TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science, Chair of Privacy and Data Security

Contributions to the Resilience of Peer-To-Peer Video Streaming against Denial-of-Service Attacks

Dissertation

Submitted to the Faculty of Computer Science, TU Dresden,
in Partial Fulfillment of the Requirements for the Degree of Dr.-Ing.

by

M.Eng. **Giang T. Nguyen**

born on 09 September 1978 in Hanoi, Vietnam

Committee members:

Prof. Dr.-Ing. Thorsten Strufe	TU Dresden, Germany
Prof. Dr. Jussi Kangasharju	University of Helsinki, Finland
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill	TU Dresden, Germany
Prof. Dr. rer. nat. habil. Gerhard Weber	TU Dresden, Germany
Prof. Dr.-Ing. Jeronimo Castrillon	TU Dresden, Germany

Date of defense: 02 March 2016

Dresden, March 2016

Zusammenfassung

Um die ständig wachsenden Anforderungen zur Übertragung von Live Video Streams im Internet zu erfüllen werden kosteneffektive und resourceneffiziente Lösungen benötigt. Eine adäquate Lösung bietet die Peer-to-Peer (P2P) Streaming Architektur an, welche bereits heute in unterschiedlichsten Systemen zum Einsatz kommt. Solche Systeme erfordern von der Streaming Quelle nur moderate Bandbreiten, da die Nutzer (bzw. Peers) ihre eigene Bandbreite zur Verbreitung des Streams einbringen. Dazu werden die Peers oberhalb der Internetarchitektur zu einem Overlay verbunden. Das geplante Verlassen, sowie der ungewollte Absturz von Peers (genannt Churn) kann das Overlay schädigen und den Empfang einiger Peers unterbrechen. Weitaus kritischer sind Angriffe auf die Verfügbarkeit des Systems indem relevante Knoten des Overlays von Angreifern attackiert werden, um die Verteilung des Streams gezielt zu stören.

Um Overlays zu konstruieren, die robust gegenüber Churn sind, nutzen so genannte pull-basierte P2P Streaming Systeme eine Mesh Topologie um jeden Peer über mehrere Pfade mit der Quelle zu verbinden. Peers fordern regelmäßig Teile des Videos, sog. Chunks, von ihren Partnern im Overlay an. Selbst wenn einige Partner plötzlich nicht mehr im System verfügbar sind kann ein Peer alle Chunks von den verbleibenden Nachbarn beziehen. Um dies zu ermöglichen tauschen Peers regelmäßig sog. Buffer Maps aus. Diese kleinen Pakete enthalten Informationen über die Verfügbarkeit von Chunks im Puffer eines Peers. Um dadurch entstehende Latenzen und den zusätzlichen Mehraufwand zu reduzieren wurden hybride Systeme entwickelt. Ein solches System beginnt pull-basiert und formt mit der Zeit einen Baum aus einer kleinen Untermenge aller Peers um Chunks ohne explizite Anfrage weiterzuleiten. Unglücklicherweise sind sowohl pull-basierte, als auch hybride Systeme anfällig gegenüber Denial-of-Service Angriffen (DoS). Insbesondere fehlen Maßnahmen zur Abschwächung von DoS Angriffen auf die Partner der Quelle. Die genannten Angriffe werden weiterhin dadurch erleichtert, dass die Identität der Quelle-nahen Knoten akkurat aus den ausgetauschten Buffer Maps extrahiert werden kann. Hybride Systeme sind außerdem anfällig für Angriffe auf den zugrundeliegenden Baum.

Aufgrund der schwerwiegenden Auswirkungen von DoS Angriffen auf pull-basierte, sowie hybride Systeme stellen wir drei Gegenmaßnahmen vor. Zuerst entwickeln wir das *Striping* Schema zur Abschwächung von DoS Angriffen auf die Partner der

Quelle. Hierbei werden Peers dazu angeregt ihre Chunk-Anfragen an unterschiedliche Partner zu senden. Als zweites entwickeln wir das *SWAP* Schema, welches Peers dazu bringt proaktiv ihre Partner zu wechseln um Angreifer daran zu hindern die Quellenahe zu identifizieren. Als drittes entwickeln wir *RBCS*, einen widerstandsfähigen Baum zur Abschwächung von DoS Angriffen auf hybride Systeme.

Da bisher kein Simulator für die faire Evaluation von P2P-basierten Live Video Streaming Algorithmen verfügbar war, entwickeln wir *OSSim*, ein generalisiertes Simulations-Framework für P2P-basiertes Video Streaming. Des weiteren entwickeln wir etliche Angreifermodelle sowie neuartige Resilienzmetriken on OSSim. Ausgiebige Simulationsstudien zeigen, dass die entwickelten Schemata signifikant die Widerstandsfähigkeit von pull-basierten und hybriden Systemen gegenüber Churn und DoS Angriffen erhöhen.

Abstract

The constantly growing demand to watch live videos over the Internet requires streaming systems to be cost-effective and resource-efficient. The Peer-to-Peer (P2P) streaming architecture has been a viable solution with various deployed systems to date. The system only requires a modest amount of bandwidth from the streaming source, since users (or peers) contribute their bandwidth to disseminate video streams. To enable this, the system interconnects peers into an overlay. However, churn–meaning the leaving and failing of peers–can break the overlay, making peers unable to receive the stream. More severely, an adversary aiming to sabotage the system can attack relevant nodes on the overlay, disrupting the stream delivery.

To construct an overlay robust to churn, pull-based P2P streaming systems use a mesh topology to provide each peer with multiple paths to the source. Peers regularly request video chunks from their partners in the overlay. Therefore, even if some partners are suddenly absent, due to churn, a peer still can request chunks from its remaining partners. To enable this, peers periodically exchange buffer maps, small packets containing the availability information of peers' video buffers. To reduce latency and overhead caused by the periodic buffer map exchange and chunk requests, hybrid systems have been proposed. A hybrid system bootstraps from a pull-based one and gradually forms a tree backbone consisting of a small subset of peers to deliver chunks without requests. Unfortunately, both pull-based and hybrid systems lack measures to mitigate Denial-of-Service (DoS) attacks on head nodes (or the source's partners). More critically, they can be identified accurately by inferring exchanged buffer maps. Furthermore, hybrid systems are vulnerable to DoS attacks on their backbones.

Since DoS attacks can badly affect both pull-based and hybrid systems, we introduce three countermeasures. First, we develop the *striping* scheme to mitigate DoS attacks targeting head nodes. The scheme enforces peers to diversify their chunk requests. Second, to prevent attackers from identifying head nodes, we develop the *SWAP* scheme, which enforces peers to proactively change their partners. Third, we develop *RBCS*, a resilient backbone, to mitigate DoS attacks on hybrid systems.

Since a simulator for a fair evaluation is unavailable so far, we develop *OSSim*, a general-purpose simulation framework for P2P video streaming. Furthermore, we develop several attacker models and novel resilience metrics in OSSim. Extensive simulation studies show that the developed schemes significantly improve the resilient of pull-based and hybrid systems to both churn and DoS attacks.

Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Dresden, den 18.01.2016

_____

(Giang T. Nguyen)

Acknowledgements

When I started working on this PhD project, I had only a vague idea of where I was heading to. Looking back, I am very glad how much I have learned, and how significant steps the PhD project went through over the years. This would have never been possible without all the people who accompanied me or who I met during this journey. Here, I want to express my gratitude to them.

First and foremost, my advisor Professor Dr. Thorsten Strufe, for his consistent support on me and my work. His advice, guidance, patience, and inspiration were essential to this thesis and the progress in my journey to learn how to do research.

Next, my coauthors (Stefanie, Benjamin, Mathias), for the insightful discussions, the invaluable comments and feedback, improving the publication's quality; my other colleagues at the P2P Networks group (TU Darmstadt) and Chair of Privacy and Data Security (TU Dresden), especially Stefanie, Benjamin, Hani, Martin, and Stefan for proofreading my dissertation and translating the abstract into German.

Three organizations, for their financial support for my research project. They are the German Academic Exchange Service (DAAD), the Chair for Privacy and Data Security and the Graduate Academy at TU Dresden.

Staff at the Vietnamese-German Center, especially Mr. Viet Duc; my colleagues at the University of Civil Engineering in Hanoi, for all their support and care.

Professors in Vietnam, for recommending me and connecting me with Germany so that I had a chance to start my journey. They are Prof. Hoang Dang Hai, Dr. Pham Thieu Nga and Prof. Pham Minh Ha. I also would be thankful for their experiences and valuable discussions with them about studying in Germany.

Friends of our family in Darmstadt and Dresden, especially the WSD group, for being with us in difficult time and helping us settle our lives in new cities.

My parents, for making me who I am today and shaping my perspective about the world; my parents in law, my brother and sister in law, for being with me and my small family during highs and lows. These sources of support help me a lot to concentrate on the research work and to complete the last miles in my journey.

Last but not least, my wife Trang and my two kids Khanh Linh and Duy Duc, for always being the love of my life and for sharing our lives in both difficult time and joyful moments, for their understanding when I was away completing my thesis.

Contents

1

Introduction

Due to the growing user demand to watch live events, such as music festivals, football matches, and TV news etc., video streaming over the Internet has become a popular service. Video streaming offers the capability to start playing a video as soon as a small portion of the video is successfully downloaded. Thus, the waiting time is reduced. Statistics show that the traffic from video streaming has increases drastically and this trend is projected to continue in the coming years [21]. To satisfy this demand, several architectures have been proposed for video streaming over the Internet: unicast, IP multicast, Content Delivery Network (CDN) and Peer-to-Peer (P2P) as illustrated in Figure 1.1.

In a straightforward approach, the unicast architecture [6] shown in Figure 1.1(a) requires the streaming server to deliver the video directly to its users. To do that, every user has to establish a unicast connection to the server. As long as the server still has enough resources, mainly its upload bandwidth, every newly arriving user can receive the video stream with a low latency. However, the bandwidth demand to the server increases linearly with the number of users. This generates a huge load not only on the streaming server but also on the Internet's core network. The network has to simultaneously deliver several times of the same stream to users.

To resolve the bandwidth demand to both the streaming server and the core network, the IP multicast architecture [24, 75] is developed. As can be seen from Figure 1.1(b), the server sends only one stream on the shared paths of the connections between the server and its users. The stream is then replicated at the routers where the paths to different users diverge. Therefore, the video traffic duplication is minimized. Additionally, the delay is also minimized since the video stream is replicated very early at the IP layer without being passed through upper ones. In spite of those huge advantages, IP multicast has two major drawbacks. First, it

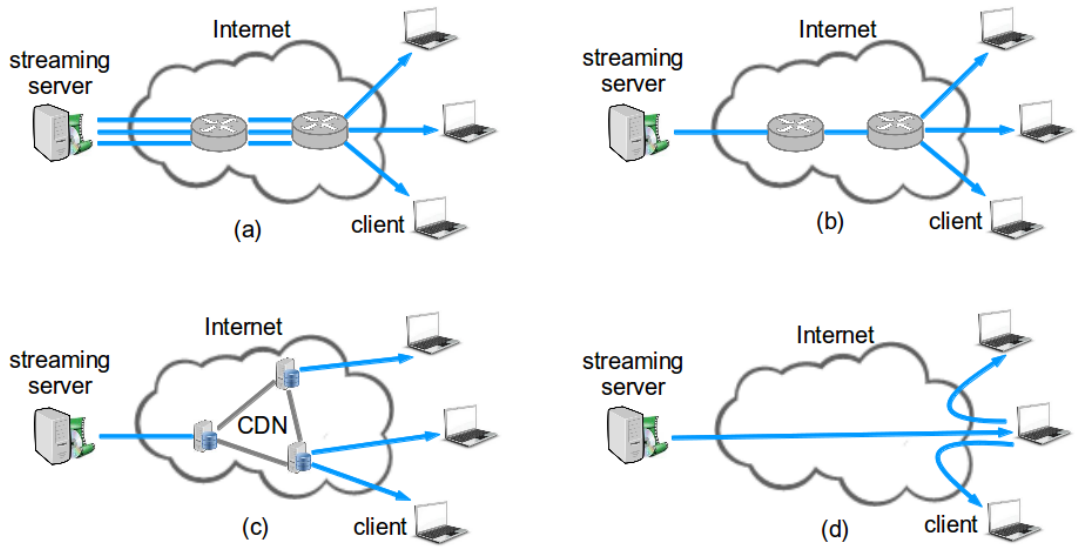


FIGURE 1.1: Four architectures of video streaming over the Internet: (a) Unicast; (b) IP multicast; (c) Content Delivery Network (CDN); and (d) Peer-to-Peer (P2P)

requires routers to maintain a state for each multicast group, which complicates the management and limit the scaling at the IP layer. Second, the architecture requires modification at the infrastructure level, thus slowing down the deployment [25]. So far, the IP multicast architecture only appears in restricted deployments (e.g., at IPTV backbones) [46].

The Content Delivery Network (CDN) architecture, illustrated in Figure 1.1(c), addresses the deployment problem of IP multicast by introducing a dedicated network consisting of several replication servers [60]. They are equipped with abundant bandwidth and installed strategically in high traffic regions. Therefore, the video source needs to send the stream to the CDN only once. The stream is then distributed within the CDN to its replication servers and duplicated there to serve nearby users via unicast connections. In addition to a drastic reduction of the bandwidth demand to the streaming server, CDN offers several advantages, such as high capacity and high performance. However, CDN also has several drawbacks, which are high costs and the poor support to live streaming.

The Peer-to-Peer (P2P) architecture, illustrated in Figure 1.1(d), resolves the high-cost problem of a CDN by requiring users (or peers) to contribute their resource, mainly the upload bandwidth, to disseminate video streams. Participating peers not only receive but also send video streams to others. Since P2P streaming only requires a modest upload bandwidth from the streaming server (or source), it reduces the bandwidth demand on the source. Additionally, the architecture also enables scaling up the system since peers participate with their own resources. The more peers join the system, the more resources the system has. Despite the above advantages,

the P2P architecture poses higher latency than its alternatives since a peer generally receives the stream after it is forwarded via several other peers. The general challenge for P2P streaming is building an overlay interconnecting the source and peers and deciding a method to disseminate video streams so that the overall system maintains a low latency and a low signaling overhead. Since ordinary peers can join and leave the system freely at any time, churn is inevitable. Building an overlay robust to churn is a must to maintain the service without disruptions. Furthermore, the system has to provide attack resistance. To damage the system, a relevant subset of peers can be shut down to disrupt the video dissemination to the rest of the peers. Such sabotage can be performed by an individual hacker simply for fame or by an organization or a government to block the delivery of a certain content. Over the past years, various P2P streaming systems have been proposed. They can be categorized into three classes: push-based, pull-based and hybrid ones as shown in Figure 1.2.

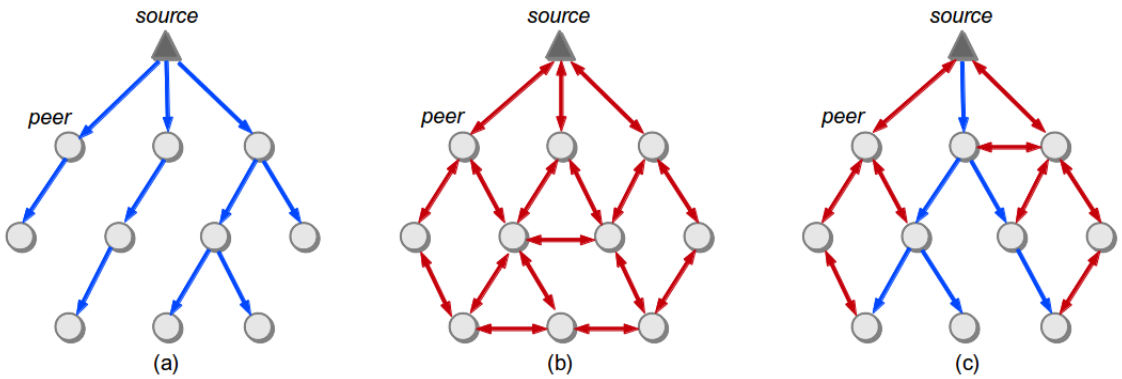


FIGURE 1.2: Three classes of P2P streaming systems: (a) push-based; (b) pull-based; and (c) hybrid push-pull. The links between nodes depict the overlay connections while the arrowheads indicate the directions of sent video chunks

As illustrated in Figure 1.2(a), an early push-based system builds a tree rooting at the source and spanning all the peers. Along the tree's branches, the video chunks of the stream are sent (or pushed) from the source and immediately forwarded by intermediate peers until they reach peers at the leaves of the tree. This way, the system achieves a low latency. The single-tree topology, however, is fragile to high churn and especially to attacks on peers close to the source since each peer solely depends on its predecessors. To improve the robustness to churn, later push-based systems maintain multiple-tree topologies providing each peer several direct predecessors to improve its connectivity. Moreover, each tree disseminates video chunks of a partial stream. Even when a predecessor leaves, its successors only lose a partial stream and still receive the rest from their remaining predecessors. Furthermore, some of the multiple-tree push-based systems are even theoretically resistant to a perfect attacker [14]. However, those superior properties have not been proven in large-scale real-world deployments.

