there are two types of participants in the content swarm. A *compliant* peer faithfully implements protocol descriptions, and cooperates with other peers. It has a limited set of neighbors (peer set), and establishes connections to them. It keeps in contact with the neighbors to know their current states through the exchange of control messages. Using this information, a compliant peer makes continuous decisions about which pieces to download and which peers to unchoke (upload to). A *Byzantine* attacker, called a *liar* deviates from the protocol, and may deliver arbitrary messages. It deceives compliant peers, attempts to mislead their decision making, and fails to perform expected actions. For example, such an attacker may exploit *fake-reporting* (e.g., under(over)-reporting [9]), message omission [6, 7], and data pollution [11].

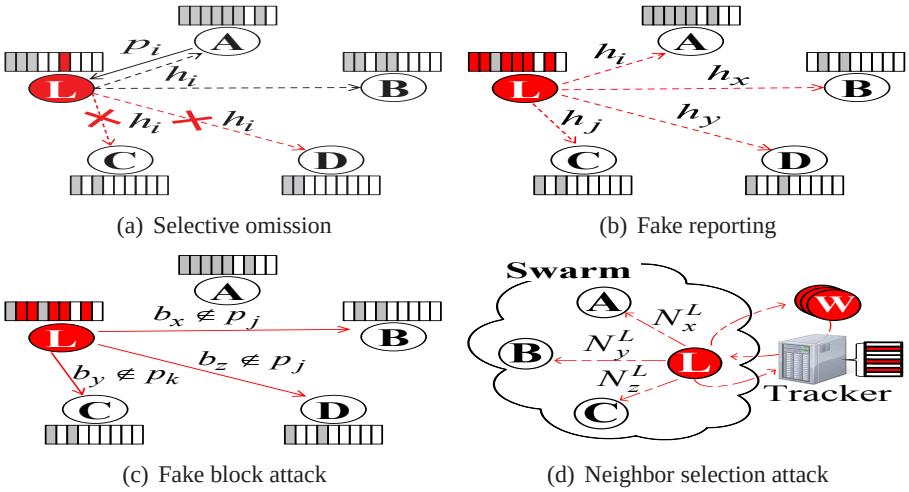
We assume a deterministic model for each download of streaming data. A peer *A* records a piece state in the local log, whenever obtaining a streaming piece. The number of pieces obtained by *A* monotonically increases until it has downloaded the entire stream. If *A* obtains a streaming piece p_i, its state is updated. After advertising the changed state, *A*'s neighbors reflect the current view of *A* in their own log. Peers can compare views to measure *A*'s consistency. *A* is considered as *correct* if its current view is consistent and agrees with the downloads provided by other peers. Otherwise, *A* is considered as *faulty* or *suspected*. A more complete description is provided in Section 6.4.3.

A mesh-based overlay network is assumed to be asynchronous, where messages between correct peers are delivered with a bounded delay. In case of a lost message, a correct peer retransmits the lost one until it is delivered. Participants ignore messages that are not part of the normal protocol. Each participant has a public/private key. A message signed with a private key provides non-repudiation (i.e., forgeries are detectable). A cryptographic one-way hash function has collision resistant and pre-image resistant properties. Adversaries are assumed not to be able to break cryptographic primitives.

We consider a restricted adversary model with collusion. A swarm S in the given mesh-based streaming system contains a bounded percentage of liars $0 \leq L < S/3$. This assumption is consistent with the bounds on most Byzantine agreement solutions [63, 71, 74]. In the BitTorrent-like swarm liars behave arbitrarily. They can act individually and collude with one another. Participants are not trusted, but their messages may be authenticated by means of digital signatures.

6.3.2 Threat Models

We assume that a liar is an inside attacker in a mesh-based streaming system. The ultimate intention is to subvert the system through cheating attacks; that is, by *selective omission, fake reporting, fake block, and neighbor selection attacks*. Such malicious behaviors will make it difficult for compliant peers to achieve a good quality of service. As assumed in [11], if peers often experience quality below their expectations, they will leave the system, and streaming (resource) capacities decrease. Such attacks



(a) Selective omission (b) Fake reporting

(c) Fake block attack (d) Neighbor selection attack

Figure 6.1: Cheating strategies in a mesh-based streaming system. A circle represents a peer, a dotted line for a control message, and a solid line for streaming data. In Fig.6.1(a), a liar L reports the receipt message h_i to neighbors A and B, but omits h_i to local neighbors C and D. Fig.6.1(b) shows that L generates and pushes fake Have messages (h_i, h_j, h_x and h_y) to deceive local neighbors A, B, C, and D. In Fig.6.1(c), L uploads a fake block of the piece requested by neighbors. In Fig.6.1(d), L informs a corrupted PEX message ($N^L_{x/y/z}$) that include L's colluders. L compromises participants list maintained by the tracker. If L is detected, it leaves the swarm, and rejoin through whitewashing.

eventually lead to system collapse.

Threat models in which a liar deliberately deceives peers through fake data are shown in Figure 6.1. Peers unknowingly experience cheating attacks during message gossip and piece exchange. A liar misinforms peer decisions, and makes compliant peers waste bandwidth and time.

Selective omission

A liar L can selectively omit an update message so that its neighbors do not gain necessary data, as shown in Fig.6.1(a). For example, if a peer A does not receive the acknowledgment message h_i (the *Have* message in BitTorrent) after supplying L with streaming piece p_i, A will suspect L's behavior. An astute L will easily avoid the suspicion by sending h_i to A. For the purpose of enhancing the *interest* relation[1], L may also send h_i to peer B. On the other hand, L may omit sending h_i to neighbors C and D to reduce their chance of obtaining piece p_i. The neighbors cannot know whether or not L obtains p_i without L's reporting. Selective omission increases the time to download needed pieces, blocks expected redistribution of streaming resources, and degrades QoS of compliant peers.

[1]If A has a piece and B does not have the piece, B takes an interest in A.

Fake reporting

L may send neighbors false control messages, such as *Bitfield*[2] and *Have* message, to increase interest relations and establish or prolong connections unproductively. Typically, a gossip message pushed by a neighbor is accepted without a verification step. Fig.6.1(b) shows fake reporting. L may omit or send a random Bitfield to a new neighbor after establishing a connection. If omitted, the neighbor regards L as a newcomer. Otherwise, it computes piece differences, and decides whether or not L is interested. If L sends fake Have messages, some neighbors may select L as one of their unchoke recipients with expectation of the subsequent reward of interested pieces. Then, L can consume their upload bandwidth, and ignore the expectation of reciprocation. Despite the lack of reciprocation, L's neighbors may keep the connection because L's reported pieces are attractive. As a result, fake reporting results in keeping a futile connection, wasting upload bandwidth, and reducing the chances of peer cooperation.

Fake block attack

Recall that a mesh-based streaming encourages peers to exchange data with a simple give-and-take incentive (i.e., TFT scheme is based on a fair exchange in bilateral relations). Each peer monitors and rewards its neighbors to obtain required data within a scheduled time. Since each piece is additionally divided into equal-sized blocks (the unit of data transmission), a peer may download blocks of a single piece from multiple neighbors. Such a parallel download approach helps peers download streaming data efficiently. The peer checks data integrity using the corresponding SHA1 value in the metainfo, after completely downloading a piece.

However, a liar L can exploit the weakness of the parallel download with a piece pollution. L may send fake (corrupted) blocks in order to degrade QoS by consuming download bandwidth and buffering time, as shown in Fig.6.1(c). Since neighbors may already be interested in L due to fake reporting, L receives some streaming data. To avoid the penalty from Tit-for-Tat (TFT) incentives, L uploads fake blocks. The victims have to download the remaining blocks, possibly from multiple neighbors, before noticing the violation of data integrity for the piece. The hash of the piece, however, does not allow the peer providing corrupted block(s) to be identified uniquely. The fake block attack expends a non-trivial amount of bandwidth, buffer space, and reassembly effort. The victims have to discard the corrupted piece, and attempt to re-download a genuine one.

Neighbor selection attack

The goal of neighbor selection attacks is to persuade compliant peers to establish neighbor relations with as many liars as possible. Liars may deceive the tracker, and directly connected neighbors, as shown in Figure 6.1(d). On joining a swarm, L establishes connections with randomly selected neighbors

[2]Bitfield messages indicate how many pieces a peer has at that moment

provided by the tracker. L accepts all incoming connection requests, and continuously requests potential neighbors from the tracker using the large-view exploit [4]. In addition, L periodically sends a heartbeat message to the tracker. If detected and penalized, L may whitewash (assume a different identity), and re-join the swarm [109]. Although the tracker is trusted, a random neighbor list it provides to a newcomer may thus contain liars. Whenever a compliant peer tries to discover unknown participants from its neighbors, L may send a PEX message that mainly includes colluders (other liars). The compliant peer will have no means of detecting this bias.

6.4 AntiLiar: Defense against Cheating Attacks

This section presents a defense against cheating attacks in mesh-based streaming systems. We call the approach *AntiLiar*. The purpose of AntiLiar is to sustain stable QoS for compliant peers in a practical manner. Each peer keeps a secure progress log, and advertises the most current state of obtained pieces. Peers can detect an abnormal report through a verifiable evidence sent from one-hop neighbors. In order to distinguish willful omissions in an asynchronous network, AntiLiar uses non-repudiation of verifiable evidence. Block filtering negates adverse effects of fake block attacks. The proposed scheme helps peers to improve peer cooperation and protects against lying. The notation of the proposed scheme is summarized in Table 6.1.

Table 6.1: Summary of notation

F	Cryptographic one-way hash function
V	Video streaming content being shared by a swarm
N^A	A's local neighbors (peer set)
n	# of streaming pieces ($V = \{p_1, p_2, \ldots, p_n\}$)
m	# of blocks of each piece ($p_k = \{b(k,1), b(k,2), \ldots, b(k,m)\}$)
d_k	Digest of piece p_k ($d_k = F(p_k)$, where F is SHA1)
h_k	Have message of piece p_k
BF^A	A's BitField message
O^A	A's order of downloaded pieces
S_R^A	Signature seed that is randomly chosen by peer A
S_0^A	A's peer ID ($PID^A = F^{n+1}(S_R^A)$)
S_i^A	A's symmetric key used for signing the i^{th} piece
C_i^A	i^{th} commitment in A's progress log
α_i^A	i^{th} authenticator signed with A's private key ($\alpha_i^A = [S_i^A \mid C_i^A]_A$)

6.4.1 Basic Idea

AntiLiar provides peers with a deterministic model for tracking neighbors' progress states. AntiLiar uses a *progress log* for the detection of attacks. The progress log is associated with a signature, a piece digest, and a history of previously downloaded pieces. Each peer provides an identity and is assumed to create a hash chain for use as a source of one-time keys. These keys can then be used for cryptographic signing. After downloading and verifying a piece, the peer appends another entry in the progress log. Based on the commitment of entries in the progress log, participants are able to verify a peer's state.