Also tackling the issue of churn, pull-based systems follow a very different approach, which is inspired by the success of the famous P2P file sharing application BitTorrent [22]. A pull-based system strengthens the system's connectivity by constructing a mesh topology interconnecting the source and its peers with one another as depicted in Figure 1.2(b). Thus, there are several source-to-peer paths for each peer preventing that the departure of a single node leads to the node's isolation and hence its inability to receive the stream. To obtain the video stream, every peer has to explicitly and regularly request video chunks from others. This reactive behavior also helps peers adapt quickly to their neighboring peers' churn. However, the trade-off is that peers and the source have to periodically send buffer maps (BMs), small packets carrying the availability information of peers' video buffers. The combination of frequently exchanging BMs and explicit requests causes high latency and high signaling overhead. Nevertheless, the pull-based class of systems has been adopted widely in many real-world P2P streaming systems [31], such as PPLive [51] and Sopcast [56].

To combine the advantages of both push-based and pull-based systems, a hybrid push-pull (or hybrid) system bootstraps from a pull-based one to leverage its robustness to churn [68]. Subsequently, a subset of peers starts to push video chunks directly to their neighboring peers without requests to reduce the latency and overhead. The process of pushing video chunks begins from the source and gradually to other peers to form a backbone. To strengthen the backbone, it should only consist of stable peers meaning that they already stay sufficiently long in the system. A measurement study shows that such a backbone delivers a majority of video chunks in the system [68]. At the same time, peers in the backbone still keep their existing mesh connections with neighboring peers for two reasons. First, backbone peers can send requested chunks to other peers to reduce the overall latency. Second, when a backbone peer loses the pushed stream from a predecessor due to churn, the peer still can request missing chunks from neighboring peers, which increases the robustness. The topology of a hybrid system is illustrated in Figure 1.2(c).

Despite the robustness to churn, pull-based systems and their derivatives, hybrid ones, can be vulnerable to Denial-of-Service (DoS) attacks on head nodes, which are the source's partners. Those nodes can be identified by inferring the information in exchanged buffer map. The inference of the overlay structure including head nodes from collected buffer maps has been experimented on PPLive, a pull-based system [33]. It is however unknown whether the inference of head nodes is feasible on pull-based systems in general. However, if an inference attacker can successfully infer the head nodes and subsequently shut them down, the remaining peers are disconnected from the source, thus, disrupting the video stream delivery. Hybrid systems, bootstrapping from a pull-based one are also vulnerable to the inference and therefore are vulnerable to the attacks. Furthermore, the backbone topology of a hybrid system is a single-tree that is fragile to attacks. Disrupting this backbone can drastically affect not only backbone peers but also the video stream delivery to

the rest of the peers. So far, these problems have not been addressed.

In summary, pull-based systems with their proven robustness to churn are potential candidates for P2P video streaming over the Internet. However, two major problems need to be solved: (*i*) the resilience of pull-based systems to attacks on head nodes and (*ii*) the resilience of hybrid push-pull system to attacks on the backbone.

Problem description

This thesis needs to improve the resilience of pull-based and subsequently hybrid P2P streaming systems to DoS attacks. For this reason, pull-based systems need to improve their resilience to attacks on head nodes. On top of that, hybrid systems need to build a resilient backbone to counter DoS attacks.

Goals of the study

This thesis sets several goals as follows:

1. A set of metrics needs to be defined to quantify the steady-state performance of P2P streaming systems in benign scenarios and especially to quantify the resilience of the systems under attacks.
2. To thoroughly investigate and fairly compare different classes of P2P streaming systems, a multi-purpose simulation framework needs to be developed.
3. To mitigate the attacks on head nodes, the thesis needs to study the impact of the attacks on pull-based P2P streaming systems and subsequently develop countermeasures against such attacks.
4. To maintain a low latency and a low signaling overhead and at the same time to mitigate the attacks on the backbone of a hybrid systems, we needs to develop a countermeasure against such attacks.

Contributions of this thesis

A general-purpose simulation framework: The evaluation of different schemes in P2P streaming requires a common framework that allows for a fair comparison. In addition, the framework should be generic and extensible to support the implementation of various P2P streaming systems. Since such a framework has not been available so far, we develop OSSim [48], our general-purpose simulation framework for P2P video streaming. The framework allows for simulating all classes of P2P streaming

systems, namely push-based, pull-based and hybrid. To validate the framework we implementations one representative system for each class of P2P streaming systems. The simulation results from our framework match closely with published results. In addition to that, we develop and implement several novel resilience metrics and attacker models, such as [27]. Consequently, we use OSSim as our main tool to evaluate the proposed countermeasures against DoS attacks in P2P streaming systems.

Resilience against DoS attacks on head nodes: Even though pull-based systems are vulnerable to DoS attacks on head nodes. Shutting down those nodes can disrupt the stream delivery from the source to the rest of the peers. We develop the striping scheme [49] to mitigate damages caused by such attacks even if the attacker has a global knowledge. Our approach is to increase the number of partners each peer has, consequently increasing the number of head nodes. Therefore, the system has more chance to maintain connections between the source and the peers under attacks, thus, reducing the negative impacts of the attacks. However, we also need to prevent peers from overloading themselves by chunk requests from their enlarged partner lists. We solve this problem in our striping scheme by enforcing peers to diversify their chunk requests. To do so, we introduce two techniques: First, the video stream is divided into k stripes, which are partial streams. Second, each peer assigns its partners into k groups. Consequently, in each scheduling cycle, every peer has to request chunks of a particular stripe from the respective group of partners only.

Resilience against inference attacks: In pull-based systems, the overlay structure, especially head node can be identified by the inferring the exchanged buffer maps. Despite mitigating the damages caused by the attacks on head nodes, the striping scheme does not prevent the identification of head nodes. For this reason, we develop SWAP as a countermeasure to the inference attacker that collects buffer maps to infer head nodes. Our idea is to enforce peers to regularly and proactively change their partners. As the result, the attacker is unable to accurately identify head nodes any more. Consequently, we introduce the following three primitives: First, to prepare for the swapping operations, every peer should suggest its replacement candidate. Furthermore, the nominations should be forwarded several times, allowing peers to connect to more diverse new partners. Second, to prevent the nomination message from infinitely forwarded, we introduce a counter. Third, to be ready for attacks, peers periodically replace a subset of their partners by the respectively nominated ones.

Resilience against DoS attacks on the backbone: After strengthening the resilience of pull-based systems to DoS attacks, we address the vulnerability of hybrid systems. Specifically, to protect hybrid systems from the attacks on the backbone, we develop the Resilient Backbone Construction Scheme (RBCS) [50]. The backbone only includes stable peers that stay sufficiently long in the system. The scheme is build from three primitives. First, peers only use their local observation to identify

stable peers that stay sufficiently long in the system. Identified stable peers are invited to join the backbone. Using invitations, the system does not reveal peers in the backbone, thus, lowers the risks of attacks on the backbone. Second, the backbone employs the Optimally Stable Topology (OST) to include peers into a multiple-tree overlay. OST has theoretically proven its resilience to DoS attacks, thus, strengthening the backbone against attacks. Third, RBCS introduces an adaptive bandwidth reservation scheme to flexibly coordinate the bandwidth of peers when they push chunks in the backbone and respond to chunk requests at the same time.

Collaborations

Even though I myself have done an original work in this thesis, the work has been resulted from collaborations with several people. They are my supervisor, colleagues at TU Darmstadt and TU Dresden and several students. My supervisor, Professor Strufe, contributed in all the collaborations with his guidance and valuable comments in problem identification, solution design, evaluation and writing. In the following, the collaborations are elaborated in detail.

The first collaborative work is conducted in Chapter 4. The main part of this chapter is OSSim, our general-purpose simulation framework for P2P video streaming. The idea to develop the framework from scratch was finalized after discussions with Professor Strufe. During the early phase of design and development, I worked closely with Dr. Fischer. Together we revised the overall design of the framework. Afterwards, I implemented the core and substantial parts of the framework. Later on OSSim was extended by two students at TU Darmstadt when they worked with me on their project and theses. They implemented three P2P streaming systems. Julian Wulfheide developed Optimally Stable Topology (OST), the multiple-tree push-based system and conducted experiments to validate his implementation as part of his Bachelor thesis. Thorsten Jacobi developed two hybrid systems, namely Coolstreaming and mTreebone during his research project and Master thesis. He later performed simulation experiments to validate his implementations.

The second collaborative work is performed in Chapter 5. Even though I identified the problem of pull-based systems when head nodes are attacked, it was the consequence of previous unsuccessful attempts. Together with Dr. Fischer, we tried various attacker models on pull-based systems. On the search for a solution, we started our collaboration since the early design stage of the striping scheme. The subsequent implementation of the striping scheme as well as the evaluation were conducted by myself. In this chapter, my colleague Stefanie Roo helped me formalize the optimal partner grouping problem through her valuable comments.

The third collaboration is done in Chapter 6 with my two colleagues Stefanie Roos and Benjamin Schiller. I posed the challenge to undermine the attacker's

capability in accurately identifying head nodes. The discussions with them helped me eliminate invalid approaches to solve the problem. Even though I proposed the idea of the SWAP scheme later, our discussions shaped the design and the development of the scheme. Especially, Stefanie Roos formalized the problem and formally derived the lower bound of the attack's accuracy. This bound was then validated through simulation results. The subsequent implementation of the SWAP scheme as well as the evaluation were conducted by myself. However, I also received valuable comments from Benjamin Schiller during this phase.

The fourth collaboration is performed in Chapter 7 with Dr. Fischer and Stefanie Roos. Dr. Fischer identified the problem of self promotion to identify stable peers, which undermined the resilience of mTreebone and proposed his idea on the invitation as the solution. Together we discussed possible strategies towards an improved hybrid scheme. Later, we focused on my idea on building a resilient backbone from the multiple-tree push-based scheme OST to form the RBCS scheme. In this chapter, Stefanie Roos formalized the system and described the attacker model and the problem formally. Afterwards, the implementation of the RBCS scheme as well as the evaluation were conducted by myself.

Structure of the thesis

Following the introduction in this chapter, Chapter 2 first provides a broad background on the topic of video streaming from various video coding techniques to metrics to measure the performance of video streaming systems. After briefly discussing the pros and cons of alternative video streaming architectures, the chapter describes the P2P streaming architecture. Three classes of P2P streaming systems are then presented, namely push-based, pull-based and hybrid push-pull. Later sections discuss common challenges in P2P streaming, which are churn (including failure) and Denial-of-Service (DoS) attacks.

Chapter 3 focuses on the resilience aspect of P2P streaming systems. It reviews previous work of the three classes of P2P streaming systems. For that, the discussion is based on three criteria: (*i*) resilience (meaning robustness to churn and resistance to attacks); (*ii*) performance in terms of latency; and (*iii*) cost in terms of signaling overhead. Afterwards, the chapter also discusses a potential vulnerability in pull-based systems, in which an adversary attacks head nodes identified by inferring exchanged buffer maps.

Chapter 4 introduces the methodology used in this thesis. It first provides abstract models of P2P streaming systems. It also formulates the metrics used to evaluate and compare systems' performance and resilience in the rest of the thesis. After describing churn models, attack strategies (with and without global knowledge) are presented. In the second part of the chapter, OSSim–a generic simulation

framework for P2P streaming–is presented. The description includes OSSim's architecture, implementation and validation. The last part of the chapter introduces common settings and parameters used for later simulation studies of various P2P streaming systems.

Chapter 5 addresses the vulnerability of pull-based systems. In this case, an adversary with global knowledge targets head nodes to perform DoS attacks. The impacts of such attacks are then presented and discussed. Subsequently, the chapter introduces the striping scheme as the countermeasure to the DoS attacks on head nodes. This part includes the idea, design and specifications of the scheme. In later sections of the chapter, extensive simulation studies are conducted to show the superiority of the developed scheme.

Chapter 6 addresses the problem of pull-based systems in the presence of an inference attacker with local knowledge. The attacker identifies head nodes by inferring collected buffer maps and subsequently performs DoS attacks on those nodes. The system is formally modeled and a lower bound on the attack's accuracy is then derived. The bound is compared and validated with simulation results. Extensive simulation studies are then conducted to explore the dependence of the attack's accuracy on the parameters of the attacker model. Afterwards, the chapter introduces SWAP, a partner swapping scheme to counter the inference attacker. Results from comprehensive simulations are then presented to show the effectiveness of the SWAP scheme in mitigating the negative effects of the attacker.

To enhance the resilience while keeping the latency and overhead low, Chapter 7 tackles the vulnerabilities in hybrid systems, including the unsecured self promotion and the fragile single-tree backbone. The system and the attacker are formally modeled. Subsequently, the Resilient Backbone Construction Scheme (RBCS) is presented as the countermeasure to attacks. After that, the chapter presents results from comprehensive simulation studies to show how RBCS mitigates the attacks. We also present results concerning the cost of the scheme in terms of signaling overhead.

Finally, Chapter 8 concludes the thesis by summarizing its content and contributions. Lessons learned from the thesis as well as potential followup research directions are discussed.

2

Background

This chapter presents essential background information for this thesis. First, Section 2.1 introduces two methods for video deliver over the Internet. Next, Section 2.2 focuses on live streaming and its requirements. Subsequently, the chapter sketches four main architectures supporting video streaming over the Internet in Section 2.3. After describing three classes of P2P video streaming systems in Section2.4, we summarize the chapter in Section 2.5.

2.1 Video delivery over the Internet

This section introduces alternative methods to deliver videos over the Internet, namely Download-and-Play and Streaming. We also discuss advantages and disadvantages of each method.

2.1.1 Download and Play

In the Download-and-Play method [3], users request the videos they would like to watch from a streaming server. Upon receiving those requests, the server sends the requested video files to the users. When they receive the whole video files successfully, they start watching the videos. Despite its simplicity, the Download-and-Play method induces two major disadvantages: (*i*) The user has to wait for very a long duration before the video starts. (*ii*) This method requires the user's end device a large storage to cache the downloaded video file. If the user soon realizes that the video is not interesting to him and subsequently stops watching, then the whole consumed bandwidth and the large storage are wasted. (*iii*) Most critically, this

method is inapplicable to deliver videos from live events, such as music festivals or football matches.

2.1.2 Streaming

Video streaming [3] solves the problems of the Download-and-Play method. The idea of video streaming is to allow users to watch a video as soon as a small portion of the video is successfully received at the users. In addition, the user keeps downloading the rest of the video while watching the video. In other words, the video is watched while being downloaded. This method requires that the video is split into many small chunks. More importantly, those chunks need to be encoded in a way that their decoding does not require all other chunks [28, 72, 5]. Furthermore, the streaming method also requires advanced compression techniques [10, 4] to significantly reduce the size of the video chunks before they are transmitted.

As compared to the Download-and-Play method, the waiting time in the streaming method is significantly reduced. Additionally, the users only need to keep a small portion of the whole video, thus, reducing the storage. If the user decides to stop watching the video, only a small storage of the downloaded portion is wasted. Furthermore, since the users can start watching the videos quickly, the streaming method is suitable to deliver live videos.

2.2 Live video streaming

Live video streaming offer users the capability of watching live events over the Internet. In a live streaming system, streaming servers send video streams of an event to its subscribing audience. In principle, the audience should watch the events at same time as they are happening. However, there is always a small latency due to encoding and decoding processes and the transmission of the video chunks from the servers to their users [3]. However, the smaller the latency the better the quality a streaming system offers its users. Live video streaming has several requirements to meet the users' satisfaction: (*i*) The method needs to ensure a smooth play-out of the video. This means that the rendered video has to progress at the same pace as it is recorded at the source. If the stream delivery is interrupted at some stage due to various reasons, such as the server breaks down or users' devices fail, the missing video portion is skipped. Waiting for the retransmission of a portion results in a further lag to the live events. (*ii*) Live video streaming needs to minimize the play-out lag. That is the difference of play-out points among users and especially between users and the server.