Gossip messages have the drawback that they cannot be attested without verifiable evidence. The interaction between two communicating parties is assumed to be observed only by them. AntiLiar expands the range of knowledge of interactions to one-hop neighbors the use of evidence. Once downloading a piece, a peer acknowledges the uploader with a signed receipt, called *authenticator*. The uploader can attest to one-hop neighbors that it has provided a piece to a peer by means of the authenticator.

In AntiLiar, a peer tries to maintain a consistent neighbor view. Available resources are restricted by the number of neighbors (a peer set). Adaptive and flexible neighbor management is essential to achieve good QoS under changing conditions in a content swarm. The expansion of its local view helps a peer to find useful neighbors. This may be achieved by assistance from the tracker, or from neighbors. The reliance on information from neighbors is problematic, as they are not trusted. AntiLiar uses a decentralized approach defending against false neighbor information, by comparing new information with previously-collected information.

AntiLiar adopts *block filtering* to prevent piece pollution [46]. A *block filter* summarizes all blocks of each piece in the shared content, and is provided in the metainfo file. Peers are able to detect fake blocks quickly using this information. Block integrity is inspected by repeating hash operations and checking that the expected indexes in the filter are set.

6.4.2 Peer ID & Signatures

In order to identify liars triggering cheating attacks, user authentication is required. Digital signatures using public keys are fairly expensive AntiLiar primarily uses symmetric keys generated by hash-chaining for message authentication, which substantially reduces computational costs. Public key signatures are reserved only for the purpose of non-repudiation.

Each participant must have a unique peer ID (*PID*), used to distinguish participants. PID (e.g., 20-byte string by SHA-1) is a random hash value. A newcomer A to the swarm begins by randomly choosing a sufficiently large seed S_R^A. A iteratively applies a one-way hash function F, starting from this seed, to create n keys, plus the PID. The last-generated value S_0^A in the resulting hash chain is A's PID ($PID^A = F^{n+1}(S_R^A)$), which is made public to the swarm. A peer is expected to have different PIDs, and different hash chains, for each swarm it joins.

With knowledge of a PID for B, a peer A may verify a signature created by B by first verifying

that the key S_k^B used by B for signing is legitimate. It does so by iteratively computing $F^k(S_k^B)$ and comparing it to B's PID (i.e., $S_0^B = F^k(S_k^B)$). Note that if another key S_i^B, where $0 < i < k$, is already released, then A can verify it by applying the one-way function F $k - i$ times. From the properties of hash chains, it is cryptographically easy to verify a key, but cryptographically hard for another peer to guess or undetectably forge B's keys, before they are revealed by B. Once S_k^B is verified, a message signed using S_k^B may then be verified.

AntiLiar prevents liars from impersonating with previously released keys, through a digital signature at the connection establishment stage. A reflection attack[3] (or replay attack[4]) for masquerading as another peer is possible if a liar collects and uses that peer's released keys. To foil this attack, two communicating parties confirm their most current state through signed messages at the initial stage of peering. When a peer A exchanges a Bitfield message to a peer B, A signs (using its private key) and sends a message indicating the current state ($[O^A|S_i^A|C_i^A|ts]_A$). Since a timestamp ts restricts a message validity within a small skew range, a replay attack is prevented. B verifies the signed message using A's public key (and A may do likewise for B). Use of B's keys for singing messages containing some other initial state can quickly be detected by A. Such a digital signature blocks a reflection attack as well as the initial state confirmation prevents a replay attack.

6.4.3 Progress Log

A progress log is a commitment tree that chronologically reflects a peer's progress state. Each commitment C_j consists of a signature created from S_j, a piece digest d_k^j, and the previous commitment C_{j-1}. The signature is an intermediate value created from a key that is part of the hash chain generated by a peer. The piece digest is a SHA1 value that is used for piece integrity. As shown in Figure 6.2, the initial commitment $C_1 = F(S_1 \mid d_k^1))$ is a signature, consisting of the digest of the first downloaded piece and the first key in the hash chain. Whenever downloading a piece, a new commitment $C_j = F(C_{j-1} \mid F(S_j \mid d_k^j))$ is added where | indicates concatenation.

The purpose of the progress log is to authenticate the message source and ensure message integrity. A peer shares its most current commitment, and key in the hash chain, with neighbors after downloading a new piece. Each neighbor is able verify such a message by applying the hash operation from the peer's PID, and previously-received commitments. If a peer D obtains a piece from a peer U, an *authenticator* is created. The authenticator α_j^D = $[S_j^D \mid C_j^D]_D$ (the notation $[message]_D$ indicates a message signed with D's private key) is created, which means D surely obtained a piece and added an entry to its progress log.

In AntiLiar, an authenticator is used as an evidence to detect cheating attacks. The transaction of streaming data is conducted between the uploader and the downloader. Using an authenticator the

[3]It is a method of attacking an authentication protocol by reusing collected keys. To impersonate another peer an attacker attempts to trick a victim peer with a symmetric key obtained from eavesdropping.

[4]An attacker copies messages between communication parties and replays them to other peers for masquerading.

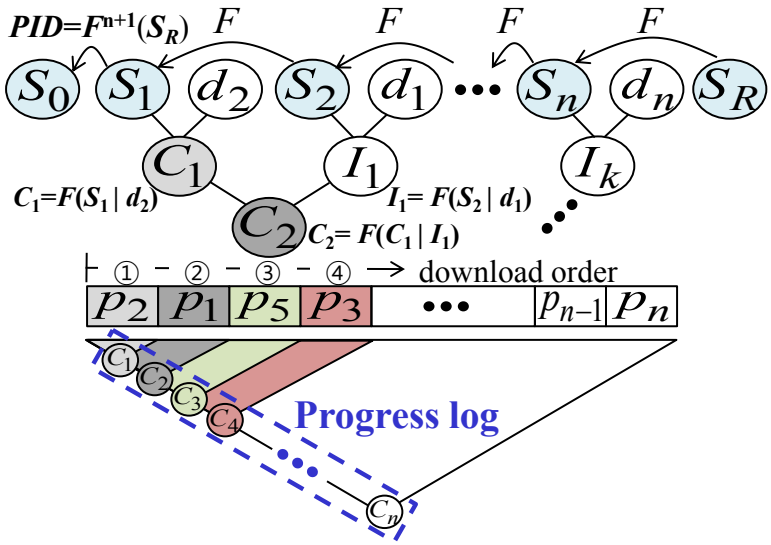


Figure 6.2: Example of updating of progress log. When joining the content swarm each peer generates its own ID (PID) using one-way hash function (F) with the initial random seed. Progress log reflects the most current state of holding pieces into a hash value, called *commitment* (C_j). Each commitment of pieces is associated with a piece digest and a peer's signature in the hash chain.

coverage of gossiping is expanded to one-hop neighbors. After D downloads p_k, the uploader U receives the authenticator α_j^D, S_j^D, and C_{j-1}^D in return. U then verifies the authenticator with D's public key. Additionally, U checks the integrity of C_j^D in the authenticator by hashing the transferred messages and the piece digest that it uploaded. If verified, U advertises α_j^D, j (i.e. download index), and $S_0^D (= PID^D)$ to D's neighbors. At the same time, D notifies local neighbors via a Have message that contains a signature S_j^D and a commitment

In an asynchronous network, AntiLiar faces the additional problem of distinguishing message loss from omission attack. To tackle this problem, AntiLiar uses non-repudiation by an authenticator. Since a one-hop neighbor informs an authenticator, the neighbor expects a Have message including a new signature and commitment. If a neighbor notices the absence of the message, it can suspect a local neighbor who is supposed to advertise the next signature and commitment. In the case the message is omitted, there are two possible causes; an omission attack or a simple message loss. To establish the cause, the detector requests the most current commitment and signature from the neighbor. The detector then can receive them as long as the neighbor is compliant. If the neighbor repeatedly omits the message or denies having the piece, the detector adds it to a blacklist (attacker list), and disconnects the neighbor.

AntiLiar defends against fake reporting. Suppose that a liar L arbitrarily reports a Have message with a fake signature and commitment. Before accepting the message, a neighbor waits for a matching authenticator. If L's colluder sends a fake authenticator, the neighbor will update L's state. However, L's false evidence will remain thereafter; since a progress log is append-only, a corrupted entry in the

log cannot be modified. Moreover, the neighbor is able to detect inconsistency because of the evidence connected to the fake messages.

6.4.4 Consistent Neighbor View

Each peer maintains a limited set of participants as neighbors in the swarm. The state of local neighbors is immediately updated through direct experience. If a neighbor disconnects a TCP connection or does not reply to a heartbeat message, the neighbor in the peer set is promptly removed. Each peer can keep a consistent local view without intervention. However, available resources of streaming data are limited to only local neighbors. For effective data trading, finding a compatible neighbor with mutual interests is essential to achieve good QoS.

AntiLiar mainly applies decentralized approaches for peer discovery. Each peer periodically exchanges its own peer set through the PEX protocol [39]. Indirect observation of neighbors expands a local view to one-hop ranges. Such a approach simply discovers the presence of participants in the swarm, which cannot know compatibility of interest until establishing a connection. In addition, a referral scheme [103] improves discovering compatible peers without establishing connections. The referral scheme is restrictively utilized based on relative seniority under asynchronous interests.

The neighbor selection attack, however, compromises an expanded view. To tackle this problem, AntiLiar maintains a *consistent neighbor view*. Suppose a liar L attempts to use a large-view exploit. Then L will likely repeatedly occur in a expanded view. This does not however prove that L is deliberately engaged in a neighbor selection attack. To determine whether a neighbor selection attack happens, a peer A investigates intersection comparing previously-collected information of peers' relation with a new reported peer set N^L. If N^L does not include common peers, A suspects L.

On the other hand, if L does not use a large-view exploit, there may be no intersection, although L is still providing false information. A then randomly selects one of its one-hop neighbors in N^L, and asks that neighbor for the most current peer set reported by L and the recorded timestamp. A can identify inconsistency by comparing these pieces of information, as long as it has not inadvertently selected a colluder of L.

AntiLiar adaptively manages a swarm view. Although a mesh-based streaming has resilience under dynamic membership, the quality of service is partly affected. In the situation of that a neighbor who is a seeder or a bartering partner leaves, adverse effects are increased. With referral information, AntiLiar adaptively replaces neighbors during the lifetime. Each peer independently decides to replace either a liar or a leaver to a compatible peer. A one-hop view is liable to become stale from dynamic membership. PEX and referral messages make a expanded view up-to-date. Note that one-hop expansion has a trade-off between overhead and consistency.

6.4.5 Block Filtering

AntiLiar uses *block filtering* to detect fake block attacks in our previous work [46]. Block filtering enables peers to quickly detect a fake block sent by a malicious polluter. A *block filter* (B_{Filter}, i.e., a Bloom filter) is a summary of all blocks of each piece in the content. As shown by Algorithm 3, the streaming server creates B_{Filter} for the streaming content. The streaming server performs k hash functions for each block, and marks the corresponding k bits in the block filter. After processing all blocks, B_{Filter} is added to the metainfo file provided to each peer. Note that since a newcomer obtains the metainfo file before bootstrapping, all participants can inherently use B_{Filter} in the swarm. From this, they can identify whether a block obtained from the neighbor is fake. Verification of a block involves repeating the hash functions and checking that the expected k bits in the filter are set.

Algorithm 3 Block filter creation

1: $V = \{b(1,1), b(1,2), ..., b(n,m)\}$
2: $B_{Filter} \leftarrow \{0\}$
3: **for** $i = 1$ to n **do**
4: **for** $j = 1$ to m **do**
5: $Index_k = H_k(b_{i,j})$ where $k = 1, ..., l$
6: $B_{Filter}[Index_k] \leftarrow 1$
7: **end for**
8: **end for**

6.4.6 Costs Analysis

We analyze AntiLiar's overall costs in a mesh based streaming system. Since actual costs depend on the frequency of cheating attacks, the cost analysis is presented only for the no-attack model.

Communication cost

The communication cost for downloading a stream includes the payload, and control messages containing Have information, Bitfields, authenticators, and evidence. The payload contains file pieces (p_i). The Have and Bitfield messages for AntiLiar's functions are slightly modified. The Have message contains a naive piece index, a one-time signature, and a commitment ($Have = h_k \mid S_i \mid C_i$). The Bitfield ($BF$) has an original Bitfield message, the order of downloaded pieces (O), the most current signature, and commitment ($BF = original\ BF \mid O \mid S_i \mid C_i$). The authenticator is signed with a downloader's (D) private key ($\alpha_j^D = [S_j^D \mid C_j^D]_D$). The evidence message contains authenticator, an index of downloaded pieces, and the PID ($Evidence = \alpha_j^D \mid j \mid S_0^D$).

The total communication cost depends on the number of neighbors ($|N|$), the number of streaming file pieces (n), and the number of changing local neighbors (δ). Given the parameters $|N^A|$, $|N^x|$ (number of neighbors in x's peer set), n, δ, $|p_i|$ (size of a file piece), and $|F(\cdot)|$ (size of a hash function image), the total communication cost of a peer A can be determined as follows:

$$\begin{aligned} Communication\ cost = n \cdot |p_i| + n \cdot |N^A| \cdot |Have| \\ + \delta \cdot |BF| + n \cdot |\alpha| + n \cdot |N^x| \cdot |Evidence|. \end{aligned}$$

Processing delay

The processing delay includes generation and verification of a digital signature and hash operations. Each peer performs a digital signature operation for the defense against cheating attacks. Whenever a peer A downloads a file piece, it generates an authenticator; therefore, n digital signatures are required. Each neighbor ($x \in N^A$) must verify each such authenticator as long as it is connected. In addition, each peer performs hash operations for information integrity and authentication. To add an entry in a progress log, two hash operations are needed. When a peer verifies a commitment, it performs three hash operations (one for verifying a one-time signature, and two for verifying a commitment). When exchanging Bitfield messages, two communicating parties verifies each other's state. This verification requires three times the total number of the other's downloaded pieces hashes. For block integrity, a peer performs k hash operations per block.

Storage cost

The storage cost for AntiLiar includes a progress log and block filter. A progress log contains commitments, signatures, and digests of all pieces obtained ($Cost_{PL} = \sum_i^n (|C_i| + |S_i| + |d_i|)$). The size of block filter depends on false positive rates of Bloom filter. The block filter that maps $n \cdot m$ blocks through k hash functions is a y-bits string. Note that n is the number of pieces and m is the number of blocks in a piece. According to [40], false positives (fp) can be derived:

$$fp = (1 - (1 - \frac{1}{n \cdot m})^{k \cdot y})^k \approx (1 - e^{\frac{-k \cdot y}{n \cdot m}})^k.$$

False positives depend on how many bits are used for indexes, how many blocks are in, and how many hash functions are used. Given the values of y, n, m, and the desired false positive rate fp, the optimal size of y is

$$y = -\frac{(n \cdot m) \cdot ln(fp)}{(ln2)^2}.$$

Actual costs that contain communication, computation, and storage overhead are illustrated in section 6.5.2.

6.5 Evaluation

This section evaluates the robustness and costs of mesh-based streaming in the presence of cheating, by means of detailed simulations. The features of AntiLiar are added to a mesh-based streaming system called CoVoD [103], which uses an adaptive neighbor management protocol. Other methods of defending against attacks or failures are implemented as well, for purposes of comparison. Although an actual deployment provides real patterns, results cannot be limited in effect ranges of cheating, and a real system will unexpectedly suffer from cheating to normal users. Therefore, the performance results of liars' effects are shown by simulation. The experimental conditions are first described, followed by the results.

6.5.1 Experimental Setup

An event-driven simulator was developed for mesh-based streaming under the presence of malicious attacks. In the simulation, basic protocol operations were modeled as events. The events include joins, leaves, rejoins, playbacks, sending a piece, and sending a control message (i.e., Have, Bitfield, (un)choke, (un)interest, and PEX messages). The malicious attacks described in Section 6.3.2 were also implemented, i.e., fake control messages, sending fake blocks, omitting messages, delaying pieces, and whitewashing. To reduce simulation complexity, messages sent from a sender had a configurable delay, while transmission errors and network bottlenecks were not considered.

Liars were assumed to have static identities, or to use whitewashing. In the case of a liar using a static identity, participants could effectively prune the attack link after detection. However, whitewashing incapacitates peers from penalizing the detected liar. To maximize attack impact, liars could also use the large-view exploit [4] for establishing more attack links. Liars were assumed to engage in productive piece exchanges to increase TFT incentives. Following that, they may strategically change their behaviors to mislead unchoking selection, based on TFT incentives. Additionally, liars were assumed to collude with each other. To explore cheating impacts, the fraction of liars was varied from 10% to 30%.

The proposed method is compared to two different models, BiToS [86] and PeerReview [70]. BiToS is a "baseline" method that supports basic BitTorrent protocols, such as TFT and LRF including a window-based piece selection, for video distribution. This model does not include any defense against cheating attacks. PeerReview was proposed as a method to detect Byzantine attacks. Since PeerReview was initially applied to multi-tree streaming, we adapted its functions (based on [110]) to mesh-based streaming. The witness set was two peers, which were randomly selected. Although all input and output messages are originally considered as entries in a tamper-evident log in PeerReview, our experiment limited the messages logged to those related to pieces and neighbors for a fair comparison. With AntiLiar each peer maintains a progress log, verifies gossiped messages, and validates its neighbors. PeerReview and AntiLiar were based on an adaptive neighbor management protocol [103], although such methods could be adapted in any mesh-based streaming system. This protocol uses a referral to find compatible

neighbors, and provides the better quality of services than BiToS.

Each experiment started with one streaming server, and one tracker [103, 85]. The streaming server distributed a single video. It generated a block filter for block integrity before sharing the video content. The tracker informed a random set of $|N|$ potential neighbors when a peer joined the system, or a peer had less $\lceil |N|/2 \rceil$ neighbors. These servers served all participants in the swarm throughout the experiment. Peers joined the content swarm based on a Poisson arrival rate, with an inter-arrival time (λ) set to 4 seconds as in [85]. Each compliant peer had at most $|N| + 5$ neighbor connections, while liars were able to increase neighbor connections using the large-view exploit. Participants selected five unchoking recipients[5], every unchoke period (5 seconds, which is the same as rechoke cycles). A newcomer filled streaming data in a video buffer memory after joining the swarm. Once passing a startup delay of ten seconds, the newcomer started playback. Peers simultaneously performed playback and downloading. When a peer completed downloading the content, it became a seeder, and supported other participants. After finishing the playback, it was assumed to immediately leave the swarm.

Parameter settings were taken from previous studies [103, 85, 86, 84]. The total number of participants was fixed to 3,000. The number of participants in the swarm maintained fewer than 400 peers at any one time, which is based on a recent measurement study [100]. The shared video file was 10 minutes long and the streaming encoding rate was fixed to 500Kbps. The content was evenly divided into 64KB pieces, and had 600 pieces. Each piece is further separated into 16KB blocks as a transmission unit. The upload capacity of the streaming server was fixed to 5Mbps and a peer's upload capacity of 600Kbps was used. These settings are realistic, and lead to good performance when the mesh-based streaming system is not under attack. In addition, we modeled the heterogeneous case with standard ADSL speeds from 500kbps to 1Mbps.

The above parameter settings were used in all simulations, for all methods, except when stated otherwise. In this chapter, the two properties of interest are primarily protocol costs and service quality of mesh-based streaming. The costs were divided into communication overhead, storage overhead, and processing overhead. For the service quality the *continuity index (CI)* introduced in [41] was used (CI = $\frac{\#\ of\ received\ pieces\ before\ deadline}{total\ \#\ of\ pieces\ in\ the\ content}$). Points in each figure show the mean of a specified measurement, and I-shaped 95% error intervals.

6.5.2 Experimental Results

Service quality

Acceptable service quality is the main goal in a mesh-based streaming system. A peer having a continuity index (CI) of 1 means that it has experienced seamless (uninterrupted) streaming service during

[5]In BiToS, the streaming server selected five peers for optimistic unchoking, and peers allocated four peers for TFT unchoking and one peer for optimistic unchoking. In PeerReview and AntiLiar, the streaming server assigned five slots with referral unchoking, and peers selected three TFT unchoking and two referral unchoking peers, based on results in [103].

its playback. If the CI is less than 1, the peer has experienced some playback jitter or quality loss, with a CI of 0 meaning that no portion of the stream was available for playback. Note that a traditional client server model usually supports seamless service to all participants as long as a set of servers can accommodate them. However, mesh-based streaming is based on decentralized approaches for service quality without the dependance of the dedicated reliable servers. Thus CI is useful to evaulate different target protocols in mesh-based streaming systems. In this experiment CI was measured under several conditions.

The first experiment investigated the average CI of all compliant peers as a function of the percentage of liars in the system. Figure 6.3 shows the comparison of three methods. On the one hand, three methods provided good quality of services to each participants when there was no attacker. In the detailed view in this Figure, AntiLiar had the best CI followed by BiToS and PeerReview. Although two advanced methods were based on the same protocols by CoVoD [103], the results had different patterns. BiToS achieved better QoS than PeerReview due to lightweight protocol overhead. Peers with PeerReview used some portion of bandwidth for accountability and an eventual detection. However, AntiLiar outperformed BiToS through an adaptive neighbor management by CoVoD. This is because AntiLiar effectively discovered compatible neighbors as well as required much lower overhead than PeerReview. The overhead results are addressed in the following Section 6.5.2.

On the other hand, PeerReview and AntiLiar both effectively defended against the effects of liars. Since witness peers in PeerReview immediately propagated liar's information to participants with verifiable evidence, peers experienced good QoS. Due to cooperating with direct and one-hop neighbors, AntiLiar also maintained acceptable QoS. However, BiToS was severely affected. This is because BiToS does not have any detection or prevention mechanism against cheating attacks. Interestingly, results of BiToS are similar to a simple analytic model that shows CI variation under the fake block attack, based on [10] [6].