The critical timing requirements of live streaming raise a challenge for this service. Being unable to satisfy those timing requirement can drastically reduce users' quality

of experience (QoE), meaning the impression of the users about the service. A low QoE can discourage the users from using the service again. However, there are few critical constraints in the context of live video streaming. On the one hand, the users are unable to watch a missed part of the video soon unless they diverge from the live event they are watching. On the other hand, the users are unable to pre-fetch the video stream in advance simply because it has not yet been available to retrieve. Ensuring timing requirements of live video streaming in a best-effort environment, such as the Internet, is even more challenging. Congestion is common in that environment [29], causing transmission delay or even data loss. Video chunks might also be received with errors, which make them unable to be decoded correctly. In both cases the video chunks are not successfully rendered at the users.

Regardless of the challenges, live video streaming has become a high-demand service nowadays. In the coming section, we introduce notable architectures supporting video streaming over the Internet.

2.3 Video streaming over the Internet

The combination of advances in video streaming and the constantly growing user demand to watch live events have driven video streaming over the Internet a popular service. Statistics show that the traffic from video streaming has increases drastically and this trend is projected to continue in the coming years [21]. Several architectures have been developed to satisfy this huge demand. They are unicast, IP multicast, Content Delivery Network (CDN) and the Peer-to-Peer architectures.

In a straightforward approach, the unicast architecture [6] requires the streaming server to deliver the video directly to its users. To do that, every user has to establish a unicast connection to the server. As long as the server has enough resources, mainly its upload bandwidth, every newly arriving user can receive the video stream with a low latency. Despite its simplicity, this architecture causes a very high bandwidth demand to the server. The reason is that the demand increases linearly with the number of users. Additionally, this architecture is inefficient for the Internet's core network because the network has to simultaneously deliver several times of the same stream to users.

To mitigate the bandwidth demand to both the streaming server and the core network, the IP multicast architecture [24, 75] is developed. In this architecture, the server sends only one stream on the shared paths of the connections between the server and its users. Only at the routers where the paths to different users diverge, the stream is then replicated. Consequently, the video traffic duplication in the Internet's core network is minimized. Additionally, the delay is also minimized since the video stream is replicated very early at the IP layer without being passed through upper ones. Despite those huge advantages, IP multicast has two major drawbacks.

(*i*) It requires routers to maintain a state for each multicast group, which complicates the management and limits the scaling at the IP layer. (*ii*) The architecture requires modification at the infrastructure level, thus slowing down the deployment [25]. So far, the IP multicast architecture only appears in restricted deployments (e.g., at IPTV backbones) [46].

The Content Delivery Network (CDN) architecture addresses the deployment difficulty of the IP multicast architecture by introducing a dedicated network consisting of several replication servers [60]. The servers are placed strategically in regions with high traffic demand. Therefore, they are equipped with abundant bandwidth to deliver the video to large local audiences. In this architecture, the video source sends the stream to the CDN only once. The stream is then distributed within the CDN to and duplicated there to serve nearby users via unicast connections. In addition to a drastic reduction of the bandwidth demand on the streaming server, the CDN offers several advantages, such as high capacity and high performance. However, the major drawbacks of a CDN are high costs and a poor support to live streaming.

The Peer-to-Peer (P2P) architecture resolves the high-cost problem of a CDN by requiring users (or peers) to contribute their resource, mainly the upload bandwidth, to disseminate video streams. Participating peers not only receive but also send video streams to others. Since P2P streaming only requires a modest upload bandwidth from the streaming server (or source), it reduces the bandwidth demand on the source. Additionally, the architecture also enables scaling up the system since peers participate with their own resources. The more peers join the system, the more resources the system has. Furthermore, unlike IP multicast the P2P architecture does not require any modification in networking devices. This enables and speeds up the deployment of the service to large audiences.

Despite the above advantages, the P2P architecture poses higher latency than its alternatives since a peer generally receives the stream after it is forwarded via several other peers. A part from that, the general challenge for P2P streaming is building an overlay interconnecting the source and peers and deciding a method to disseminate video streams so that the overall system maintains a low latency and a low signaling overhead. Since ordinary peers can join and leave the system freely at any time, churn is inevitable. Building an overlay robust to churn is a must to maintain the service without disruptions. Furthermore, the system has to provide attack resistance. To damage the system, a relevant subset of peers can be shut down to disrupt the video dissemination to the rest of the peers. Such sabotage can be performed by an individual hacker simply for fame or by an organization or a government to block the delivery of a certain content.

In the coming section, we describe in more detail the general architecture of a P2P video streaming system as well as different classes of the systems.

2.4 P2P video streaming

As we discussed in the previous section, the P2P video streaming architecture offers various advantages to deliver live videos over the Internet. These advantages include the scalability, the ease of deployment and the reduced demand of bandwidth on the streaming server. Because of that, P2P streaming has been becoming a viable solution [45, 44, 26]. Many systems have been proposed and deployed in real-world conditions so far, such as PPLive [51], Sopcast [56] and UUSee [59] among others.

Regardless of their variety, they share the same architecture. From an abstract level, a P2P video streaming system consists of three main elements [26, 32]. They are one or several *Channel Servers*, one or several *Streaming Servers*, and a multitude of *Peers*. This architecture and its components are illustrated in Figure 2.1.

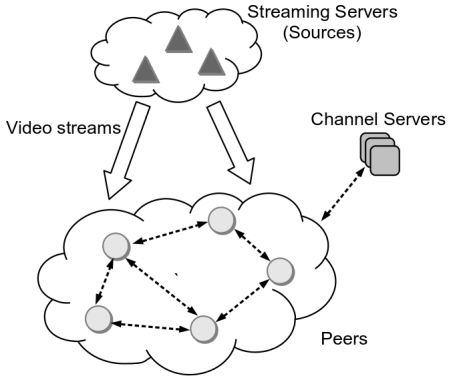


FIGURE 2.1: Architecture of a P2P streaming system with three main elements: (i) channel servers, (ii) streaming servers, and (iii) a multitude of peers

When an ordinary user (or peer) would like to join the system, it has to contact a rendezvous node whose name or address is well-known to all the peers. In this architecture, a channel server serves as the rendezvous node. The server maintains a list of channels from which the user can select to watch. In addition, the channel server can also provide a bootstrapping service. Hence, it additionally holds a list of active peers (and streaming servers) per channel. The list is used to bootstrap joining peer with knowledge about one or several other peers in the respective channel. After that, the new peer communicates with other peers to receive the video stream from them. The video stream is originated at streaming server (or source). For each channel, a streaming server (or source) distributes a continuous stream of video chunks. Multiple streaming servers can be accounted for by using one super node with their total upload bandwidth. Moreover, the source participates in the streaming system as a normal peer, except that the source does not request video chunks. The connections that peers and the source establish with one another form an overlay. This overlay network exists logically on top of physical networks.

So far, various P2P streaming systems have been proposed. They differ from each other in two main design choices: (*i*) The overlay topology, which is either a tree or a mesh; (*ii*) The method that peers retrieve the video stream from one another, which is either push or pull. Therefore, P2P streaming systems can be categorized into three main classes: push-based, pull-based and hybrid push-pull. They are described in detail as follows.

2.4.1 Push-based systems

Push-based systems construct a tree-like overlay topology as illustrated in Figure 2.2. Each link between two nodes in the topology represents the parent-child relation between the respective two peers. To establish the connection, the child node subscribes to its parent node. This also means that the child node receives the whole stream of video chunks from its only parent node. As soon as new video chunks arrive at the parent, the parent node forwards (or pushes) those chunks to its child nodes. The chunk pushing process starts from the source and continues at intermediate nodes. A peer situated at the leaf node of the tree stops forwarding since it has no children. Early push-based systems [20, 34, 35] follow this design.

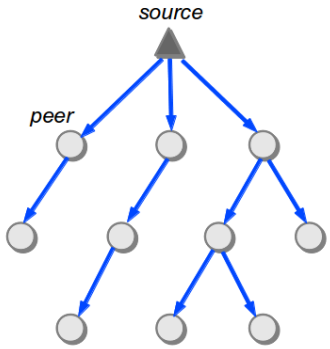


FIGURE 2.2: Overlay topology of a push-based system with a single tree

Push-based systems offer a low latency in chunk delivery. Specifically, it introduces a minimized difference between the instance the video chunk is generated and the instance the video chunk is received at a peer. However, push-based systems has two major drawbacks. First, the single-tree overlay is severely fragile to churn. When a node's parent is suddenly absent, due to churn, its child nodes immediately lose the whole video stream. Subsequently, all the successors of those child nodes also lose the entire video stream. If the child nodes of the leaving parent cannot find their alternative parents soon enough, not only they but their successors need to find new parents for themselves. In other words, the sub-tree rooting at the leaving parent is broken and need to reconstruct. When churn happens frequently, the overall topology is often in the recovery state. Therefore, it is hard for disconnected

peers to find their reliable parents who still have available bandwidth to send them the video stream. The system's performance is affected drastically. Second, the leaf nodes do not contribute their uploading bandwidth because they do not have any child nodes. This is a degradation of the system's bandwidth utilization, because leaf nodes account for a large portion of peers in the system.

To address the drawbacks of the push-based systems with only one tree, multiple-tree systems have been proposed. Multiple trees all root at the source and span all the peers. Figure 2.3 illustrates a system consisting of two overlay trees. By participating in several tree overlays, each peer potentially has several different parents. Even when some of its parents leave or fail, the peer can still remain connected to the overlay. Thus, they do not lose the video stream completely.

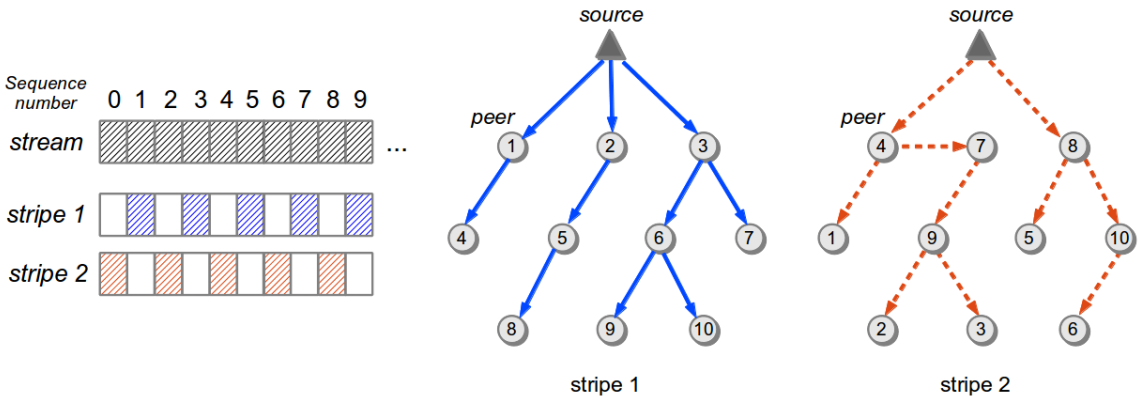


FIGURE 2.3: Overlay topology of a multiple-tree push-based system with two trees disseminating two video stripes

Furthermore, to reduce the direct dependency of a child node on a particular parent, the video stream is split into multiple partial streams (or stripes). Figure 2.3 illustrates how a stream of video chunks is split into two stripes. Stripe 1 consists of chunks whose sequence numbers are odd. Whereas stripe 2 consists of even chunks. Video chunks in each stripe is disseminated using a separate spanning tree. A child node should receive different stripes from different parents. When a peer loses a stripe, because its parent leaves or fails, the peer still can receive other stripes. The multiple-tree systems can further increase its robustness by reducing the indirect dependency on a particular node, meaning the number of successors it has. This is done by ensuring the so-called *inner-node disjointness* property in which an inner node in a tree is a leaf node in all other trees. As illustrated in Figure 2.3, nodes 1, 2, 3, 5, and 6 are inner nodes in tree 1. However, they are all leaf nodes in tree 2. Another advantage of a multiple-tree push-based system is that peers' uploading bandwidth is utilized. Since peers are inner nodes in at least one tree, they contribute their bandwidth in disseminating chunk in that tree. The drawback of multiple-tree push-based systems is the complexity in maintaining several trees at the same time.

2.4.2 Pull-based systems

Maintaining multiple trees at the same time is a complex task, especially when those trees need repair due to churn. To overcome this issue, pull-based systems interconnects peers in a mesh topology as illustrated in Figure 2.4. Peers become partners with each other, meaning that there is no explicit dependency between those peers. The system's connectivity is strengthened since each peer has multiple paths to the source. The mesh overlay topology is illustrated in Figure 2.4.

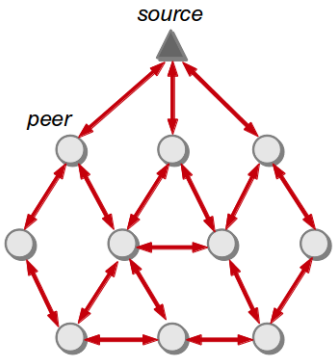


FIGURE 2.4: Overlay topology of a pull-based P2P streaming system

To obtain the video stream, peers actively and frequently request video chunks from others. Therefore, peers need to exchange Buffer Maps (BM) with each other. A buffer map is a small packet containing the availability information of the peer's video buffer at a specific instance. To provide partners with a fresh status of the video buffer, peers need to send BMs periodically. Each peer uses the BMs it receives from its partners to decide which chunks it should request from which partners. This decision is executed via chunk scheduling algorithms. The chunk scheduling problem can be reduced to the parallel machine scheduling problem and is proven to be NP-hard [82]. For this reason, pull-based systems often use heuristics (e.g., the Rarest-First in DONet [82]).

With its simplicity in maintaining the overlay and protocol implementation, the pull-based design is widely preferred in various real-world deployments [38, 2, 51, 77, 52, 56, 59]. Nevertheless, pull-based systems introduce a high signaling overhead and a high latency, due to the buffer map exchange and chunk request operations.

2.4.3 Hybrid push-pull systems

The approach of hybrid push-pull systems [41, 81] is to combine the advantages of both the push-based and pull-based systems. It means that a hybrid system strives to achieve the increased robustness of a pull-based system with the high performance of a push-based one. For that reason, a hybrid system organizes peers

in both overlays. The tree overlay interconnects peers to deliver video chunks without requests. Meanwhile, the connections in the mesh overlay allows peers to pull video chunks from others or send chunks to others upon requests.

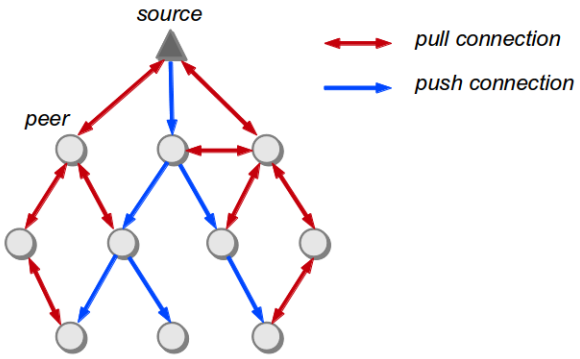


FIGURE 2.5: Overlay topology of a hybrid P2P streaming system

As illustrated in Figure 2.5, the blue arrows connects the nodes to form a delivery tree. Therefore, the video stream is pushed from the source to its child nodes. The process continues at the intermediate nodes until the stream reaches the leaf nodes of the tree. Additionally, the red arrows interconnects peers to form a mesh topology.

The tree overlay allows peers to receive the video stream with a minimal latency. Furthermore, the signaling overhead is reduced because those tree nodes stop sending chunk requests. In addition to that, the mesh overlay allows peers to improve the connectivity among peers. Peers in the mesh overlay can pull chunks from peers in the tree overlay, thus, reducing the overall latency. Peers in the tree can also pull chunks from their partners in the mesh in case the stream delivery is disrupted in the tree.

In spite of the above advantages, hybrid systems have not demonstrated its superior in real-world deployments. Furthermore, the resistance of hybrid systems to attacks has not been studied extensively so far.

We have presented the architecture and the general challenges for P2P streaming systems. Subsequently, we have discussed three classes of systems, which are push-based, pull-based and hybrid. In the coming section we summarize this chapter.

2.5 Summary and discussion

In this chapter, we present a brief background knowledge on the area of video streaming over the Internet. First, we introduce and differentiate the two main methods to deliver videos over the Internet to users: Download-and-Play and streaming. The former method causes a long waiting time and possibly a waste of storage and bandwidth. More importantly, the method is unsuitable to deliver live videos. The latter

method offers users the capability to watch a video as soon as they receive a small portion of the video. The downloading process keeps running in the background while the users are watching the videos. Therefore, video streaming has advantages of a short start-up latency and a low buffering storage. To do so, video streaming requires advanced coding techniques to significantly reduce the video size.