To further understand QoS performance, the next experiment detailed the average CIs of compliant peers with AntiLiar. Figure 6.4(a) shows the CI over time as a function of the percentage of liars. The CI of compliant peers was above .98 when the percentage of liars was less than or equal to 30%, demonstrating outstanding effectiveness for AntiLiar. The CI is degraded, although still higher than .90, when liars repeatedly change their identity to avoid the penalty from previous cheating. This degradation is caused by liars' strategic collusion and whitewashing.

Figure 6.4(b) shows CDF of liars residence time with a static identity or multiple identities. Regardless of the fraction of liars, most static liars were effectively removed in each peer set of compliant peers around 30 seconds after joining. However, whitewashers stayed approximately 10 times longer than static liars, through strategic collusion. Moreover, they cheated compliant peers, affected service

[6] Suppose a peer that attempts to download a specific piece. This peer has n neighbors (where n=35 in this case), and the probability of l liars is varied from 0% to 30%. Assume u unchoking neighbors out of the n neighbors are randomly selected. The probability of downloading the genuine piece is $\frac{{}_{n-l}C_u}{{}_nC_u} = \frac{(n-l)(n-l-1)\ldots(n-l-u+1)}{n(n-1)\ldots(n-u+1)} \approx (1-\frac{l}{n})^u$.

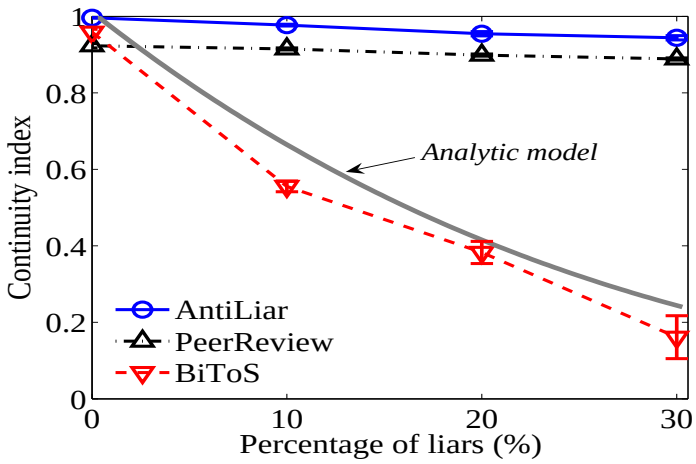
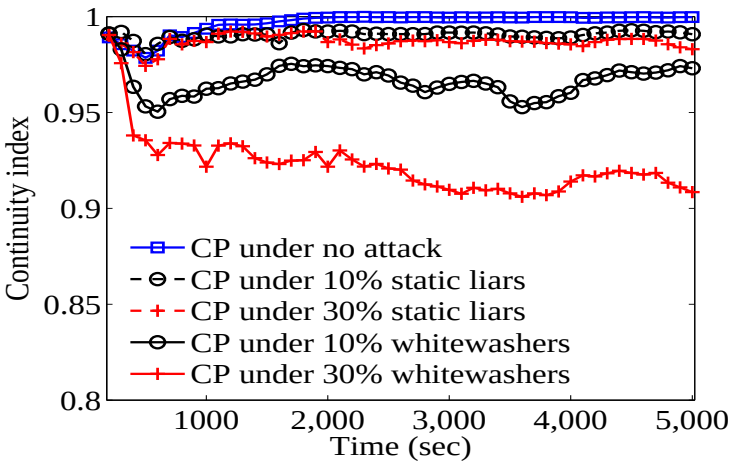


Figure 6.3: Comparison to QoS, which shows the average continuity index for peers on three methods.

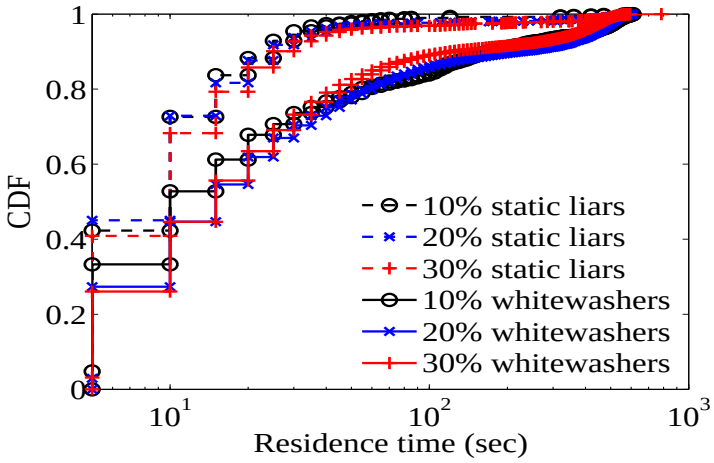
degradation about more than 6%.

To achieve good QoS, streaming resources are fully distributed to each peer. Such condition is sufficiently met, as long as participants perform maximum uplink utilization. In this experiment, the average uplink utilization of streaming resources for compliant peers was also investigated. Figure 6.5 shows uplink utilization of just streaming data as a function of the percentage of liars. This metric was calculated by the uplink used for obtaining streaming piece over the maximum uplink uprate. BiToS showed the worst results, and AntiLiar the best for all cheating models. Since PeerReview required more communication cost for eventual detection than AntiLiar, the pure uplink utilization for streaming data was relative low. As a result, AntiLiar improves more than 10% uplink rates as well as at least 5% CI.

The availability of streaming data depends on the number of local neighbors. Intuitively, the more neighbors increase, the higher chance to barter is. Figure 6.6 shows the average CI of compliant peers under 10% liars using whitewashing, as a number of directly connected neighbors. BiToS followed the above priori knowledge, while PeerReview and AntiLiar showed different results. With PeerReview peers had the best CI when they have 20 local neighbors. With AntiLiar peers resulted in near constant CI, regardless of the number of neighbors. This is because PeerReview has much heavier overhead than AntiLiar. In addition, we increased a uplink rate to 800Kbps, and investigated the CI of PeerReview. Interestingly, PeerReview had similar CIs to AntiLiar with 600Kbps when the peer set were less than 40. It was about 33% improvement of uplink resource. Overall, AntiLiar has a scalable and practical feature against cheating attacks.



(a) QoS of compliant peers



(b) Residence time of attackers

Figure 6.4: Service qualities and attacker's residence time in AntiLiar. Fig. 6.4(a) shows transition of service quality for compliant peers. Fig. 6.4(b) shows CDF of attackers' residence time.

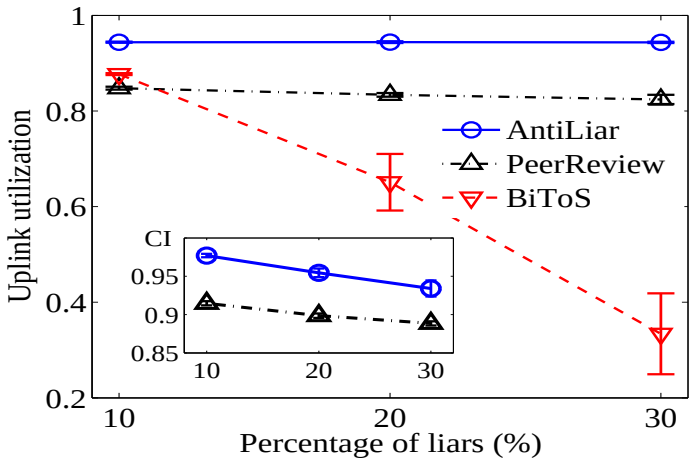


Figure 6.5: Comparison to the average uplink utilization of just streaming data for compliant peers, which shows the fraction of used for obtaining streaming pieces and maximum uplink rate by each peer. The inner graph shows continuity indexes as the results of uplink utilization.

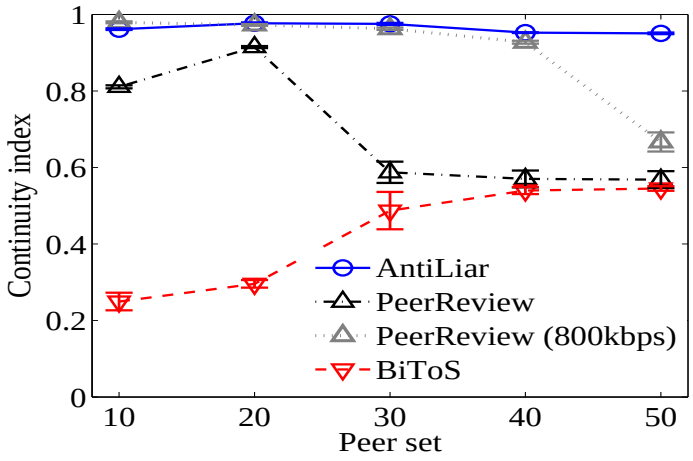


Figure 6.6: Service qualities as a number of directly connected neighbors (peer set), which shows the average CIs of compliant peers under 10% liars using whitewashing. A homogeneous model in which every peer has upload capacity of 600Kbps was used for three methods. Additionally, PeerReview was investigated with 800Kbps as another homogeneous model.

Dynamic features

The previous experiments examined the effects when compliant peers were stable in the swarm. The next experiments evaluated the effectiveness in the presence of dynamic membership and flash crowds.

During the lifetime, a peer may continually change its neighbors because of peers leaving (called *peer churn*). Following [111], it was assumed that approximately 10% of total participants left the swarm in the first half of streaming service, according to a Weibull distribution.

Figure 6.7 compares the results of the two methods in the presence of dynamic members, as a function of the percentage of liars. The top Figure 6.7(a) shows the average CIs of compliant peers. The two methods are both impacted by churn, although PeerReview much more so than AntiLiar. Witness peers leaving the swarm required to transfer their log to a new witness peer and to replay them.

To further understand the effect of churn, the bottom Figure 6.7(b) shows the fraction of compliant peers who left the swarm due to worsening QoS. This was measured as the number of compliant peers left because of low CI over the total number of peers left in the middle of video playback. This rate was approximately 6-8% higher for PeerReview than for AntiLiar.

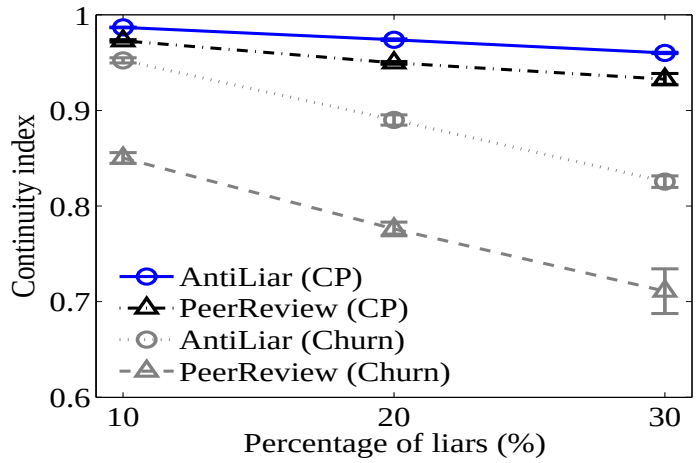
Flash crowds (i.e., many peers joining the swarm simultaneously) happen in P2P VoD systems. Such a dynamic feature can strain the distribution system, as well as degrade the service quality. To test this, two flash-crowds were modeled. The first flash-crowd arose when 200 newcomers abruptly joined the swarm during a 10 second interval, at 1,200 seconds after starting a streaming service. After that, the second flash crowd occurred when 100 newcomers suddenly joined at 3,400 seconds.