Live video streaming has become a popular service because it allows users to watch live events, such as football matches or music festivals over the Internet. Since the live video needs to be delivered as quickly as possible to users, live streaming requires strict timing commitments. Its goals are to reduce the viewing lags between users and the source as well as among users. The combination of a consistently growing demand to watch live videos and the advances of video coding techniques generates a huge bandwidth demand on the streaming servers and the Internet's infrastructure. To alleviate this demand, various video streaming architectures have been proposed. They are unicast, IP multicast, Content Delivery Network (CDN) and Peer-to-Peer (P2P).

The unicast architecture is a straightforward solution for video streaming. However, it does not reduce the bandwidth demand on the streaming server. IP multicast significantly reduce both the demand at the server and the duplicated traffic at the Internet's core network. To do that, the video is only replicated at the router where the paths to different users diverge. However, deployment complexity prevents this architecture from being a widely applicable solution. CDNs resolve the deployment complexity of the IP multicast architecture. To do that, a CDN replicates the video traffic within its dedicated network consisting of servers with a large amount of bandwidth. The benefits that CDN offers, such as a low latency and deployment simplicity result in a high cost. More importantly, CDN's support to live videos remains limited.

To address the problem of its alternatives, Peer-to-Peer (P2P) streaming has been proposed. In this architecture, users (or peers) contribute their bandwidth to deliver the video stream to one another. Therefore, the bandwidth demand from the streaming server (or source) is significantly reduced. Nevertheless, the general challenge of P2P streaming is to maintain the stream delivery under churn, that is the leaving and failing of peers. More severely, P2P streaming has to resist to attacks caused by an adversary aiming to damage the system.

So far, various P2P streaming systems have been proposed. Despite their variety, they share the same architecture which consists of streaming servers, channel servers and a multitude of peers. The bootstrap process for each peer joining the system is in general the same. Nevertheless, P2P streaming systems differ from one another in two design choices: (*i*) The topology of the overlay interconnecting peers, which is either a tree or a mesh; (*ii*) The method for retrieving the video stream from other peers, which is either push or pull. Subsequently, P2P streaming systems can be classified by the following three classes: push-based, pull-based and hybrid

push-pull.

A push-based system organizes peers in one or several overlay trees. A peer subscribes to its parent and receives the video stream directly from there without requests. Therefore, push-based systems offer a low latency and a low signaling overhead. Multiple-tree push-based systems avoid the fragility of the single-tree overlay under churn. Subsequently, the video stream is split into stripes (or partial streams). Each stripe is delivered in a separate tree. In a different approach, a pull-based system organizes peers in a mesh overlay. Peers become partners and exchange buffer maps, small packets containing the availability information of peers' video buffers. Using those BMs, peers decide which chunks to request from which partners via chunk scheduling algorithms. Pull-based systems are robust to churn since each peer has several paths to the source. However, the buffer map exchange and frequent chunk requests cause the system a high latency and a high signaling overhead. Hybrid P2P streaming systems strive combine the advantages of both push-based and pull-based classes of systems. However, hybrid systems have not demonstrated their advances in real-world conditions. Furthermore, their resistance to attacks has not been extensively studied in the literature.

In the coming chapter, the thesis reviews various P2P streaming systems with regard to their robustness to churn and resistance to attacks.

3

Related work

Various P2P video streaming systems have been proposed during the past years. However, we restrict this chapter to previous work on constructing resilient overlays because this dissertation focuses on the resilience aspect of P2P streaming systems. Specifically, this chapter comprehensively reviews the prior work from three aspects:

1. Resilience (meaning both robustness to churn and resistance to DoS attacks);
2. Performance in terms of latency; and
3. Cost in terms of signaling overhead.

The systems are categorized into three classes: push-based, pull-based and hybrid. They will be discussed in detail as follows.

3.1 Push-based systems

Recalling from the background in Chapter 2, push-based systems construct a tree-like overlay topology. Each link between two nodes in the topology represents the parent-child relation between the respective two peers. The child node subscribes to its parent node to receive the stream of video chunks. As soon as new video chunks arrive at the parent, it forwards (or pushes) those chunks to its child nodes. This way, the latency between the instance the video chunk is generated and the instance the video chunk is received at peers is minimized. Early push-based systems [20, 34, 35] follow this design. However, they are severely fragile to churn. When a node's parent is suddenly absent, due to churn, its child nodes immediately lose the whole video stream. Subsequently, all the successors of those child nodes also lose the entire video stream. If the child nodes of the leaving parent cannot find their alternative

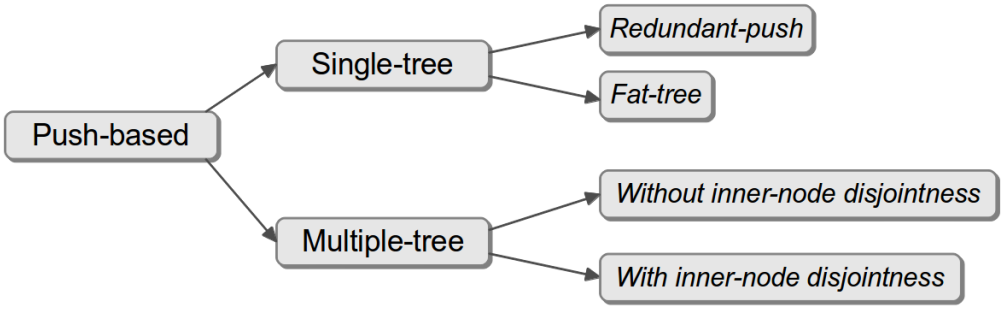


FIGURE 3.1: Category of push-based systems

parents soon enough, not only they but their successors need to find new parents for themselves. In other words, the sub-tree rooting at the leaving parent is broken and need to reconstruct. When churn happens frequently, the overall topology is often in the recovery state. It is hard for disconnected peers to find their reliable parents who still have available bandwidth to send them the video stream. The system's performance is affected drastically.

To improve the robustness of push-based systems to churn, several approaches are proposed. They differ from one another in how they construct the overlay topology and are categorized into two groups: single-tree and multiple-tree systems.

3.1.1 Single-tree

To make the overlay robust to churn, systems in this approach try to minimize the probability of breaking the tree overlay by one of the two strategies: redundant-push or fat-tree. The redundant-push strategy represented by the Probabilistic Resilient Multicast (PRM) [8] establishes extra connections alongside a single tree overlay to improve the connectivity between nodes, thus preventing the tree overlay from partitioning. Additionally, one peer at one end of the connection pushes video chunks to the peer at the other end. An extra stream allows the receiving peers to maintain an uninterrupted stream when the other stream is lost. On the one hand, this ensures the stream delivery of the system. On the other hand, significant traffic redundancy is generated. To diminish the redundancy, only a small fraction of peers are allowed to establish additional connections. It has been shown that the whole system can maintain a high delivery ratio under churn.

To avoid redundant traffic completely, the fat-tree strategy represented by Nemo [12] seeks to arrange peers within the overlay topology without any additional connections. The idea is to minimize the number of intermediate parents of every nodes. Intuitively, the shorter the path between the source and a peer, the less likely the peer can be disconnected from the overlay when any of the intermediate nodes along the path leaves or fails. Therefore, the system considers nodes' upload bandwidth, or

the number of children nodes can accept. The more bandwidth a nodes has the closer to the source the node is placed. On the contrary, nodes with less bandwidth are moved farther away in the downstream. The resulting tree topology is low and fat. Since churn is more likely to happen on downstream nodes, the overall connectivity of the overlay is less affected.

Even though systems adopting the above two strategies work well under churn, they are not resistant to DoS attacks. The whole system depends on a small number of relevant nodes that are close to the source. Attacks on those nodes can disconnect the source from the remaining peers, thus disrupting the stream delivery of the whole system.

3.1.2 Multiple-tree

Addressing the drawbacks of the single-tree approach, multiple-tree systems are proposed. Each tree roots at the source and spans all the peers. By participating in several tree overlays, each peer potentially has several different parents. Even when some of its parents leave or fail, the peer can still remain connected to the overlay, thus do not lose the video stream completely. Furthermore, to reduce the direct dependency of a child node on a particular parent, the video stream is split into multiple partial streams (or stripes). Video chunks in each stripe is disseminated using a separate spanning tree. A child node should receive different stripes from different parents. When a peer loses a stripe, because its parent leaves or fails, the peer still can receive other stripes.

The multiple-tree systems can further increase its robustness by reducing the indirect dependency on a particular node, meaning the number of successors it has. This is done by ensuring the so-called *inner-node disjointness* property in which an interior node in a tree is a leaf node in all other trees. Unfortunately, realizing this property is challenging. It requires that participating peers can sense their relative positions in different trees. Therefore, peers need to exchange with one another extra information. This increases the signaling overhead and more importantly, allows for collecting massive information to infer the system's structure. An adversary can abuse this information to attack certain nodes whose sudden disruption can severely affect the whole system. For those reasons, not all multiple-tree systems strive to implement this property. In the following, we categorize the multiple-tree systems into two groups: *Without inner-node disjointness* and *With inner-node disjointness*. Since the dissemination of stripes' video chunks on overlay trees is similar between those approaches, our subsequent discussion focuses on the constructions of the overlays only.

Without inner-node disjointness

Systems in this approach ignore the inner-node disjointness to avoid the cost from signaling overhead. The overhead comes from constant updates of the system's topology. It becomes costly when peers join and leave frequently, especially in large systems. Instead, systems, e.g., Chunkyspread [64], only aim at balancing the load, which is the upload bandwidth demand, with regard to the given target load of each node. First, a random graph is created and then locally optimized by peers' load and latency between peers. Node discovery on the graph is performed by Swaplinks [65]. Then, each node advertises both its current and maximum load, so that other nodes can decide whether they should subscribe to this one. To avoid loops bloom filters [70] are used. However, there are two major issues for this approach. First, optimizing local connections mainly by peers' own upload bandwidth does not guarantee optimized topologies for all delivery trees. Nodes with high bandwidth might have many successors and therefore become potential targets of attacks. Second, advertising peers' capacity can be abused. Malicious peers can freely collect those information to infer relevant nodes and attack them. Attacks on branches with many successors can cause significant damage to the system.

With inner-node disjointness

To avoid the consequences of an unbalanced topology under attacks, systems, such as THAG [58] and NHAG [37] strive to guarantee the inner-node disjointness property. The main idea is to group all participants into several small Arrangement Graphs (AGs) [23]. An AG is a regular interconnection topology which allows for embedding multiple trees into its topology. This overlay construction offers several benefits by design. First, AGs enable the inner-node disjoint trees. Second, the maximum delay is guaranteed since the distance between any two nodes in the AG is limited to three hops. And third, the overlay is robust to churn. Due to the small number of nodes in each AG, the absence of a few nodes can be compensated by others in the AG taking delegated roles. When the system size exceeds the maximum number of nodes that an AG can host, a new AG is created. In this way, the whole overlay network is built as a hierarchical structure where each element in the hierarchy is an AG. The system is therefore also scalable. To further improve the system to support the heterogeneity of peers, variable-size AGs are introduced in NHAG. Unfortunately, the systems in this approach have a critical drawback that is the expose of their topology. Every new peer joins the system by contacting the top AG to know whether it can be accepted directly or need to iteratively communicate with other AGs. Therefore, a malicious peer can easily identify the top AG when joining the system. The subsequent DoS attacks targeting the top AG whose size is relative small can disrupt the whole system.

In a different approach to implement the inner-node-disjointness property, Split-

Stream [18] uses one Pastry [54] overlay for each stripe tree. In addition to that, SplitStream strives to support the heterogeneity by limiting the node degree by a push-down operation. If a peer is unable to accept a connection request, it forwards the request to its existing child nodes. This way, heterogeneous peers fairly balance their forwarding load with regard to their upload bandwidth. Unfortunately, the push-down operation can lead to very long branches which increase the latency. Furthermore, near-root nodes of a long branch can become targets of DoS attacks, affecting all their successors.

In a different approach, the Optimally Stable Topologies (OST) group of systems [73, 57, 15, 27] are proposed. They aim at constructing an overlay resilient for both churn and DoS attacks. Therefore, they strive to build more balanced topologies that keep the impact of worst-case DoS attacks at a minimum. The topologies that satisfy this requirement are called optimally stable topologies. They also ensure the inner-node-disjointness property. However, such optimal topologies require a global knowledge, which is unrealistic in real networks.

To practically construct topologies containing the OST's properties, a distributed procedure is designed. Using a cost-based optimization, nodes periodically try to optimize the topology based on local knowledge only. Each node adjusts its child list by deciding which child should be kept or removed. A cost function is therefore defined to calculate the contribution of each child. The node identifies the link to the child that contributes the least to the overall system (i.e. the costliest link) as well as the link to a child, which might be able to forward video packets to an additional child (i.e. the cheapest link). The node will then try to drop the costly child to the cheap child in order to improve its own local situation.

To find those two children, a node uses cost functions to rate each of its links. This cost function is calculated from four separate cost functions: stripe cost, forwarding cost, balance cost, and dependency cost. (*i*) The stripe cost's goal is to increase the costs of links in trees in which a node only has a few children. Ideally a node wants to forward only one stripe and that should be the stripe it has the most children in, hence links in trees with only a few children are costlier than links in trees with many children. (*ii*) The forwarding cost assigns higher costs to links that do not forward stripes. That way such nodes are more likely to be dropped and eventually become leaf nodes in that tree, thus not taking up spaces at the internal nodes. (*iii*) Next, the balance cost assigns higher costs to links to children with only a few successors. (*iv*) Finally, the dependency cost increases the costs of links to nodes that receive multiple forwarded stripes. This is undesirable because it increases the direct dependency of such a child on the node and consequently increases the number of lost video chunks when the node fails or leaves. Using different weights for different costs, the costs are combined to form a final cost function.

Even though the resilience of the OST scheme is proven theoretically, they have not been adopted in a large-scale real-world deployment.

3.2 Pull-based systems

Pull-based systems follow a different approach to build their resilient overlays. First, peers actively request video chunks from others. Therefore, missing chunks in one request can be obtained in the subsequent requests. Second, to interconnect peers, a pull-based system establishes a mesh overlay. The system's connectivity is strengthened since each peer has multiple paths to the source. However, pull-based systems differ from each other in their strategies to construct the overlays. In the following, we introduce and discuss the two strategies, namely directed and undirected mesh topologies. After that, we discuss a potential threat leading to DoS attacks in pull-based systems.

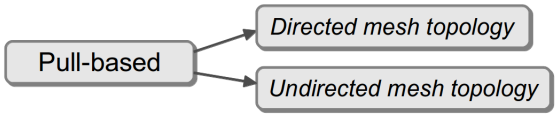


FIGURE 3.2: Categorization of pull-based systems

3.2.1 Directed mesh topology

To increase the probability that a peer can request chunks from at least one parent, DagStream [43] allows each peer to have multiple dedicated parents from its partners. The rest of the partners become the peer's children. As the result, the peer requests chunks from its parents and sends requested chunks to its children. By having multiple parents each peer reduces the probability of being isolated when some of its parents leave or fail. The overlay's topology becomes a directed mesh in which an edge represents the parent-child relation between two peers. However, a loop is formed when a child peer very far from the source becomes a parent of another peer close to the source. Unfortunately, loops can happen frequently with a directed mesh, preventing chunks from properly disseminated. DagStream solves this problem by two techniques. First, each peer is assigned a level, which is calculated from those of its parents. Peers' levels become larger as they position further away from the source. Second, a peer must find a parent whose levels is smaller than its own level, thus avoiding loops. With this overlay construction, DagStream has several advantages. Peers provision better the bandwidth demand from others since they know exactly how many children they have. DagStream also reduces the direct dependency of the peer to a particular parent. The overall system's robustness to churn is therefore strengthened. However, this strategy has several drawbacks. First, the lower the level of a peer is the more difficult for it to find an alternative parent because of the strategy to avoid loops. Second, the policies to select parents in DagStream prioritize peers that are close to the source. Those preferred peers can become the bottle-neck of the system [15] resulting in an hourglass-like topology. When many peers depend

on a few relevant parents, attacks on those parents can cause a huge negative impact on a large fraction of peers.

3.2.2 Undirected mesh topology

To resolve the inflexibility of the directed mesh topology, most pull-based systems [82, 52, 51] construct undirected mesh topologies instead. An edge in the mesh represents a partnership relation between two peers. A measurement study on PPLive, one of the largest running P2P streaming systems, has shown that its overlay resembles a random mesh [66]. A peer can request any of its partners to send it video chunks as long as the chunks are available there. Consequently, pull-based systems are robust to churn for two reasons. First, without constraints on levels, peers can easily find and establish new partnership connections to replace the dropped ones. Second, the reactive nature helps pull-based systems adjust quickly to changes. If the requested chunks do not arrive, a peer still can request them again from other partners.