Figure 6.8 shows the change in the CIs of AntiLiar with flash crowds arrival rates over time, while varying the portion of liars. AntiLiar recovers QoS after flash crowds, although the recovery rate is significantly slowed by whitewashing. AntiLiar is resilient through effectively redistributing the newcomers streaming data. This is because peers with AntiLiar adaptively manage local neighbors using an expanded one-hop view.

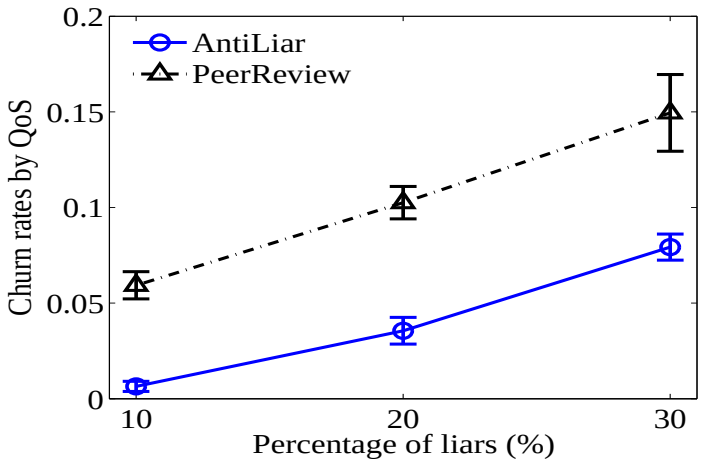
Swarm coverage

The next experiment investigates *swarm coverage* for compliant peers with AntiLiar. The swarm coverage is the average of the fraction of known peers and currently-active total participants in the system. AntiLiar adaptively manages a swarm view. Although a mesh-based streaming has resilience under dynamic membership, the quality of service is partly affected. In the situation of that a leaver is a seeder or a bartering partner, peers experience the degradation of QoS.

To mitigate this adverse effect, AntiLiar adaptively maintains swarm view during the lifetime. Figure 6.9 shows swarm coverage for compliant peers as download progresses. Although compliant peers only knew about 15% participants regardless of download progress, referral scheme continually expanded the swarm view. Since PEX expansion was effective to confirm identities of swarm participants in the first stage, compliant peers defeated neighbor selection attack. Referral containing piece differ-



(a) QoS by dynamic membership



(b) Churn rates by QoS

Figure 6.7: Service qualities and churn rates by QoS of compliant peers under dynamic membership. In Fig. 6.7(a), the gray lines denoted "Churn" indicate the results of compliant peers who left the swarm before finishing watching a video content. The churn peers followed a Weibull distribution. In Fig. 6.7(b), Y-axis indicates the number of peers left the swarm because of low CI over the total number of peers left in the middle of video playback.

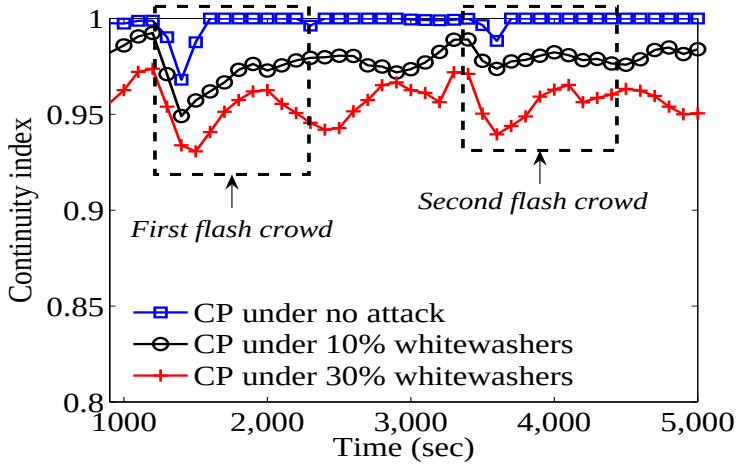


Figure 6.8: Service qualities with flash crowd arrival rates, which shows the average CIs as varying the portion of liars using whitewashing. The first flash crowd arose at 1,200 seconds of starting a streaming service in which 200 peers joined the swarm during 10 second interval. The second one occurred at 3,400 seconds in which 100 peers entered the swarm for the same interval.

ence and playback progress of one-hop neighbors enabled to find compatible neighbors for the improvement of QoS. As a result, compliant peers achieved peer cooperation despite the presence of liars and dynamic peers.

Costs

Although BiToS is simple and lightweight, it does not support a defense mechanism against cheating. Advanced methods support accountability and detection methods. In this experiment, AntiLiar is mainly compared to PeerReview, for assessing communication, processing, and storage costs.

The first experiment evaluated communication overhead caused by messages for normal operations and liar detection. Table 6.2 shows the communication overhead when all the participants in the swarm are compliant. The first column in an item indicates the number of sent messages, and the second indicates the total size (in MB) of all messages. Content data as a payload[7] accounts for most of the total size. BiToS has the least overhead. Although there are no liars in the swarm, PeerReview and AntiLiar still incur overhead. Sizes of conventional messages are slightly increased, while authenticator and evidence messages against cheating are newly added. For accountability and attacker detection, PeerReview has much heavier overhead than AntiLiar. In PeerReview, each peer must ensure that a neighbor sends or receives messages related to content data, and witness peers verify each log entry of the designated peers. However, AntiLiar sharply reduces communication costs through the use of the progress log and evidence messages.

[7]It indicates the average size of the total streaming data uploaded by compliant peers during the swarm.

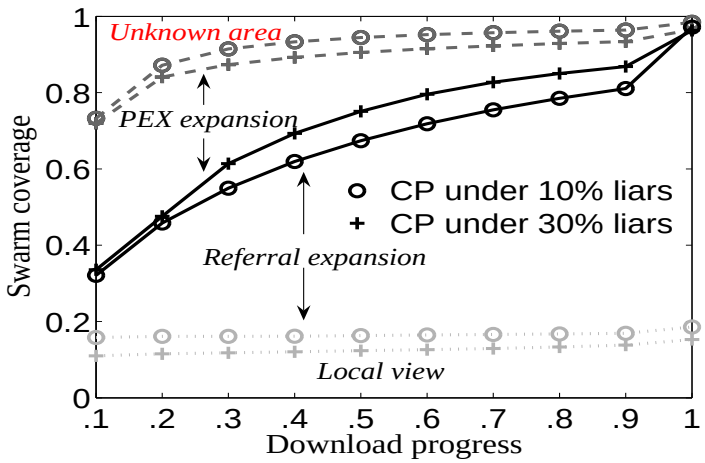


Figure 6.9: Swarm coverage

Table 6.2: Communication overhead

Type	BiToS		PeerReview		AntiLiar	
	# Msgs	Size (MB)	# Msgs	Size (MB)	# Msgs	Size (MB)
Payload[a]	2462.5	39.4	2441.1	39.0	2436.1	39.0
Have	17652.2	0.158	20065.4	0.983	19549.1	0.958
BF.	94.2	0.008	111.9	0.022	114.7	0.023
Auth.	-	-	21774.4	1.459	599.6	0.051
Evid.	-	-	32451.9	12.4	19399.5	1.3
Total	20208.9	39.566	76844.7	53.864	42099.0	41.332
Ratio[b]	1	1	3.80	1.36	2.08	1.04

[a] It indicates the average size of the total streaming data uploaded by compliant peers during the swarm.
[b] It is calculated by the ratio of each method (BiToS, PeerReview, and AntiLiar) to BiToS.

On average, control message overhead of BiToS is 166KB, while for PeerReview it was 14.9MB, and for AntiLiar it was 2.3MB. During 610 seconds (startup delay and playback duration), each peer in PeerReview had to upload 53.9MB, and AntiLiar and BiToS took up 41.3MB and 39.6MB, respectively. Total upload bandwidth for PeerReview is 30% greater than for BiToS. These results agree with the results of using smaller witness sets [70].

Note that communication overhead varies by sizes of contents. As the size of a streaming content increases, the number of both authenticator and evidence messages increases. AntiLiar proportionally increases the size of control messages ($n \cdot |\alpha| + n \cdot |N^x| \cdot |Evidence|$, where $|N^x|$ indicates the number of neighbors of neighbor x and n is the number of pieces in the content). On the contrary, PeerReview additionally adds the size of designated peers' logs and all messages related to a piece exchange as authenticators and evidence messages. Since such control messages provide PeerReview's

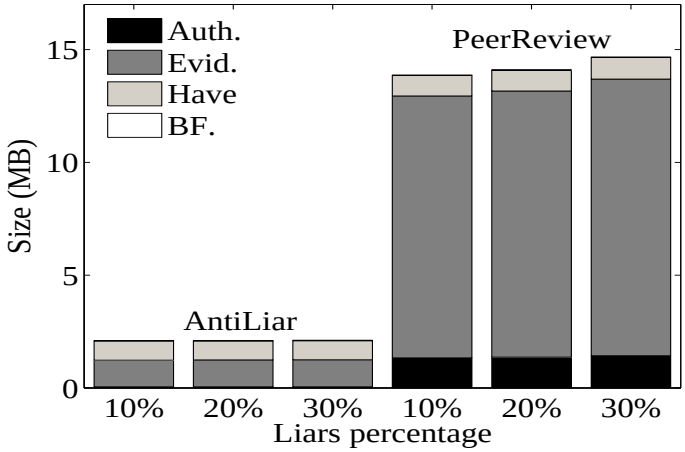


Figure 6.10: Comparison to communication overhead

basic functionalities, the communication cost sharply increases. An additional numerical comparison between PeerReview and AnitLiar remains in the future work.

Figure 6.10 shows the comparison to additional communication costs between AntiLiar and PeerReview, as a function of the percentage of liars, showing similar trends. The number of Have and Bitfield messages is similar between AntiLiar and PeerReview, while the number of authenticator and evidence messages is significantly different. The dominant overhead comes from the quantities of these messages for an eventual detection mechanism and a tamper-evident log in PeerReview. PeerReview periodically generates communication overhead for witness peers; for example, added log entries, commitments, and authenticators every 100*ms*. AntiLiar reduces additional communication messages with progress log and one-hop neighbor observations. The number of authenticator messages is adjusted to the content size for progress log, while the number of evidence messages is bounded by the number of one-hop neighbors.

To further compare neighbor scalability between PeerReview and AntiLiar, Figure 6.11 shows the communication cost as the function of the size of the directly connected neighbor set. In the aspect of scalability of peer set, AntiLiar has more scalable than PeerReview. AntiLiar incurs less communication cost although both methods proportionally increased. When a peer with PeerReview has 30 neighbors, the communication overhead increased 50% of total content size. The overhead of AntiLiar approximately increased every 600KB, whereas PeerReview increased roughly every 4MB from the initial cost of 6MB, as the number of peer set.