Even though the robustness of pull-based systems to churn has been thoroughly studied, little attention has been paid on their resistance against DoS attacks. Especially, the consequences when an adversary identifies and attacks relevant nodes in the topology have not been explored. The concern increases when a measurement study from Hei et al. [33] has demonstrated that they can identify the system's structure of PPLive, a pull-based system. In the following, we describe the findings from the measurement study. After that, we discuss subsequent implications on the resilience of pull-based systems under DoS attacks.

Potential attacks basing on buffer map exchange

The information in exchanged buffer maps in a pull-based system reveals the overlay structure in general and head nodes in particular. This observation comes after the measurement study conducted by Hei et al. [33]. To understand overlay structure of a pull-based P2P streaming system, they measure PPLive, an active pull-based one. They assume that there is an ordering of peers according to their play-out points. Since the play-out points can be inferred from the buffer's availability information, they analyze the information in the buffer maps. Subsequently, they collect a large amount of buffer maps (BM) from peers by both active crawling and passive sniffing. Information from peers' BMs is aggregated to infer the play-out lags between peers. The results show that peers' play-out lags form distinguished tiers in the system, which also remain stable over time. Especially, partners of the source (or head nodes) can be clearly identified from their small play-out lags. Since head nodes situate geographically close to the source and having direct connections to it, they receive video chunks significantly earlier than others. This tiering effect also allows for inferring the flow of video chunks. The flow is likely to travel from the tier with

small lags to other tiers with much larger lags.

The feasibility of the above measurement raises a question on the resistance of pull-based systems against DoS attacks. An adversary can employ a similar technique to infer the tiers in the overlay structure, especially to identify head nodes. Completely shutting down peers of a certain tier can fragment the system's overlay and disrupt the flow of chunks. More critically, when head nodes are attacked the remaining peers completely lose the video stream, causing the whole system out-of-service. So far, the question of how pull-based systems can be affected by such attacks has not been addressed.

3.3 Hybrid systems

From the discussion in the preceding sections, it is evident that both the push-based and pull-based systems have advantages at some aspects and disadvantages at the others. Hybrid push-pull systems (or hybrid systems, for short) strive to combine the advantages of both classes, meaning to provide a low latency and a low overhead while maintaining a robust overlay to churn. Various hybrid systems have been proposed so far in the literature. They can be categorized into two main groups, namely the push-first or the pull-first depending on whether a hybrid system emerges from a push-based system or a pull-based one. The categorization of hybrid systems is illustrated in Figure 3.3.

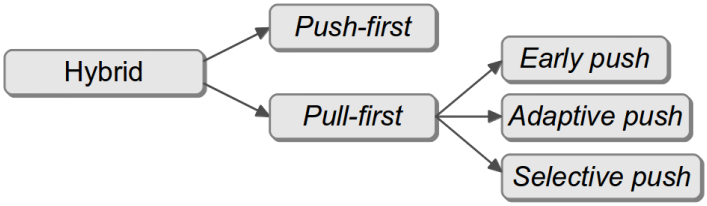


FIGURE 3.3: Categorization of hybrid push-pull systems

3.3.1 Push-first approach

A push-first system, such as TOMO [7], bootstraps from a multiple-tree overlay. Therefore, it tries to solve the fragility problem of the tree topology. Consequently, it establishes several mesh connections between tree nodes closer to the source, or so-called top nodes. By organizing peers in that way, the overlay consists of a mesh layer, covering top nodes, and a tree layer, covering the remaining peers. The mesh connections strengthen the connectivity between top nodes. They have more influences to the overall robustness of the system's overlay than those which are farther down at the leaves of the trees. In addition, to ensure the stream delivery,

duplicated video chunks are also sent from one node to the other of a mesh connection. To constrain the redundant forwarding of video chunks, peers maintain those mesh connections for limited periods only. Despite the cost of redundant forwarding, tree-first systems maintain a low latency while keeping the overlay robust to churn. However, determining sizes of the mesh and tree layers requires a global knowledge of the system, making the approach unrealistic, especially in large systems.

3.3.2 Pull-first approach

In a different approach, pull-first hybrid systems leverage the robust mesh topology by bootstrapping from a pull-based system. Subsequently, they strive to reduce both the latency and the signaling overhead. Consequently, peers convert mesh connections for chunk pulling into dedicated connections for chunk pushing, thus, reducing the latency. Since peers stop frequent chunk requests and buffer map exchange after the conversion, the signaling overhead is also reduced. However, different systems apply different conversion strategies. They are early push, adaptive push and selective push.

Early push

Early hybrid systems, such as Coolstreaming [76], iGridMedia [81], and [80], fall into this category. In those systems, a peer A pushes video chunks directly to the requesting peer B as soon as a few chunks are successfully sent. Afterwards, B can stop sending chunk requests. Moreover, those peers can stop exchanging buffer maps, significantly reducing the signaling overhead. Furthermore, to reduce the direct dependency to a particular peers, the video stream is also divided into stripes, similar to that of a multiple-tree push-based system. A peer should receive different stripes from different parents. There are algorithms to ensure an optimal strategy of scheduling pushed stripes [19]. When peer B experiences a certain ratio of missed chunks, it requests buffer maps from its partners. Then it resumes the chunk request operations.

There are several drawbacks for this strategy. (i) the system's overlay resembles a mesh topology consisting of multiple implicit spanning trees, one for each stripe. However, those trees are in general unoptimized because topology optimization is not performed. Therefore, possible long branches at some overlay trees can increase the system's latency. More importantly, a large fraction of peers might depends on a few relevant ones. When those peers leave or are attacked, the system can be badly affected. (ii) peers might not receive a constant stream of video chunks in highly dynamic scenarios. Due to switching the delivery method from pull to push too early, peers probably need to frequently switch back to pull methods, causing extra signaling overhead. (iii) the strategy allows for buffer map requests from a

peer at any time as soon as the peer needs to convert back to the pulling mode. An inference attacker can abuse this operation to collect buffer maps quickly and easily for inferring head nodes. Attacks on those nodes can disrupt the stream delivery in all stripe trees.

Adaptive push

To overcome the limitation of the early push strategy in highly dynamic scenarios, Wichtlhuber et al. develop TRANSIT [71] with an adaptive switching between the pulling and pushing modes. TRANSIT introduces for each stripe subscription a *lifetime* parameter. The subscribed peer forwards all chunks of a certain stripe within the lifetime period. Shorter lifetimes are used in a more dynamic scenario, e.g., when peers join and leave frequently, which is eventually similar to the pulling mode. Whereas, in less dynamic scenarios, longer lifetimes are used, so that the system resembles a pushing mode. Consequently, peers in TRANSIT can switch seamlessly between the pulling and pushing modes by varying the lifetimes of their subscriptions.

However, TRANSIT has the following two drawbacks: (*i*) They do not distinguish between short- and long-lived peers. The dynamics of ephemeral peers causes frequent connection repairs to their neighboring peers. This leads to the overlay's inherent instability and and extra overhead. (*ii*) Sooner or later, the system's overlay topology eventually evolves into multiple-tree push-based systems with suboptimal topologies. The delivery trees might be very high, which increases latency. More critically, a small set of peers might be preferred by many successors in several trees. When those critical peers fail or are attacked the whole system can be affected drastically.

Selective push

The previous strategies have two common issues: (*i*) the implicit trees from push connections are unoptimized, degrading the performance in benign scenarios and potentially increasing damages when the system is attacked; (*ii*) all peers are allowed to participate in the tree overlays, leading to inherent instability of the overlays. The selective-push strategy addresses the above two issues with systems, such as mTreebone [68] and its variation HyPO [16]. Instead, peers are allowed to join the push-based backbone only when they stay sufficiently long in the system. The underlying reason is that those peers might stay longer with a high probability [13]. Those peers are called stable peers. To identify stable peers, peers perform an operation named self-promotion, i.e., peers observe their ages to decide whether it should join the tree. Initially, all peers are organized into a mesh overlay. Afterwards long-lived peers whose elapsed lifetimes exceed certain thresholds join the push-based backbone

overlay. Backbone peers push video chunks to their children in the backbone and at the same time response to chunk requests from other peers. Due to their stability, backbone peers though in a small number, deliver majority of video chunks in the system [68].

The design of mTreebone introduces distinguish characteristics. First, mTreebone has a low source-to-peer latency because the backbone tree is constructed to minimize the system's latency. Since the backbone delivers the majority of video chunks, the overall latency of the system is reduced. Furthermore, the system offers a low signaling overhead for overlay maintenance. Since the backbone overlay consists of a small set of peers that are stable, recovering the overlay's connections is unlikely. Furthermore, the overlay's outskirt consists of less stable peers but their dynamics only cause the outskirt to recover but not the backbone.

Unfortunately, mTreebone induces several flaws that an adversary can exploit to attack the system. (*i*) The backbone overlay of mTreebone is a single tree which is vulnerable to attacks. Shutting down a few stable peers that position closely to the source disrupts the video stream delivery to their successors from the source. Even though by design, backbone peers should be able to request missing chunks from their partners in the mesh overlay, this is insufficient. A sudden demand from a large portion of the backbone peers can overload the mesh, which cause congestion and delay. The system's performance can therefore be severely affected. (*ii*) mTreebone exposes the information on its internal structure, including the distance of every peer to the source and its number of children as well as the list of backbone nodes. Exploiting this information increases the severity of attacks. (*iii*) Self-promotion allows for cheating by manipulating the calculation of ages. Selfish peers might exploit this flaw to join the backbone earlier despite being unreliable, thus causing instability.

3.4 Summary and problem statement

This chapter reviews P2P streaming systems proposed to construct resilient overlays. We categorize them into three groups: push-based, pull-based and hybrid. Due to the focus of the thesis, our discussion concentrates on the following three aspects: (*i*) resilience, meaning robustness to churn and resistance to attacks; (*ii*) performance in terms of latency, and (*iii*) cost in terms of signaling overhead.

Early push-based systems construct a single tree topology spanning all peers to push video chunks, which is fragile to peer churn. To tackle this problem, strategies, such as redundant-push or fat-tree, are developed. They either establish a few extra connections between peers or place peers with more children closer to the source. These strategies are helpful in case of churn. However, since each peer has only one parent, the whole system can easily depend on a few peers close to the source.

Attacking those peers affect the systems drastically.

The multiple-tree push-based systems address the inherent weakness of the single-tree topology. They allow peers to have multiple parents, thus reducing the direct dependency on any single parent. Consequently, multiple trees root at the source to span over all peers. Furthermore, the video stream is split into several stripes and each stripe is delivered on a separate spanning tree. When a peer loses some of its parents, it loses chunks in the respective stripes only. It still receives the rest of the stripes from its remaining parents. Consequently, the system is robust to churn. To even further improve the resilience of multiple-tree push-bases systems to both churn and DoS attacks, the Optimally Stable Topology (OST) class of systems is developed. The systems strive to minimize both the direct dependency and the number of predecessors along the path. Even though systems in this class have proven its superiority both theoretically and in simulation, they have not been deployed in a large-scale real-world deployment.

In a different approach to tackle the problem of churn, a pull-based system establishes a mesh topology between peers to ensure that each peer has multiple paths to the source. Peers become partners with one another. To obtain video chunks, each peer periodically requests them from its partners. If a few connections are abruptly broken, a peer still can obtain chunks from other partners by requesting in subsequent periods. This reactive nature of peers in a pull-based system makes it adapting quickly to churn. However, to do so, peers need to exchange their buffer maps, which contain the current availability of chunks in their video buffers. In practice, pull-based systems have proven their robustness with various real-world deployments to date. Nevertheless, systems in this approach experience a high latency and a high signaling overhead due to chunk requests and the frequent exchange of buffer maps.

Hybrid systems combine the low latency and the low overhead of push-based systems with the demonstrated robustness to churn of pull-based ones. Normally, a hybrid systems bootstraps from a pull-based system. After that, peers push video chunks to its partners without requests to reduce the latency and the overhead. There are different strategies to do this. In early-push systems, peers start forwarding chunks immediately after the first few chunks are sent successfully. Meanwhile in adaptive-push systems, peers forward video chunks for a preset durations. Peers use longer durations for less dynamic scenarios and shorter ones for higher dynamics. This means that one or several implicit trees are established between peers receiving pushed chunks. The major drawback of the above two strategies is that they ignore the dynamics of peers. Due to their dynamics, frequently lost connections requires the recovery to maintain the connectivity of the overlay.

To increase the stability of hybrid systems, the selective-push strategy is developed. Peers only push chunks to stable peers that already stay sufficiently long in the system because they tend to stay even longer. The resulting backbone, consisting of only stable peers, is constructed to deliver the majority of chunks in the system.

Therefore, the latency and overhead are reduced while the backbone tree is more robust.

Despite the robustness to churn, pull-based and hybrid systems have two major vulnerabilities to DoS attacks. First, they are vulnerable to attacks on head nodes, which are the source's partners. These nodes connect the source and the rest of peers. When head nodes are attacks, the source is isolated, disrupting the stream delivery. More critically, head nodes can be identified from the information of exchanged buffer maps. An adversary can infer the system's overlay structure including head nodes. Second, the single-tree overlay backbone of hybrid systems is fragile. Attacks on the backbone can affect the chunk delivery of the whole system. More critically, the current self promotion strategy to identify stable peers is insecure. By design, it allows an adversary to identify backbone peers easily and early, enabling attacks on the systems.

In summary, it is challenging to construct an overlay resilient to both churn and attacks for P2P video streaming while maintaining a low latency and a low signaling overhead. Even though pull-based systems have proven their robustness to churn in reality, there are two open questions for the pull-based and its derivative the hybrid systems.

1. How to mitigate the DoS attacks on head nodes in the pull-based systems?
2. How to mitigate the DoS attacks on the backbone of the hybrid systems?

In the coming chapters, we present our methodology to tackle the above problems and our solutions for the resilience of P2P streaming systems against DoS attacks.

4

Modeling of P2P streaming systems

This chapter first provides a general model for P2P streaming. After that, the set of metrics for evaluating the resilience and the performance of P2P streaming systems is introduced and described. Subsequently, the chapter presents the churn model and the attacker models. Next, the chapter introduces the simulation model, that will be used as the main evaluation tool in this thesis. Consequently, a general-purpose simulation framework is developed to simulate P2P streaming systems. After presenting the validation and the performance results of the framework, we introduce common simulation settings, which are used in later chapters. Finally, we summarize the chapter.

4.1 General Model for P2P streaming systems

The topology of a P2P (live) streaming system is represented by a dynamic graph $G_t = (V_t, E_t)$ for $t \in [0, T]$, T being the *session length*. V_t denotes the set of online peers at time t, and E_t the edges between them. Nodes can join or leave the system at any time. The forwarding capabilities of a node v are limited by its *upload* and *download bandwidths* $bw_{up}(v)$ and $bw_{down}(v)$. The set of *neighbors* or *partners* of node v at time t is given by $N_t(v) = \{w \in V_t : \{v, w\} \in E_t\}$. The neighborhood is constantly adapted based on a *neighbor refinement scheme* R.

Content is distributed from a source node s, which remains online for the complete session, i.e., $s \in V_t$ for all $t \in [0, T]$. Multiple source nodes can be modeled by using one super node with their total upload capacity. The video stream is split into small chunks $c_1, \ldots, c_n$. At time t_i, s starts distributing the i-th *chunk* c_i to its partners. A peer v has to receive chunks on schedule to continuously play the video, i.e., the time

$t_i(v)$ at which v receives c_i has to be before the *play-out time* $T_i(v)$. If c_i is received, let $pre_i(v)$ denote the peer forwarding the chunk to v. Otherwise, set $pre_i(v) = \varnothing$. P2P streaming systems are classified according to the selection process for $pre_i(v)$.

Traditionally, content distribution in streaming systems is either push- or pull-based. In *push-based* systems, peers select one of their partners to receive a certain set of chunks in a publish-subscribe like manner. In the above terminology, the designated partner $pre_i(v)$ to receive chunk c_i from is defined shortly after join, and the assignment is only revised if the connection between v and the designated partner is severe. In that case, v immediately subscribes to a different partner to request c_i. The simplest such protocol is a *single-tree approach*: Here, the peers form a tree with the source as the root. Content is then pushed from the root to the leaves [8]. More complex and resilient *multiple-tree approaches*, e.g., [15, 27, 18], divide the chunks into k stripes $St_1 = \{c_1, c_{k+1}, \ldots\}$, $St_2 = \{c_2, c_{k+2}, \ldots\}$, $\ldots$, $St_k = \{c_k, c_{2k}, \ldots\}$. For each stripe, a separate dissemination tree across all peers is constructed explicitly and different stripe trees are preferably inner-node disjoint.

In *pull-based* systems [82, 49], each chunk has to be explicitly requested from a partner individually, i.e, v determines $pre_i(v)$ by asking a partner for c_i at the time it requires the chunk. To find peers that are already in possession of the required chunk, nodes exchange *buffer maps*, which list the chunks they can currently forward. The pull-based approach has received significant attention from the research community because of its simplicity and inherent resilience against churn. Due to these advantages, pull-based approaches also appear in various real-world deployments. However, the trade-off is significant overhead and increased distribution delays compared to push-based approaches.

Hybrid systems [80, 68] follow a compromised approach to overcome the disadvantages of both push-based and pull-based systems. A mesh topology becomes a bootstrap and underlying overlay for a hybrid system. Peers initially request missing video chunks from others as in a conventional pull-based system. However, selective connections between peers are later converted into the push-based form. Video chunks are therefore pushed on static connections. In this way, the hybrid system is resilient to churn with the mesh topology, while pushing chunks reduces the overall source-to-peer latency.

We use the general mathematical models of P2P streaming systems to define various metrics in the coming section and to formalize the solution in later chapters.

4.1.1 Metrics

To evaluate the resilience and the performance of P2P streaming systems, a set of metrics needs to be identified and defined. In the context of this thesis, we would like to assess the systems in three aspects: resilience, performance and cost as discussed in the previous chapter. Therefore, to quantify the resilience of the system, we

introduce three new metrics. They are the maximum miss ratio, the average miss ratio and the per-chunk miss ratio. Subsequently, we identify three metrics for the performance and the cost. They are the chunk miss ratio, sour-to-peer latency and the signaling overhead.

Please note, that the metrics we present in this chapter are used in all subsequent chapters of the this thesis. However, they cover only a subset of the all the metrics we use in this thesis. The rest of the metrics are presented later when they are used in the respective chapters. In the following, we describe in detail the definitions of the metrics commonly used in all subsequent chapters.

Chunk miss ratio We use chunk miss ratios as performance and resilience metrics instead of Quality-of-Experience metrics for several reasons that we will discuss shortly. In live streaming, timely delivery of video chunks is critical to ensure that the video stream can be played out smoothly. Therefore, each video chunk has its own play-out deadline for decoding. When a chunk arrives after its deadline it is considered missing. A missed chunk causes the video player to either stall or skip the chunk. In the former case, the smooth video play-out is not achieved. In the later case, the visual display of the video is impaired. Both cases reduce the perceived quality of the decoded video.

There are several methods to estimate the quality of a streaming system. Among them, the amount of missed chunks is one useful indicator because it tells how properly the system is working. It is also convenient since the calculation is straightforward. The disadvantage of this method is that it hardly reflects the quality of experience (QoE) from users' perspective. To quantify system's performance as perceived by users, studies in the literature [36] use QoE metrics, such as the Peak Signal-to-Noise Ratio (PSNR) which compares the decoded video at end users with the original one. However, this method has several drawbacks: First, calculating this metric is costly since it requires a considerable CPU power to decode the video. Second, the calculation depends on the video codec types and the benchmark video. It is therefore difficult to make general statements on the performance of a system.

From the above discussion, we select chunk miss to quantify system's performance for simplicity and flexibility. Specifically, we define the chunk miss ratio as the fraction of chunks that missed their play-out deadline divided by all chunks that should be played out. Note that the ratio is complementary to the Continuity Index (CI) [82] that is commonly used in the literature. The chunk miss ratio is favored in this paper because it is more intuitive to quantify the damages caused by the attackers to the system's performance.

The *chunk miss ratio* r_I of a node v is the fraction of chunks a node does not

receive with respect to a certain indexed chunk set $I = \{i_1, \ldots, i_j\}$, i.e.,

$$r_I(v) = \frac{|\{i \in I : pre_i(v) = \varnothing\}|}{|I|} \tag{4.1}$$

Thus, the chunk miss ratio $r_I(V)$ of set of V is defined as the average chunk miss ratio of all its nodes. The chunk miss ratio is directly related to the quality of service.

Furthermore, to comprehend the effects to the system after being attacked, we introduce three additional metrics to look at chunk miss ratios from three different perspectives:

- *Average Miss Ratio* which is the average miss ratio over a significant period of time after the attack.
- *Maximum Miss Ratio* which is the maximum instantaneous chunk miss ratio per second. It estimates the upper limit of damage that the attacks can cause to the system.
- *Per-chunk Miss Ratio* which is the fraction of peers missing a certain chunk. The metric quantifies how significantly missing a specific chunk can affect the system.

Source-to-peer latency The *source-to-peer latency* of a peer v denotes the average delay for receiving chunks averaged over all chunks in I, i.e.,

$$d(v) = \sum_{i \in I} \frac{|t_i(v) - t_i|}{|I|} \tag{4.2}$$

and $d(V)$ is obtained by averaging over all nodes in V. The delay is supposed to be small, especially for live streaming because users want to watch an event as close to its occurrence as possible.

Signaling overhead ratio The *signaling overhead ratio* gives the fraction of signaling traffic, generated by, e.g., the partner refinement, with regard to the overall traffic, the sum of signaling and content, i.e., for the number of content packets *cp* and signaling packets *sp* exchanged

$$sor = \frac{sp}{sp + cp} \tag{4.3}$$

The signaling overhead is supposed to be small in comparison to the delivered content.

4.1.2 Churn model

The inter-arrival time, defined as the duration between consecutive *open* events, is an often investigated measure in the area of IPTV systems. It characterized the expected workload on a streaming system. Veloso et al. found that the distribution of inter-arrival times in live streaming systems follows a Pareto distribution [63] while the session duration can be well represented by a Lognormal distribution. The Pareto distribution is a power law probability distribution that is found in many observable phenomena like, e.g., the number of social connections amongst individuals or the size of human settlements. The distribution is given by

$$f(x) = \frac{\beta \alpha^{\beta}}{x^{(\beta+1)}} \tag{4.4}$$

where shape α and location β determine its characteristics. Its cumulative distribution function is given by $F(x) = 1 - (\alpha/x)^{\beta}$. Whereas, the Lognormal distribution is given by the formula 4.5 as the following.

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}} \tag{4.5}$$

where σ is the scale parameter, μ is the location parameter.

4.1.3 Attacker Model

In this thesis, we consider Denial-of-Service (DoS) attacks. The ultimate goal of attackers is to prevent the video from being disseminated from peer to peer, to maximize the number of affected peers, who cannot obtain the video stream. Targets of the attacks can be the source node s or the peer nodes $v_a \in V_t$. When s is attacked, the whole P2P streaming system is collapsed since no peer receives the video stream any more. Since we are interested in finding methods to organize peers in a P2P streaming system, we assume that s is well protected and is exempt from attacks.

Due to limited resources, the attacker can only shut down a certain number of nodes. To maximize the damage to the system, the attacker has to apply a strategy which specifies which peers and in which order to attack. The process of an attack consists of two stages: *vulnerability analysis* and *attack strategy selection* as illustrated in Figure 4.1. In the first stage the attacker collects and analyzes the information to identify possible weakness in the system, which can be a set of nodes that have a key role in distributing video chunks. The sudden absence of those nodes can disrupt the stream delivery to a large number of other peers, thus, severely degrading the system performance. The second phase is to select an attack order, given the list of possible vulnerable target nodes. The ultimate goal of the process is to maximize the destructive impact of the attacks. Failure is a special case of attacker when nodes are randomly selected.

FIGURE 4.1: A two-stage process of an attack

We assume that A is able to shut down arbitrary nodes except the source within the system, but is limited with regard to the number of nodes x, called the *attacker budget*, it can shut down at any given time. Furthermore, A can only perform one attack of concurrent shut downs every t seconds. Its goal is to shut down the most relevant nodes, such that the chunk miss ratio as defined in Eq. 4.1 is maximized for some period after the attack.

We consider both practical and theoretical DoS attacks. They differ from each other in the assumption on their capability to collect information about the system. In a practical attack, the attacker needs to participate in the system and follow the system's protocol to collect information. This attack allows us to understand the consequence of an attack that can happen in a real-world system. On the other hand, we assume that theoretical attacker can have a global knowledge on the whole system. This attack type allows us know the limit of the system, even under attacks of a very powerful attacker. Both aspects bring together a more comprehensive understanding on the system's resilience under DoS attacks. In the following we describe the two attack types in more detail.

Practical attacker model

This attacker model is applied for pull-based and hybrid systems where peers exchange buffer maps with one another. In the analysis phase, the attacker collects buffer maps from participating peers to infer the implicit overlay structure of the system. The attacker can collect BMs by requesting directly active peers in the system. This operation can be allowed in a P2P streaming system as part of peers' normal behavior or as special service to monitor the system's performance. Even if a direct request of BM is not feasible, an adversary still can collect them via regular exchanged buffer maps. In this case, the attackers or compromised peers join the system, establish partnership connections with benign peers and report the collected information to a centralized external attacker for analyzing. In either case, the adversary can obtain buffer maps from participating peers.

The collected BMs are then analyzed, for example by using the techniques described by Hei et. at [32]. The resulting list of potential head nodes are then used as the input for the second phase, which is attack strategy selection. The attacker then selects the top A nodes among those head nodes to shut them down at once.

Theoretical attack model

Assuming a global knowledge on the system, the theoretical attacker collects all the chunks sent from one peer to another. A similar attack model with the greedy strategy has been used in a previous work by Grau [30]. Finding an optimal attack has been shown to be an NP-complete problem in general. However, Brinkmeier et al. showed that for optimally stable multiple-tree topologies, the presented greedy attacker corresponds to the optimal attacker [15]. We hence adapt the greedy attacker for our scheme, which makes use of optimal stable topologies for the backbone.

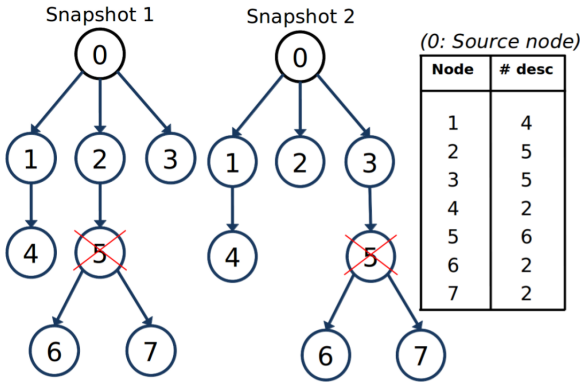


FIGURE 4.2: The greedy attacker ceases node 5 which has the highest number of descendants calculated from the two snapshots

More precisely, the attack strategy is the following: Note that for each chunk c_i, a dissemination tree g_i can be constructed by A due to its global knowledge. The tree g_i contains an edge from a peer v to a peer u if v received c_i from u. Let $desc_i(v)$ denote the *descendants* of v in g_i, i.e., all nodes u such that the path from u to the source s in g_i contains v. All nodes in $desc_i(v)$ would have to receive c_i via a different longer path if v was not present. Hence, the shutdown of v requires explicit (in push-based systems) or implicit (in pull-based systems) reorganization of all nodes in $desc_i(v)$ of the dissemination. As a consequence, considering the number of descendants in all dissemination trees can be applied to approximate the relevance, i.e., for the set I of indices, the relevance of v is given by

$$h(v) = \sum_{i \in I} desc_i(v). \tag{4.6}$$

The set I is usually chosen to consist of the last chunks distributed shortly before the attack. The attacker A then determines the node v_1 with the maximal relevance. Afterwards, A constructs the trees g_i^1 for all $i \in I$ by removing v and all its descendants from g_i. The second node v_2 is then determined as the most relevant node considering the trees g_i^1. In this manner, A iteratively obtains the nodes $v_1, \ldots, v_x$ to shut down during the attack.

We illustrate the description of the greedy attacker by an example in Figure 4.2. There are two snapshots, the number of descendants *desc* of nodes are calculated up on, and given in the table on the right. It should be noted that the nodes which directly connect to the source (or so-called head nodes) do not always have the highest *desc*. For example in the Figure 2, node 5 has the highest *desc* and it is attacked by the described greedy attacker.

4.2 Simulation Model

The complexity of the a sophisticated analytical model to investigate the resilience of P2P streaming drives us to use simulation. The method provides us a reasonable compromise between the tractability and accuracy requirements.

Over the past years, a multitude of different overlay streaming systems have been proposed. Most of them, however, have been evaluated individually. The lack of a common simulation framework makes it difficult to compare the properties of the different systems with each other, hindering us to provide a meaningful conclusion. The need for selecting the best system under the same condition of workload and network dynamics motivates us to provide our own simulation framework. Specifically, we introduce OSSim (or Overlay Streaming Simulator), a general-purpose simulation framework that allows the instantiation of different overlay streaming protocols. The framework provides a generic and modular structure, and several membership management as well as overlay streaming protocols. In the following we will describe and discuss our OSSim simulation framework.

4.2.1 Requirements

To evaluate the performance and resilience of overlay streaming systems in a fair and comparable way, we identify the need for a flexible simulation framework. We consequently provide a simulation framework. It comprises of abstract components that relate to the general functionality needed within all these systems. Additionally, the components are simple to adapt to specifications of various systems. The requirements for such a simulation framework can be summarized as follows:

R1) **Validity:** Its modules are verified for correctness.

R2) **Generic design:** It enables to simulate a large set of overlay streaming systems. Therefore, its design should not be tightly coupled with a specific system. Common functionalities have to be modeled by generic modules, instead.

R3) **Flexibility:** The framework contains interchangeable and configurable modules to harbor variety of protocols with the least additional modifications.

R4) **Extensibility:** The module interfaces need to be well defined to derive new modules easily. This consequently facilitates the simulation of different overlay streaming protocols.

R5) **Minimum Overhead:** Given a P2P streaming system, the framework should be able to simulate it with negligible overhead. It should also consume a reasonable amount of resources (e.g., CPU, RAM, and storage) when simulating a large number of nodes in a reasonable amount of time.

R6) **Underlying Network Modeling:** The underlying network model should be exchangeable. On the one hand a fully configurable network topology with realistic bandwidths, packet delays and packet losses should be provided. On the other hand there should be fast and simple alternative models for simulations with a large number of nodes.

4.2.2 Related work

Various simulators are developed to simulate P2P systems in general and specific P2P video streaming systems. This section surveys the most remarkable ones and classifies them into general purpose and tailor-made simulators.

General purpose Simulators: *OverSim* [9] is developed to simulate Peer-to-Peer systems in general. OverSim's design is tailored to the P2P systems with addressing and routing functionalities. It offers a rich implementation library of both structured and unstructured P2P systems. Additionally, some OverSim's extensions are developed to simulate particular P2P video streaming systems. Examples are *Denacast* [55] to simulate pull-based systems and *Layeredcast* [47] for hybrid systems. OverSim and its extensions have several drawbacks. (*i*) The design of OverSim does not consider P2P video streaming systems. It lacks video manipulation functions, such as a video buffer and video player. Even though it is possible to modify and adapt *OverSim* to simulate P2P video streaming, it requires significant modifications from the original design. This would also considerably enlarge the code base of *OverSim*, which would violate the minimal overhead requirement. (*ii*) The implementations of OverSim's extension are too specific. They are developed independently focusing on very particular systems without an outlook to cover other streaming classes. Furthermore, their implementations APIs are incompatible to directly integrate them into a common and more general library.

Tailor-made Simulators: The first simulator in this class is *p2pstrmsim* [79]. It was developed to study the performance of pull-based and hybrid push-pull systems. The simplified underlay network (i.e. without the TCP/IP protocol stack) allows to simulate tens of thousands of nodes within reasonable time. Other examples are *p2ptvsim* [11] and *SSSim* [1]. *p2ptvsim* is an event-driven simulator, and abstracts the underlay network by a module that is responsible for calculating transmission

delays of video packets. The other simulator, *SSSim*, is a round-based because it assumes that peers are synchronized. Therefore, the simulator compromises on accuracy. Moreover, internal states of peers are available globally for access. This design decision limits the extensibility of the simulator since it potentially produces conflicts when the code basis grows. The tailored designs of this class of simulators, however, hinder the possibility to extend them towards a general purpose framework.

To sum up, there is no generic and extensible simulation framework for P2P video streaming so far that allows to compare different classes of overlay streaming systems in a fair manner. Nonetheless, there is still a need for selecting the best system under the same condition of workload and network dynamics.