Unlike BiToS, PeerReview and AntiLiar carry computational overhead for defending against cheating attacks. The second experiment tested the average processing time of cryptographic operations for compliant peers to download the content file. Based on widely used cryptographic libraries for SHA1, ECDSA signature generation, and verification in OpenSSL [112], this overhead was measured. Note

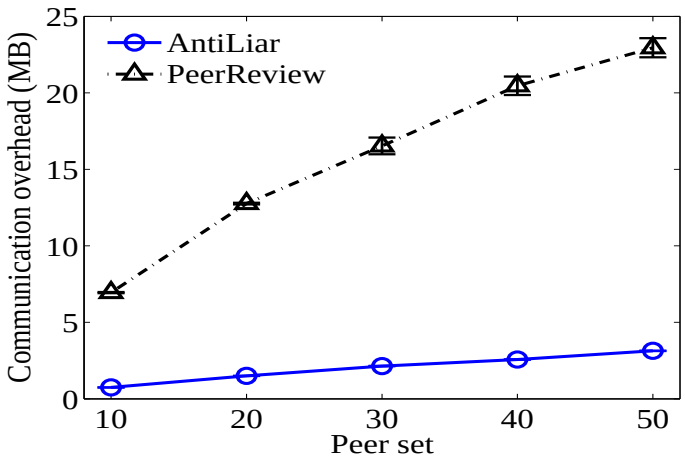


Figure 6.11: Communication overhead as the number of peer set

that PeerReview uses 1024-bit RSA digital signature, whose strength is similar to 160-bit ECDSA. For a fair comparison, ECDSA signature scheme was used for both methods. On average, SHA1 operation took $2\sim3\mu s$ on the testbed machines. Signing and verification operations were 1,000 and 2,000 times slower than SHA1. Since the process delay depends on hardware resources, the results were based on the average operation time of cryptographic operations.

The results are shown in Figure 6.12. Overall, the computational overhead was dominated by ECDSA operations. Although both methods computed a number of hash operations, they had a negligible computational overhead. ECDSA overhead was much more significant. AntiLiar restricts ECDSA operations to the number of shared pieces, whereas PeerReview incurs it for the entire set of messages related to piece exchange. Total computational overhead for PeerReview was approximately 5 times greater than for AntiLiar, and was significant enough to potentially degrade the quality of service (i.e., CPU bandwidth might in some cases be insufficient to support both verification and downloading/playback operations at the same time).

Another experiment investigated the average expected processing delay as a function of peer set size. Figure 6.13 shows these results, and again shows the growth rate for PeerReview is higher than for AntiLiar. This figure also indicates that computational overhead of verification for PeerReview may exceed residence time (the duration of video playback) under some conditions. To support scalable neighbors, peers in PeerReview must equip the more powerful process for the reduction of processing delay.

A final experiment measured the storage overhead for secure logs and evidence. In PeerReview, each peer's log was affected by the number of messages gossiped for both witnessing, and normal protocol operation. As the number of neighbors increased, the storage cost linearly increased from 2.7MB to 17.6MB. In contrast, a peer with AntiLiar just updates its own progress log for obtained

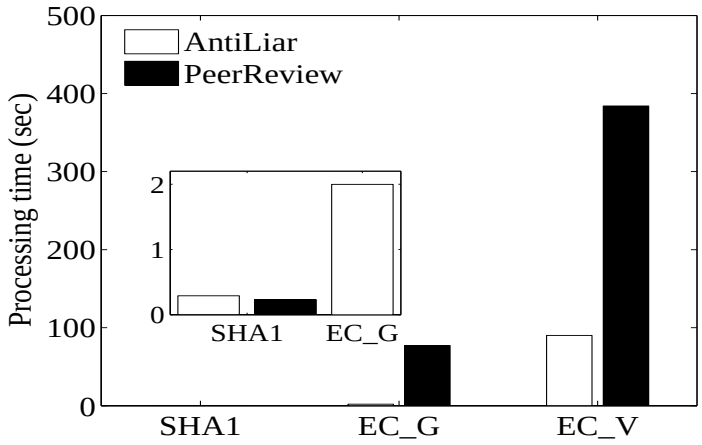


Figure 6.12: Processing time of cryptographic operations. The inner graph magnifies the results.

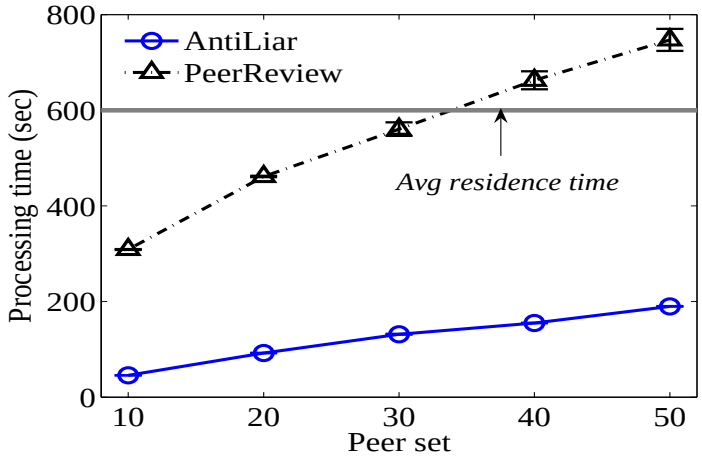


Figure 6.13: Average expected computational delay as a function of peer set. The gray line indicates the average residence time of compliant peers in the swarm.

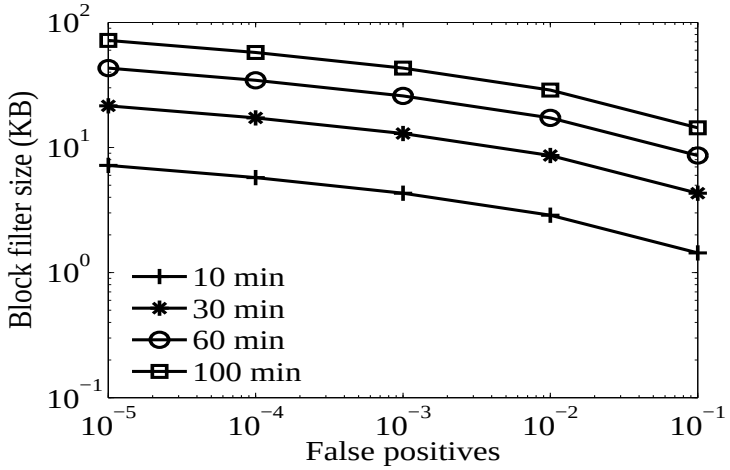


Figure 6.14: Size of Block filter by false positive rates

pieces. Regardless of the number of neighbors, the storage cost of a progress log was approximately 26.4KB.

In addition, AntiLiar deployed block filter to combat fake block attack. The accuracy of detecting a fake block depends on the size of filters. Figure 6.14 shows the size of Block filter by the expected rate of false positives, as a function of content sizes. The memory overhead of Block filter with a 10^{-5} false positive rate required 72KB for a 100 minute video content. The overall storage cost of AntiLiar for the block filter and progress log combined is extremely modest.

6.6 Comparison with Other Approaches

The Byzantine agreement problem is a well known topic of research in distributed systems. There are two categories of solutions, "classic" approaches, and more recent (practical, scalable) approaches. *State machine replication* schemes are classic approaches that keep a globally correct view among a set of servers (replicas) with event casuality (i.e., "happen-before" relations) [63, 66, 68, 74]. Although a small set of replicas are capable of having the same view, a great number of peers in P2P systems rarely achieve the same view. Even if it is possible, reaching a consistent view requires extremely high communication costs and long delays as the number of participants grows.

A more recent approach uses trusted (hardware) components [69, 8]. This approach delegates a security mechanism to a hardware module, is more efficient and scalable, and is thus more suitable for the conditions of P2P systems. However, the additional hardware represents a barrier to adoption, and raises concerns about privacy. All users must equip a trusted component in the hardware. Their machines are automatically subordinated to manufacturers, which produces a privacy issue. Hardware malfunction by an attack may cause an overall system crash [72].

Mounting evidence [70, 71] is another practical approach. PeerReview [70] provides an eventual detection scheme through witness peers. Individual logs can however be compromised by malicious behavior (fake reporting), and the issue of assigning witness peers is not clear. Restoring consistency between peers holding inconsistent views of course requires time and bandwidth. Nysiad [71] assigns a set of guards (i.e., the same role of witness peers in [70]) to each peer. This scheme, however, requires a central server creating and maintaining the guard graph, which can be a single point of failure, and a bottleneck under frequent membership changes.

SecureStream[6] adopts admission control and auditing to deal with Byzantine failures. SecureStream mainly focuses on Byzantine behavior for freeriding purposes. BAR Gossip [7] defines three types of participants (malicious, selfish, and compliant peers), and penalizes selfish and malicious peers through a random neighbor selection and a fair exchange in live streaming. BAR Gossip considers only the case of static membership, however.

AntiLiar does not require additional hardware, and provides privacy through randomly generated peer IDs. While AntiLiar also adopts accountability to detect fake reporting, AntiLiar does not have to deploy a guard (witness) server, and provides cost-effective monitoring. AntiLiar works with a rapidly-changing set of dynamic peers (churn). Finally, AntiLiar is designed for video on demand streaming, while some of the approaches mentioned are designed for live video streaming.

6.7 Summary of AntiLiar

Mesh-based streaming systems are vulnerable to attacks that can easily degrade quality of service to the point of system collapse. This chapter has proposed a method of defending against cheating attacks in mesh based streaming systems based on BitTorrent. The method makes use of a secure progress log, a verifiable evidence, and block filtering. Verifiable evidence in the progress log enables a compliant peer to perform the function of a recorder and a detector. This method is practical due to the use of hash chaining. Each peer keeps a consistent expanded view through decentralized peer discovery and peer cooperation. Compliant peers with the method experience a high (greater than 95%) quality of service, even when a significant fraction of peers are liars, and collude. Quality of service is however degraded if whitewashing is used, and remains an area for future improvement.

The next chapter summarizes this dissertation and address future research plans.

Chapter 7

Conclusion and Future Work

This chapter summarizes the proposed schemes and discusses future works for each of the proposed methods in this dissertation.

7.1 Conclusion

We have studied the vulnerabilities and threats, as well as addressed a practical countermeasures in BitTorrent systems. The proposed schemes improve system availability and informational integrity in a cost-effective and decentralized manner, focusing on Sybil attacks, freeriding, and cheating attacks. In particular, this dissertation is composed of four main parts: Part I (Chapter 3) surveyed the research problems and classified existing models; Part II (Chapter 4) addressed a simple reputation method (GOLF) and locality filtering to mitigate the impacts of Sybil attacks; Part III (Chapter 5) proposed an adaptive neighbor management method, called CoVoD, to defeat uncooperative peers in a VoD streaming system; and Part IV (Chapter 6) focused on a defense method against cheating attacks in a mesh based streaming system.