4.2.3 OSSim Framework

The lack of a generic simulation framework prevent us from providing a meaningful conclusion when comparing different P2P streaming systems with one another. For this reason, we develop OSSim, our own simulation framework. The task, however, is not straightforward. On the one hand, the framework has to harbor various classes of overlay streaming systems. On the other hand, it must be generic enough to produce common functionality modules to avoid implementing them again in different systems. In this section we first present the ideas of a generic simulation framework for P2P streaming systems. Then the overall architecture of the framework is introduced. After that, the specification and implementation of the OSSim including per-peer and global components are discussed.

Ideas of a generic simulation framework

The comprehensive background on various P2P video streaming systems presented in Chapter 2 provides us an insight to design OSSim, our generic simulation framework. The key idea is to discover and group common functionalities between the various P2P streaming classes. Indeed, they share various common operations. Every peer requires two functionalities. First, the one involving video streams includes generating, buffering and playing out video chunks. The second one relates to bootstrapping and membership operations which are very similar between systems. From the system perspective, the simulation of a P2P streaming system needs to coordinate peer' joining and leaving and monitor the list of active (or online) peers. Within a single system, the behavior of the source and peers only differs slightly, because the former generate the video stream whereas the latter consume video chunks.

The above common functionalities between P2P streaming systems and between a single system's source and peers suggest a modular design for OSSim. Global modules are shared by all peers and the source with in a system for coordinating and monitoring peers. Per-peer modules are dedicated for each peer or the source.

Common per-peer ones should be reused if their functionalities are common in various P2P streaming systems. Furthermore, to ensure the reusability of modules in different systems, the module's interface should be carefully specified. Those ideas lead us to OSSim's architecture with a modular design which is detailed in the next section.

OSSim Architecture

This section describes in detail the design of OSSim, which consists of per-peer modules and global ones as illustrated in Figure 4.3. Per-peer modules are categorized into two layers for clarity. The *Discovery Service* layer comprises of modules that are responsible for bootstrapping and membership management. Modules in the second layer, *Streaming*, are further divided into two sub-layers: The *Topology Management* harbors functionality, such as the topology optimization for push-based systems and the partnership management and the chunk scheduling for pull-based systems. In the *Media Management* sub-layer, video packets are generated at the *Packet Generator*, stored and forwarded at the the *Video Buffer*, and consumed at the *Player*. To connect various per-peer modules, this architecture requires a *Dispatcher*. The module serves as an unique interface of a peer. At the outbound direction, the module forwards messages from per-peer modules to other peers. At the inbound direction, it distinguishes messages from other peers and sends them to appropriate per-peer modules.

Additionally, the framework also consists of global modules such as the *Active Peer Table* to keep track of the currently active peers, the *Churn Coordinator* to coordinate peers' joining and leaving operations; *Statistic Collector* to collect and calculate system-wide statistics, and finally the *Underlay Topology* to manage the topology of the underlying communications network.

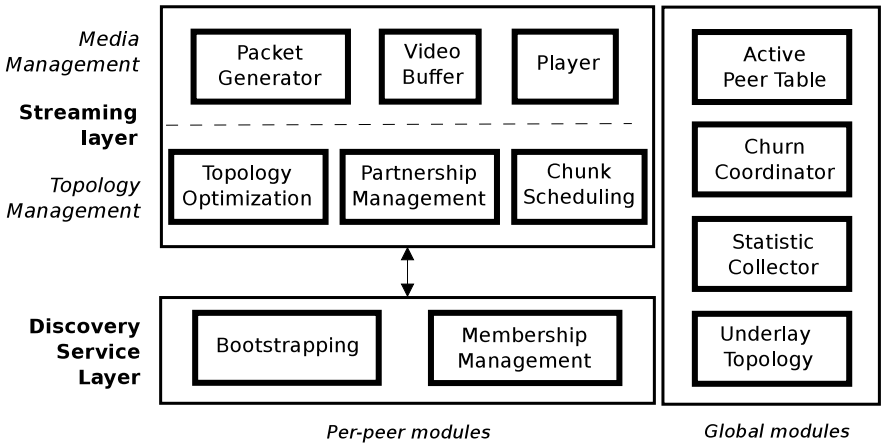


FIGURE 4.3: Architecture of the OSSim framework

Specification and Implementation

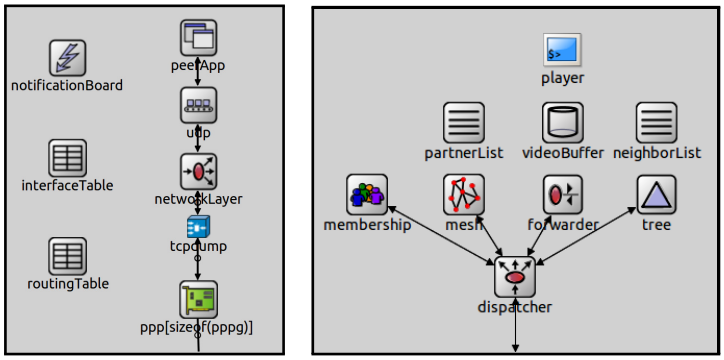
There are two approaches to realize OSSim with the architecture described in the previous section: leveraging a generic simulation package or not. Using a simulation package requires additional efforts to learn its architecture and usage in order to build our simulator on top of that. However, a generic simulation package, especially a publicly available one, provides several advantages, such as built-in simulation libraries especially the random number generator, visualization and statistical processing tools etc. Due to those advantages we follow this approach to implement OSSim on top of a simulation package.

The next question is to select one among simulation packages. Two prominent choices are available, which are NS-3 [53] and OMNeT++ [62]. Both packages are very comparable in various aspects. First, they have a modular design and provide various libraries such as random number generator and networking functionalities. Second, they both use C++ to implement modules. Third, according a comparison study conducted by Weingartner et al. [69], both of them are capable of simulating large networks in an efficient way. However, compared to NS-3, OMNeT++ has a slightly better support with a dedicated IDE for development and simulation, plus a visualization tool that is especially useful for debugging. After attempting both packages, OMNeT++ is subjectively selected to implement OSSim. Subsequently, OSSim also uses the INET model [61], which extends OMNeT++ to simulate TCP/IP networks. In the following, we describe the realization of the OSSim's architecture in OMNeT++.

Per-peer modules are mapped into OMNeT++/INET modules as follows. The whole functionality of a peer (or a source) is resembled within the peerApp module which runs on top of UDP. Figure 4.4(a) illustrates a node which follows the ISO/OSI model and bases solely on the UDP Transport layer. We simplify the StandardHost in the INET framework by removing TCP and TCPApp modules, and extend the UDPApp module by our own application module. We then model the streaming application at a peer node as illustrated in Figure 4.4(b). First of all, a generic peer application consists of a *dispatcher* module for message classification. This module acts as a gateway between the lower Transport layer which is UDP in our framework and functional modules belonging to our layers as illustrated in Figure 4.4. Even though there would be overhead due to the operation of the *dispatcher*, this design makes the model more flexible when it has to adapt to, or include other Transport layers such as TCP. The dispatcher connects various modules implementing the functionalities of the two layers at each peer.

Discovery Service Layer At the Discovery Service layer, the *membership* module implements the membership management and bootstrapping functionalities.

The main purpose of this layer is to provide the peer sampling service for the



(a) (b)

FIGURE 4.4: Simulation models of (a) a peer node and (b) a peer application module in OSSim

upper streaming layer. By maintaining a partial and frequently refreshed view on the active peers, it should provide a list of active peers on demand. In tree-push systems, it bootstraps a newly joining peer to subscribe to the overlay. In pull-based systems, it also assists peers to find new partners besides the bootstrapping functionality. The central operation of the this layer has the following API:

getARandPeer(): returns a random peer address from the set of active peers in the system

Deployed systems, recently, realize this service by gossiping protocols since they are lightweight, scalable, and resilient against network dynamics. In event-driven simulations, however, those protocols slow down the execution time significantly because gossiping peers exchange many messages with each other. Therefore, we introduce a *Dummy* module to reduce simulation time, especially in the development phase. While the *Dummy* module resembles the same API, it keeps and updates a centralized data structure of active peers. Even though this is unrealistic, it speeds up the simulation significantly.

Topology Management Layer The *mesh* and *tree* modules belong to the Topology Management sub-layer within the Streaming layer. The former requires the *partnerList* module to manage partnership connections between peers. The latter uses the *neighborList* module to manages the list of parent and child nodes and perform topology optimization operations.

Partnership Management: This is the core of pull-based and hybrid systems. It is in charge of joining the streaming overlay, handling external messages and timers, sending periodic messages (e.g., Buffer Maps), and activating other modules such as *Chunk Scheduling* and *Player*.

Chunk Scheduling: The requesting part of this functionality is activated periodically. It depends on specific algorithms (e.g., *Random* or *Rarest-First*) applied in

the overlay streaming protocol. The direct outcome of this functionality is a set of messages to request video packets. On the other hand, the responding part simply waits for request messages from other peers and replies accordingly.

Topology Optimizer: This module is the core of a tree-push system. It establishes and maintains the parent-children relations with other peers. It also sends *Heartbeat* messages to parent nodes to assist the failure recovery process. One important functionality of this module is to switch parents to improve the QoS of individual peers and to optimize the overall topology.

Streaming Management Layer At the Media Management sub-layer, the Video Buffer and the Player modules are implemented by the *videoBuffer* and *player* modules respectively. The *forwarder* acts as a communication interface between the videoBuffer and the dispatcher modules. The model of a streaming source follows a similar design, except that a packet generator replaces the player module. Other differences in the behavior of the video server remain in small implementation details only.

This layer consists of *Video Buffer*, *Forwarder*, *Packet Generator*, and *Player* modules.

Video Buffer: It stores received video packets. This implies a sliding window whose size equals the size of the buffer (in video packets). Since the sliding operation is executed quite often, the *Video Buffer* should be carefully designed and implemented. We realize it by a cyclic array containing video packets. Deciding where to insert video packets into the buffer will be done by modulo operations on packet identifiers. This avoids sliding operations, and consequently improves the overall performance of the framework. The *Video Buffer* module provides the following services:

isInBuffer(): checks whether a packet is in the *Video Buffer*

getPercentFill(): returns the percentage of available packets in the *Video Buffer*

Forwarder: It is the gateway of the *Media Management* sub-layer. The *Forwarder* receives video packets and stores them in the *Video Buffer*. It also picks up video packets from the *Video Buffer*, and sends them to other partners upon request. The *Forwarder* provides the following main service:

sendVideoPacket(): sends a video packet specified by its sequence number to a remote peer

Packet Generator: This module is active at streaming servers only. It periodically generates video packets, and then inserts them to the *Video Buffer* of a streaming server.

Player: This module has a play-out pointer that specifies the currently required video packet. Based on the pointer, the player periodically obtains from the *Video*

Buffer the video packet. The pointer moves forward consistently with the play-out rate. To maintain a smooth play-out in live streaming, each packet should be available before a strict deadline, or it must be ignored by the player, otherwise. In the context of live streaming, it is more critical that the player progresses with the same pace as the generated stream at the source. Therefore, we implement the *Simple_Skip* player. It starts playing out stored chunks after an initial period of buffering and it simply moves its pointer forward continuously and skips all unavailable packets. This module is characterized by two parameters, namely the percentage of buffering period as a fraction of the total buffer size and the fraction of filled buffer with received chunks. The player's selective API is following:

activate(): activates the player

stopPlayer(): stops the player

getPlayoutPoint(): returns the current play-out point

getPlayerState(): returns the current state of the player

To correctly and efficiently forward messages to different modules, we introduce a message hierarchy as illustrated in Figure 4.5. Messages belonging to modules of the same layer or sub-layer in Figure 4.3 are grouped together using the same group identifier. When a message of a particular group is forwarded to the respective layer, it is up to the layer to either process the message directly or forward it to a subsequent module of the same layer.

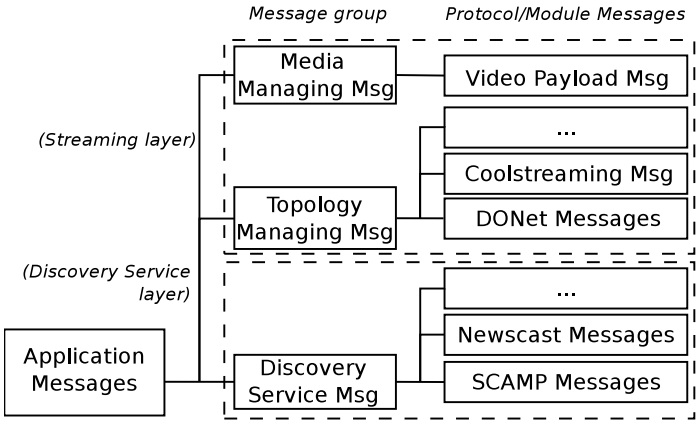


FIGURE 4.5: Hierarchical structure of application messages with their realization in specific membership and streaming protocols

Global modules

In addition to per-peer modules, global ones such as *Active Peer Table*, *Churn Coordinator*, *Streaming Configurator* and *Statistics Collector* are necessary to set up,

coordinate a simulation, and to help collect simulation results. These modules are illustrated in Figure 4.6

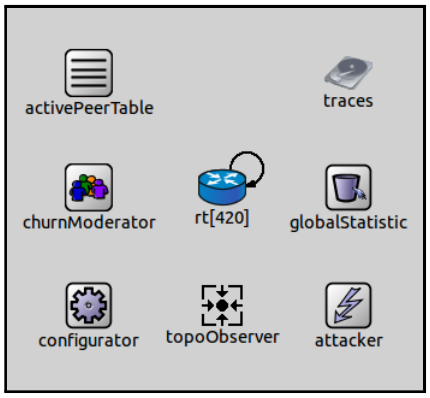


FIGURE 4.6: Network-level model with helper modules

Churn Coordinator: This module calculates and provides join and leave times of peers upon request. To scramble the order of peers joining the network, we apply a two-phase arrival assignment process. In the first phase, peers are assigned at random order. In the second phase, peers actually query the *Churn Coordinator* to get their arrival and departure times. It is possible to extend this module to harbor additional actions of peer such as switching between channels. The selective API of this module includes:

getJoinTime(): To get joining time

getDepartureTime(): To get departure time

Statistics Collector: This module collects data from all peers, and calculates system-wide metrics, such as chunk miss ratios and signaling overhead.

Underlying Topology: The underlying topology decides how routers are connected to one another and how peers are attached to those routers. This module should resemble the characteristics of the Internet core network for realistic simulation studies. By default, the flat model of the Internet while in reality Internet is better modeled by a hierarchical one. Transit-Stub is one of frequently used model to characterize the hierarchical structure of the Internet [17]. It comprises of interconnected edge routers to form stub domains. They are in turn inter-connected via core routers that form transit domains. Figure4.7 illustrates such a topology. OSSim supports both of these underlying topologies.

Summary

To sum up, we already presented the generic modules and their selective APIs. We instantiated those modules for specific protocols. Key instantiations and their

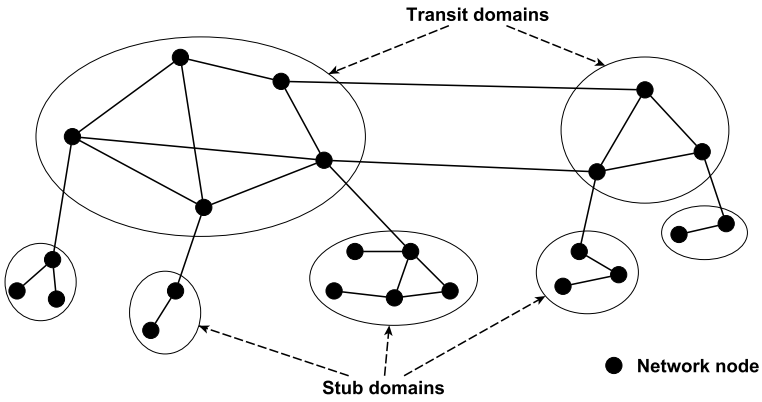


FIGURE 4.7: Example of the Internet's transit-stub model: several Stub domains are inter-connected by Transit domains (adapted from Calvert [17])

parameters are summarized in Table 4.1.