Chapter 3 investigated malicious behaviors, Sybil attacks (whitewashing), freeriding, and cheating, and classified countermeasures on P2P systems. First, there are two main categories of defense methods against Sybil attacks. Centralized approaches use public key infrastructures or monetary based systems to prevent false identities. On the other hand, decentralized approaches use the characteristics of Sybil nodes, such as limited resources, excessive attack links, and reputation. Although decentralized methods are unable to prevent Sybil attacks, they can mitigate the impact, and support P2P features (e.g., scalability and decentralization). Second, selfish behaviors are a unique problem in which freeriders do not contribute resources to the system capacity. To tackle this problem, anti-freeriding methods can be divided into direct reciprocity, reputation schemes, currency-based schemes, and piece modification schemes. Third, equivocation in distributed systems is a well-known Byzantine general problem. Although classic Byzantine solutions completely achieve consensus among the participants by state

replication with causality or a majority rule, they do not fit P2P systems. Alternatively, practical methods with trusted components or accountability use cryptographic mechanisms for authentication and messages integrity.

Chapter 4 focused on defense methods against threats from Sybil attackers in BitTorrent. A simple, direct reputation scheme (*GOLF*) fosters peer cooperation to exclude potential attackers. Locality filtering tentatively identifies Sybil nodes based on patterns in IP addresses. Under this proposed scheme, Sybil attackers may still continue to perform malicious behaviors, but their effect sharply decreases. Experimental results showed that compliant peers with the proposed scheme decrease the downloading of fake blocks to almost zero. A seeder with locality filtering also cuts down bandwidth consumption by more than 10 times in the presence of Sybil attackers.

Chapter 5 addressed a cooperative P2P VoD streaming method, called *CoVoD*, to tackle freeriding and unidirectional data flow by asymmetric interests. It achieves cooperation through adaptive neighbor management, based on referrals among peers with asymmetric interests. CoVoD helps peers find potential compatible neighbors adaptively, encourage cooperation, and penalize selfish peers. Experimental results showed that CoVoD achieves 95% or better on-time delivery of video to compliant peers when 20% of the participants are selfish peers. In addition, CoVoD achieves high utilization and fairness simultaneously, for any percentage of freeriders.

Chapter 6 proposed a practical defense mechanism, called *AntiLiar*, against cheating attacks in mesh based streaming systems. AntiLiar uses a tamper-evident progress log that consists of commitments and one-time signatures for authentication and message integrity. Peers trace their neighbors' state, and validate their commitment and signature released. They can detect a manipulated report and a selective omission through a verifiable evidence sent from a one-hop neighbor. Block filtering prevents an adverse effect from fake block attack. Experimental results showed that compliant peers with Antiliar experience 95% seamless video service, when 10% liars exploit whitewashing and collusion.

7.2 Future Work

Although this dissertation improves availability and integrity against malicious behaviors in BitTorrent systems, some problems have yet to be fully addressed. These problems still need further investigation.

- ***Defending against Sybil attacks (GOLF with locality filtering).***
  Liars (making false accusations) and traitors (engaging in productive exchanges before providing false information) can compromise trust and suspicion values in GOLF. Further investigation is required to adjust the computation of trust to reduce the effects of such attackers. We believe AntiLiar can be adapted to reduce such cheating attacks.

  Locality filtering set a threshold (i.e., 5 within an IP/24 range) based on a measurement study for RedHat9. The number of peers in an IP/24 range may depends on a content file due to users'

interests. Such a threshold may derive false positives and false negatives for detection of Sybil attackers, as mentioned in Chapter 4.6.2. Future work will include further investigation for setting a threshold and improving filter accuracy.

- ***Defending against freeriders (CoVoD).***
  CoVoD focuses on a video content against uncooperative peers. In a video streaming service, users often jump forward and backward. We believe that CoVoD will adaptively support these features. In addition, CoVoD may widely extend a file sharing system for both discovering compatible peers and punishing freeriders. Such additional experiments are included in our future direction.

  CoVoD is vulnerable to reporting a false referral. AntiLiar may help CoVoD prevent this problem. Additional future work will include combining these two methods for integrity and availability.

- ***Defending against liars (AntiLiar).***
  AntiLiar mainly addresses empirical results in mesh based streaming. Finding a theoretical bound for Byzantine fault tolerance is difficult due to scalability and dynamic features in P2P systems. Future work of AntiLiar includes a theoretical analysis.

  As shown by Chapter 6.5.2, whitewashing significantly degrades quality of service when compared with a static liar. This issue will also be improved with further research.

BIBLIOGRAPHY

[1] "The bittorrent protocol specification." [Online]. Available: http://wiki.theory.org/BitTorrentSpecification

[2] H. Schulze and K. Mochalski, "Ipoque. internet study 2008/2009." [Online]. Available: http://www.ipoque.com/study/ipoque-Internet-Study-08-09.pdf

[3] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *HotNets'06*, November 2006.

[4] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, "Free-riding in bittorrent networks with the large view exploit," in *Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS 07)*, 2007.

[5] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bit torrent," in *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, ser. NSDI'07. Berkeley, CA, USA: USENIX Association, 2007.

[6] M. Haridasan and R. van Renesse, "Securestream: An intrusion-tolerant protocol for live-streaming dissemination," *Comput. Commun.*, vol. 31, pp. 563–575, February 2008.

[7] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "Bar gossip," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 191–204.

[8] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "Trinc: small trusted hardware for large distributed systems," in *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2009, pp. 1–14.

[9] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 243–254, 2008.

[10] P. Dhungel, D. Wu, and K. W. Ross, "Measurement and mitigation of bittorrent leecher attacks," *Computer Communication*, vol. 32, no. 17, pp. 1852–1861, 2009.

[11] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena, "The pollution attack in p2p live video streaming: measurement results and defenses," in *P2P-TV '07: Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*. New York, NY, USA: ACM, 2007.

[12] P. Dhungel, X. Hei, D. Wu, and K. W. Ross, "A measurement studay of attacks on bittorrent seeds," in *Proceedings of IEEE International Conference on Communications, ICC 2011, Kyoto, Japan*, June 2011.

[13] K. Shin, D. S. Reeves, and I. Rhee, "Treat-before-trick: Free-riding prevention for bittorrent-like peer-to-peer networks," in *IPDPS '09: Proceedings of the 2009 IEEE International Symposium*

on Parallel&Distributed Processing. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–12.

[14] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Mis: malicious nodes identification scheme in network-coding-based peer-to-peer streaming," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 296–300.

[15] M. A. Konrath, M. P. Barcellos, and R. B. Mansilha, "Attacking a swarm with a band of liars: evaluating the impact of attacks on bittorrent," in *P2P*, ser. P2P '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37–44.

[16] J. R. Douceur, "The sybil attack," in *IPTPS '02: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 251–260.

[17] B. N. Levine, C. Shields, and N. B. Margolin, "A Survey of Solutions to the Sybil Attack," University of Massachusetts Amherst, Amherst, MA, Tech report 2006-052, October 2006.

[18] J. Newsome, E. Shi, D. X. Song, and A. Perrig, "The sybil attack in sensor networks: analysis & defenses," in *IPSN*, 2004, pp. 259–268.

[19] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta, "Limiting sybil attacks in structured p2p networks," in *Proc. INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 2596–2600.

[20] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," in *SIGCOMM*, 2006, pp. 267–278.

[21] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *IEEE Symposium on Security and Privacy*, 2008, pp. 3–17.

[22] N. Tran, B. Min, J. Li, and L. Subramanian, "Sybil-resilient online content voting," in *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2009, pp. 15–28.

[23] A. Cheng and E. Friedman, "Sybilproof reputation mechanisms," in *P2PECON '05: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*. New York, NY, USA: ACM, 2005, pp. 128–132.

[24] S. Marti and H. Garcia-Molina, "Identity crisis: Anonymity vs. reputation in p2p systems," in *Peer-to-Peer Computing*, 2003, pp. 134–141.

[25] H. Yu, C. Shi, M. Kaminsky, P. B. Gibbons, and F. Xiao, "Dsybil: Optimal sybil-resistance for recommendation systems," in *IEEE Symposium on Security and Privacy*, 2009, pp. 283–298.

[26] M. Zghaibeh and F. C. Harmantzis, "Revisiting free riding and the tit-for-tat in bittorrent: A measurement study," *Peer-to-Peer Networking and Applications*, vol. 1, no. 2, pp. 162–173, 2008.

[27] M. Zghaibeh, K. G. Anagnostakis, and F. C. Harmantzis, "The behavior of free riders in bit torrent networks," in *Handbook of Peer-to-Peer Networking*. Springer US, 2010, pp. 1207–1230.

[28] D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: The bell tolls?" *IEEE Distributed Systems Online*, vol. 6, June 2005.

[29] B. Cohen, "Incentives build robustness in bittorrent," *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[30] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *Proceedings of the 5th ACM conference on Electronic commerce*, ser. EC '04. New York, NY, USA: ACM, 2004, pp. 102–111.

[31] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1 –12.

[32] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA: ACM, 2003, pp. 640–651.

[33] R. Gupta and A. K. Somani, "Reputation management framework and its use as currency in large-scale peer-to-peer networks," in *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, ser. P2P '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 124–132.

[34] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, "One hop reputations for peer to peer file sharing workloads," in *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14.

[35] F. D. Garcia and J. henk Hoepman, "Off-line karma: A decentralized currency for peer-to-peer and grid applications," in *In 3rd Applied Cryptography and Network Security conference (ACNS)*. Springer, 2005, pp. 364–377.

[36] A. Ramach, A. D. Sarma, and N. Feamster, "Bitstore: An incentive-compatible solution for blocked downloads in bittorrent," in *Proc. of Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing (NetEcon+IBC)*, San Diego, CA, USA, 2007.

[37] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki, "Dandelion: Cooperative content distribution with robust incentives," in *USENIX Annual Technical Conference*, 2007, pp. 157–170.

[38] T. Locher, S. Schmid, and R. Wattenhofer, "Rescuing tit-for-tat with source coding," in *Peer-to-Peer Computing, 2007. P2P 2007. Seventh IEEE International Conference on*, Sept. 2007, pp. 3 –10.

[39] D. Wu, P. Dhungel, X. Hei, C. Zhang, and K. W. Ross, "Understanding peer exchange in bittorrent systems," in *Peer-to-Peer Computing*, 2010, pp. 1–8.

[40] A. Z. Broder and M. Mitzenmacher, "Survey: Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, 2003.

[41] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, "Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, March 2005, pp. 2102–2111 vol. 3.

[42] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe, "Contracts: practical contribution incentives for p2p live streaming," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 81–94.

[43] W. Liang, J. Bi, R. Wu, Z. Li, and C. Li, "On characterizing ppstream: Measurement and analysis of p2p iptv under large-scale broadcasting," in *Global Telecommunications Conference*, 2009, pp. 1–6.

[44] B. Fallica, Y. Lu, F. Kuipers, R. Kooij, and P. V. Mieghem, "On the quality of experience of sopcast," in *Proceedings of the 2008 The Second International Conference on Next Generation Mobile Applications, Services, and Technologies*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 501–506.

[45] G. Halkes and J. Pouwelse, "Udp nat and firewall puncturing in the wild," in *NETWORKING 2011*, ser. Lecture Notes in Computer Science, J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, Eds. Springer Berlin / Heidelberg, 2011, vol. 6641, pp. 1–12.

[46] J. K. So and D. Reeves, "Defending against sybil nodes in bittorrent," in *NETWORKING 2011*, ser. Lecture Notes in Computer Science, J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, Eds. Springer Berlin / Heidelberg, 2011, vol. 6641, pp. 25–39.

[47] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *ESA'06: Proceedings of the 14th conference on Annual European Symposium*. London, UK: Springer-Verlag, 2006, pp. 684–695.

[48] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proceedings of CRYPTO 84 on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 47–53.

[49] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "Karma: A secure economic framework for p2p resource sharing," *IEEE/ACM Transactions on Networking*, 2003.

[50] M. J. Freedman and R. Morris, "Tarzan: a peer-to-peer anonymizing network layer," in *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2002, pp. 193–206.

[51] K. Walsh and E. G. Sirer, "Experience with an object reputation system for peer-to-peer filesharing," in *Proceedings of Networked System Design and Implementation (NSDI)*. Networked System Design and Implementation (NSDI), May 2006.

[52] C. Costa, V. Soares, J. Almeida, and V. Almeida, "Fighting pollution dissemination in peer-to-peer networks," in *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2007, pp. 1586–1590.

[53] R. Landa, D. Griffin, R. G. Clegg, E. Mykoniati, and M. Rio, “A sybilproof indirect reciprocity mechanism for peer-to-peer networks,” in *Proc. IEEE INFOCOM 2009*, 2009, pp. 343–351.

[54] Q. Lian, Y. Peng, M. Yang, Z. Zhang, Y. Dai, and X. Li, “Robust incentives via multi-level tit-for-tat: Research articles,” *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 2, pp. 167–178, 2008.

[55] S. B. Handurukande, A.-M. Kermarrec, F. Le Fessant, L. Massoulié, and S. Patarin, “Peer sharing behaviour in the edonkey network, and implications for the design of server-less file sharing systems,” *SIGOPS Oper. Syst. Rev.*, vol. 40, pp. 359–371, April 2006. [Online]. Available: http://doi.acm.org/10.1145/1218063.1217970

[56] H. T. Kung and C.-H. Wu, “Differentiated admission for peer-to-peer systems: Incentivizing peers to contribute their resources,” in *P2PECON’03*, June 2003.

[57] J. Mol, D. Epema, and H. Sips, “The orchard algorithm: Building mulicast trees for p2p video multicasting without free-riding,” *IEEE Transactions on Multimedia*, vol. 9, no. 8, pp. 1593–1604, December 2007.

[58] J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, and H. Sips, “Give-to-get: Free-riding-resilient video-on-demand in p2p systems,” in *Multimedia Computing and Networking 2008*, vol. 6818, no. Article 681804. SPIE Vol. 6818, January 2008.

[59] S. Jun and M. Ahamad, “Incentives in bittorrent induce free riding,” in *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, ser. P2PECON ’05. New York, NY, USA: ACM, 2005, pp. 116–121. [Online]. Available: http://doi.acm.org/10.1145/1080192.1080199

[60] L. Lamport, R. E. Shostak, and M. C. Pease, “The byzantine generals problem,” *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[61] L. Lamport, “The weak byzantine generals problem,” *J. ACM*, vol. 30, no. 3, pp. 668–676, 1983.

[62] M. Barborak, A. Dahbura, and M. Malek, “The consensus problem in fault-tolerant computing,” *ACM Comput. Surv.*, vol. 25, pp. 171–220, June 1993.

[63] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *OSDI*, 1999, pp. 173–186.

[64] ——, “Practical byzantine fault tolerance and proactive recovery,” *ACM Trans. Comput. Syst.*, vol. 20, pp. 398–461, November 2002.

[65] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, “Separating agreement from execution for byzantine fault tolerant services,” *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 253–267, October 2003.

[66] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie, “Fault-scalable byzantine fault-tolerant services,” *SIGOPS Oper. Syst. Rev.*, vol. 39, pp. 59–74, October 2005.

[67] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira, “Hq replication: A hybrid quorum protocol for byzantine fault tolerance,” in *Proceedings of the Seventh Symposium on Operating Systems Design and Implementations (OSDI)*, Seattle, Washington, Nov. 2006.

[68] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 45–58, 2007.

[69] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested append-only memory: making adversaries stick to their word," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 189–204, 2007.

[70] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: practical accountability for distributed systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 175–188, 2007.

[71] C. Ho, R. van Renesse, M. Bickford, and D. Dolev, "Nysiad: practical protocol transformation to tolerate byzantine failures," in *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2008, pp. 175–188.

[72] R. Jordan, "Security chip that does encryption in pcs hacked; criminals can crack encryption with new attack." [Online]. Available: http://abcnews.go.com/Technology/wireStory?id=9780148

[73] "iphone, ipad tracking raises questions about privacy responsibility of smartphone makers." [Online]. Available: http://article.wn.com/view/2011/04/21/iPhone_iPad_tracking_raises_questions_about_privacy_responsi/

[74] M. Young, A. Kate, I. Goldberg, and M. Karsten, "Practical robust communication in dhts tolerating a byzantine adversary," in *ICDCS*, 2010, pp. 263–272.

[75] J. Liang, N. Naoumov, and K. W. Ross, "Efficient blacklisting and pollution-level estimation in p2p file-sharing systems," in *The Asian Internet Engineering Conference*, 2005, pp. 1–21.

[76] J. Sun, A. Banerjee, and M. Faloutsos, "Multiple identities in bittorrent networks," in *Proceedings of the 6th international IFIP-TC6 conference on Ad Hoc and sensor networks, wireless networks, next generation internet*, ser. NETWORKING'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 582–593.

[77] "Safepeer." [Online]. Available: http://wiki.vuze.com/w/SafePeer

[78] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2006, pp. 203–216.

[79] "Adsl." [Online]. Available: http://en.wikipedia.org/wiki/Asymmetric_digital_subscriber_line

[80] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in bittorrent systems," in *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2007, pp. 301–312.

[81] "Redhat 9 torrent tracker trace." [Online]. Available: http://mikel.tlm.unavarra.es/~mikel/bt_pam2004/

[82] "Tor: anonymity online." [Online]. Available: http://www.torproject.org/

[83] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2007, pp. 133–144.

[84] Y. Yang, A. L. H.Chow, L. Golubchik, and D. Bragg, "Improving qos in bittorrent-like vod systems," in *INFOCOM'10: Proceedings of the 29th conference on Information communications*. Piscataway, NJ, USA: IEEE Press, 2010, pp. 2061–2069.

[85] C. Liang, Z. Fu, Y. Liu, and C. W. Wu, "Incentivized peer-assisted streaming for on-demand services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 1354–1367, 2010.

[86] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "Bitos: Enhancing bittorrent for supporting streaming applications," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–6.

[87] S. J. Nielson and S. A. Crosby, "A taxonomy of rational attacks," in *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS 05)*. Springer-Verlag, 2005, pp. 36–46.

[88] Y.-h. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast (keynote address)," in *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2000, pp. 1–12.

[89] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: high-bandwidth multicast in cooperative environments," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 298–313.

[90] F. Chen, T. Repantis, and V. Kalogeraki, "Coordinated media streaming and transcoding in peer-to-peer systems," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*. Washington, DC, USA: IEEE Computer Society, 2005, p. 56.2.

[91] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast," in *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 2–11.

[92] X. Lan, N. Zheng, J. Xue, W. Chen, B. Wang, and W. Ma, "Manageable peer-to-peer architecture for video-on-demand," in *IPDPS*, 2008, pp. 1–7.

[93] X. Yang, P. Hernández, F. Cores, L. Souza, A. Ripoll, R. Suppi, and E. Luque, "Dvodp2p: distributed p2p assisted multicast vod architecture," in *IPDPS*, 2006.

[94] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," in *INFOCOM*, 2007, pp. 1424–1432.

[95] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai, "Improving vod server efficiency with bittorrent," in *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*. New York, NY, USA: ACM, 2007, pp. 117–126.

[96] Y. Feng, B. Li, and B. Li, "Peer-assisted vod prefetching in double auction markets," *ICNP*, pp. 275–284, 2010.

[97] "Distibuted hash tables." [Online]. Available: http://en.wikipedia.org/wiki/Distributed_hash_table

[98] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK: Springer-Verlag, 2002, pp. 53–65.

[99] J. Liang, N. Naoumov, and K. W. Ross, "The index poisoning attack in p2p file sharing systems," in *IEEE INFOCOM'06*, April 2006.

[100] H. Wang, J. Liu, and K. Xu, "On the locality of bittorrent-based video file swarming," in *Proceedings of the 8th international conference on Peer-to-peer systems*, ser. IPTPS'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 12–12.

[101] B. Fan, D. ming Chiu, and J. Lui, "The delicate tradeoffs in bittorrent-like file sharing protocol design," *Network Protocols, IEEE International Conference on*, vol. 0, pp. 239–248, 2006.

[102] "G2g simulator." [Online]. Available: http://svn.tribler.org/abc/branches/jandavid/g2gsim/

[103] J. K. So and D. Reeves, "Adaptive neighbor management for cooperative p2p video-on-demand streaming," in *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, nov. 2011, pp. 1 –10.

[104] T. Silverston, L. Jakab, A. Cabellos-Aparicio, O. Fourmaux, K. Salamatian, and K. Cho, "Large-scale measurement experiments of p2p-tv systems: Insights on fairness and locality," *in Elsevier Signal Processing: Image Communication*, 2011.

[105] E. Alessandria, M. Gallo, E. Leonardi, M. Mellia, and M. Meo, "P2p-tv systems under adverse network conditions: A measurement study," in *INFOCOM*, 2009, pp. 100–108.

[106] D. Frey, R. Guerraoui, A.-M. Kermarrec, and M. Monod, "Boosting gossip for live streaming," in *P2P 2010*, Aug. 2010, pp. 1 –10.

[107] J. Seibert, X. Sun, C. Nita-Rotaru, and S. Rao, "A design for securing data delivery in mesh-based peer-to-peer streaming," *Comput. Netw.*, vol. 55, no. 12, pp. 2730–2745, Aug. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2011.05.016

[108] A. Singh, M. Castro, P. Druschel, and A. Rowstron, "Defending against eclipse attacks on overlay networks," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, ser. EW 11. New York, NY, USA: ACM, 2004.

[109] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and whitewashing in peer-to-peer systems," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 5, pp. 1010 – 1019, may 2006.

[110] "Peerreview source code." [Online]. Available: http://peerreview.mpi-sws.org/index.html#downloads

[111] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, “A measurement study of a large-scale p2p iptv system,” *IEEE Transactions on Multimedia*, vol. 9, 2007.

[112] “Openssl.” [Online]. Available: http://www.openssl.org