Compared to the requirements presented in Section 4.2.1, our generic framework is flexible to harbor different overlay streaming and membership management protocols. Moreover, the layered and modular design also eases the extension of the framework using configurable components. The follow-up question is whether the simulation model correctly implements the systems of interest. We answer the question by conducting experiment to validate our models. Afterwards, we measure the

Table 4.1: Selective instantiations and their parameters

Component	**Instantiation**	**Parameters**
Churn Coordinator	*Uniform*	lower and upper bounds
	Exponential	rate
	Pareto	scale and shape
	Log-normal	scale and shape
Membership Management	*SCAMP*	duplication factor
	Newscast	cache size
	Dummy	-
Topology Management	*DONet*	*(various)*
	OST	*(various)*
	mTreebone	*(various)*
Chunk Scheduling	*Rarest First*	*(various)*
	Random	*(various)*
Player	*Simple_Skip*	percent fill to start
	Skip_Stall	*(various)*

performance of OSSim. We present them shortly in the coming sections.

4.2.4 Validating OSSim

In this section, we describe different experiments for the validation of our simulation model. The section aims to verify that our simulation model is correctly implemented and to validate that the model properly resembles the behavior of the simulated systems. Towards this end we carefully verify individual modules for each layer separately. Moreover, combinations of those modules in streaming protocols are also wholly validated. For each of the three classes of P2P video streaming systems, we select one representative system using three criteria. First, the systems are either representative or state-of-the-art in their class. Second, they are used in our comparison in this thesis. And third, system's descriptions and experiment set-ups are well described for their sensible implementations and repeating experiments. Specifically, we validate *DONet*, *OST*, and *mTreebone* for the pull-based, push-based and hybrid protocols respectively. Their published results are used in comparison and validation.

Validating DONet We conducted several experiments with the *DONet* implementation in *OSSim* to ensure that the simulation model of DONet is correct. The published results of DONet in [82] are used in the validation. Among various experiments in the paper, we select the ones involving the Continuity Index (CI) and the signaling overhead since the set-ups allow us to repeat the experiments without significant effort. None of the single metrics is sufficient since one can always obtain high CI with high overhead, e.g., by sending Buffer Maps more frequently and vice versa. Therefore, both metrics are selected for the validation.

Follow the settings in [82], the streaming source has its upload bandwidth of 2 Mbps first joins the streaming overlay. Then, 200 homogeneous peers with their upload bandwidth of 1200 kbps join the overlay. Their inter-arrival times follow a uniform distribution within the range [0, 60] seconds. They do not leave the overlay until the end of the simulation. The simulation duration is 6000 seconds. Summary of the key parameters for validating the implementation of DONet in our OSSim framework are shown in Table 4.2. All simulation results are averaged over 30 runs.

In the first experiment, we varied the streaming rate between 100, 200, 300, 400, and 500 kbps. We expect that CI decreases as the streaming rate increases given the same upload of peers. Figure 4.8 plots the CI metric with respect to different streaming rates.

As can be seen in Figure 4.8, the CI decreases gradually when the streaming rate increases from 100 kbps to 500 kbps. The CI also reveals the same trend as in the published results in [82] whose experiments were conducted in a practical

Table 4.2: Key parameters and their values for the validation of the DONet implementation in OSSim

Parameters	**Values**
Streaming source's upload bandwidth	2 Mbps
Peers' upload bandwidth	1200 kbps
Video packet size	5000 B
Video Buffer capacity	60 s

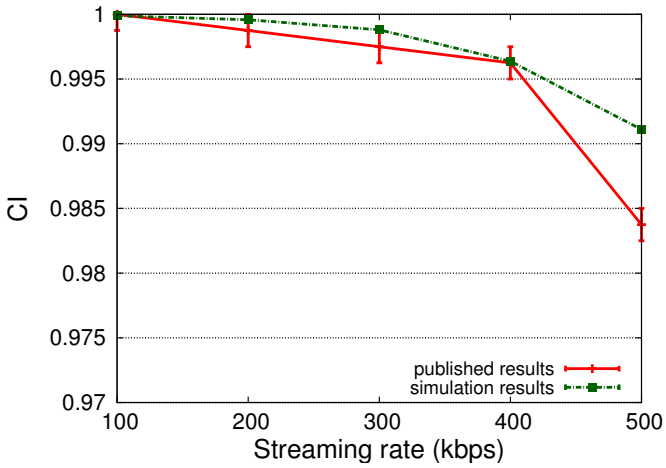


FIGURE 4.8: Validation of the DONet streaming protocol with published results in [82]: The Continuity Index as a function of the streaming rate

deployment. Both results are relatively close to each other.

In the second experiment, we measure the overhead when the number of partners varied 2, 3, 4, 5, and 6 and expect that the overhead increases respectively. Figure 4.9 plots the overhead with respect to different number of partners. As can be seen in Figure 4.9, the overhead increases gradually when the number of partners increases from 2 to 6. Moreover, our results agree closely with the published results in [82].

Validating Optimal Stable Topology (OST) Several metrics are required to validate the Optimal Stable Topology (OST) scheme because it constructs a topology consisting of multiple trees with specific characteristics. On the one hand, the trees should avoid very long branches which decrease the the performance and increases the dependence to inner nodes. On the other hand, the topology should strive to satisfy the inner-node disjointness property. Particularly, the three following metrics are used:

1. The *maximum tree height* which is the maximum height of all the trees.

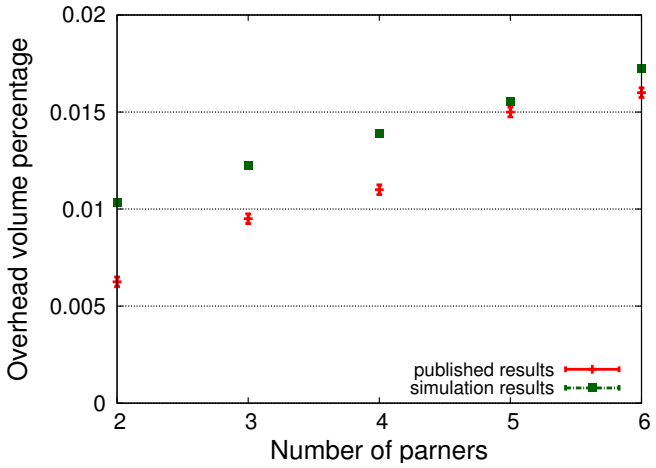


FIGURE 4.9: Validation of the DONet streaming protocol with published results in [82]: Signaling overhead as a function of the number of partners

2. The *fraction of nodes forwarding in more than one tree* which depicts how optimized the topology is. The more number of trees the nodes is forwarding the higher negative impact it has on the system's performance when the node fails or leave.

3. The *mean number of successors of head nodes* (or the child nodes of the source) which measures the average number of all successors starting from the head nodes in all trees.

To validate our implementation of the OST scheme, we repeat the experiments described in [14]. Let the upload capacity $c(v)$ denote the relative upload bandwidth of a peer v to the streaming rate. All nodes have their capacity $c(v) = 2$ while the source has an upload capacity of $c(s) = 4$. A total of 250 nodes join the system with their arrival times are uniformly distributed between 0 and 100 seconds and do not leave the system until the end of the simulation. Snapshots of the topology are taken at the $120th$ second when the topology optimization is completed. The number of stripes was fixed to $k = 4$.

We can reproduce very similar results as described in [14]. First, a maximum tree height of 8 is recorded which equals exactly the published result. Furthermore, the trees have an average height of 4.9 and a minimum height of 4. A tree with the height of 8 occurred only once in 30 runs. Second, in average 6% of the nodes forward chunks in more than one tree, thus are inner nodes in more than one tree. In addition to that, the fraction of nodes forwarding in more than one tree equals to 3% in the best case and 11% in the worst case. Third, mean number of successors of head nodes equals to 8 which is exactly the reported value in [14]. The above results are summarized in Table 4.3.

Table 4.3: Comparison of the published results in [14] and our results.

Metric	**Ours**	**[14]**
Max. tree height	8	8
Fraction of nodes forwarding in more than one tree	6%	6%
Mean number of successors of head nodes	8	8

Validating mTreebone Finally, we conduct an experiment with the *mTreebone* implementation in *OSSim*. Its implementation reuses one module running the pull-based scheme for bootstrapping. Additionally, it uses another module running the push-based scheme for constructing the backbone overlay. To quantify the results, we select the *Chunk miss ratio* metric. The parameters are set to the values in the mTreebone paper [68]. Specifically, the session length is set to 6000 seconds. There are 5000 overlay nodes. The peers' upload bandwidth is uniformly distributed between 4 and 12 times of the video stream bit-rate. The backbone is formed when peers' online period exceeds a threshold of 30% of the remaining duration of the streaming session.

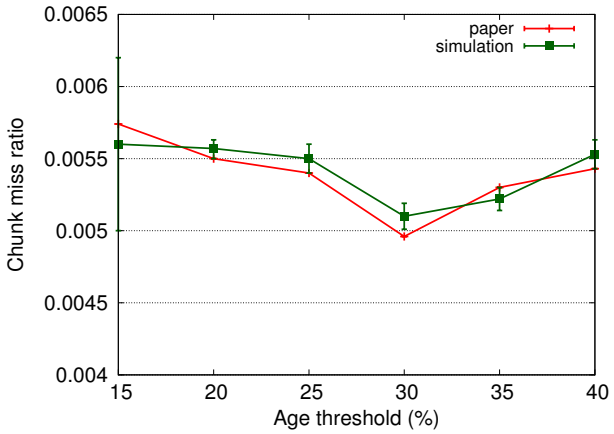


FIGURE 4.10: Validation of the mTreebone streaming protocol with published results in [68]: Chunk miss ratio as a function of the age threshold

Figure 4.10 plots the dependance of the chunk miss ratio to the age threshold to promote a node to the backbone. The figure shows that the chunk miss ratio in simulation matches rather well that of the paper. Both plots create a minimum when the age threshold equals to 30 %. The largest discrepancy occurs when the age threshold equals 25 %. The difference is roughly 5% which is can be neglected.

4.2.5 Performance of OSSim

In this section, we assess the performance of OSSim in simulating different P2P streaming systems. Because the simulation do not generate data except the simulation results, storage is not considered. Similarly, we do not consider network bandwidth since the simulation processes do not exchange information with one another. Instead, we select the memory consumption and the running duration (wall-clock time) to measure the performance of OSSim.

We select DONet, OST and mTreebone as the representatives for each class of P2P streaming systems. Thus, their simulation results provide us a comprehensive view on OSSim's performance. The simulation duration in all experiments is 5400 seconds. We vary the number of peers in each simulation between 500 and 3000. The results from 30 runs are averaged and confidence intervals of 90% are calculated.

Memory consumption We expect that the amount of consumed memory increases linearly with the number of peers because they are limited in the number of overlay connections they can have with one another. Therefore, an increase number of peers only causes a linear increase in the number of connections and subsequently the number of exchanged messages.

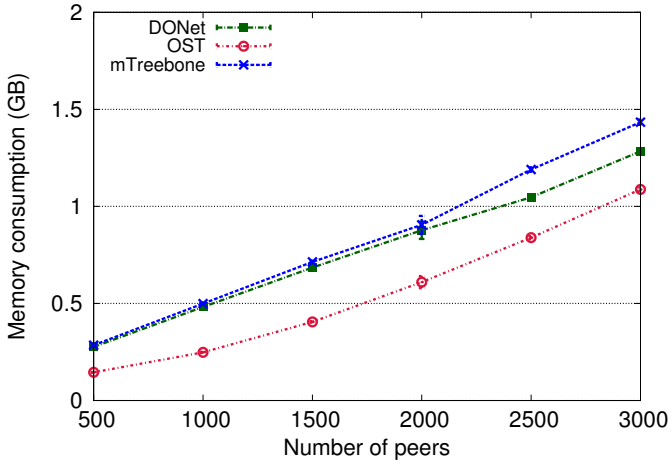


FIGURE 4.11: Memory consumption in OSSim

Figure 4.11 plot the amount of memory consumption with regard to the number of peers in each simulation. As the number of peers increases from 500 to 3000, the memory consumption of DONet and mTreebone increases from around 0.3 GB to almost 1.3 GB and 1.5 GB respectively. OST's memory consumption increases from around 0.2 GB to around 1.1 GB. Among three schemes, OST consumes the least memory while mTreebone consumes slightly more memory than DONet. This is reasonable because DONet is a pull-based system, in which peers frequently exchange

buffer maps and request video chunks. The hybrid scheme mTreebone combines both pull-based and push-based schemes into one system, therefore it consumes the most memory. More importantly, the memory consumption increases linearly with the number of peers. This means that OSSim can simulate larger networks with a reasonable amount of memory.

Running durations We also measure the running duration (wall-clock time) of the simulation. For a similar reason to the above experiment, we also expect that the running duration increases linearly with the number of peers.

The durations for DONet, OST and mTreebone are plot in Figure 4.12. The figure show that the running duration of OST increase linearly as the number of peers increases from 500 to 3000. In all cases, its durations remain well under one hour. The running durations of DONet and mTreebone increase from around one hour to around four hours. More specifically, their durations increase drastically when the number of peers increases from 500 to 2000. Afterwards, the their durations only increase slightly. Comparing the simulation of OST with DONet, the running durations of the latter are around 5 times that of the former. This can be an issue when simulating very large networks.

Summary The results suggest that, OSSim allows for large-scale simulation of OST, a multiple-tree push-based system. We project that similar conclusion can be valid for the simulation of other multiple-tree push-based systems.

Simulating a pull-based system or a hybrid one would require a reasonable amount of memory. However, for significantly larger networks, the running duration would be a main issue.

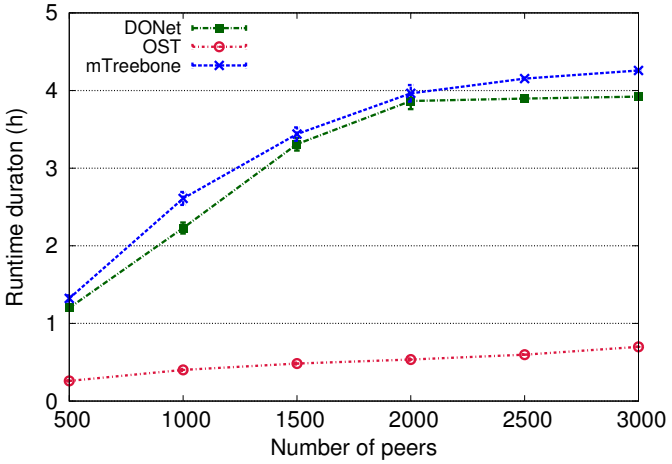


FIGURE 4.12: Simulation duration in OSSim

4.3 Common simulation settings

Our simulation framework OSSim is used in this thesis to implement and evaluate various P2P video streaming systems. For sensible comparisons between streaming systems, common settings are required, which are introduced shortly.

Underlying networks: As discussed in Section 4.2.3, to generate the transit-stub topology for the underlying Internet, we use the GT-ITM [78] topology generator. The resulting topology consists of 20 core and 400 edge routers that are interconnected by 1212 links. In particular, we use the following parameters for the topology generator: diameter 14, node degree 2.843, and path length 6.231. The latencies in the links connecting the routers are uniformly distributed in the range [1, 60] ms. The topology is generated once and used in every simulation study. In later set-ups, peers are randomly attached to the edge routers at the beginning of each simulation.

Workload: We use the synthetic churn model reported by Veloso et al. [63]. From this model, the distributions of inter-arrival times of users and session durations are Pareto and Lognormal respectively. We also use the parameters discovered by the authors, specifically $a = 2.52$ and $b = 1.55$ for the Pareto distribution and $\mu = 1.44$ and $\sigma = 5.19$ for the Lognormal one. In addition, we allow leaving peers to rejoin the system after a random period to maintain a rather stable peer population.

4.4 Summary

In this chapter, we first introduce the general model for P2P streaming systems. The model consists of the set notions to describe the parameters of P2P streaming systems. It also characterizes the fundamental aspects of a push-based and a pull-based system. These are the building blocks to describe more sophisticated systems, such as a hybrid one. Furthermore, the chapter defines a set of metrics to assess the resilience and the performance of the system. Specifically, we define source-to-peer latency and the signaling overhead as the two performance metrics. In addition, we use the chunk miss ratio to measure the damage that attacks cause to the system. Subsequently, to measure the impact to the system after the attacks, we define the maximum miss ratio, the average miss ratio and the per-chunk miss ratio to provide a comprehensive view on the damage.

The complexity of the mathematical models for different classes of P2P streaming systems would make a formal analysis impractical. Therefore, in the second part of the chapter, we focus on the simulation as the main evaluation method in this

thesis. The desired simulation tool needs to enable a fair comparison of different P2P streaming systems.

Since such a generic simulation framework supporting all the three classes of P2P streaming systems is unavailable, we have developed *OSSim*, a general-purpose simulation framework in OMNeT++ to simulate P2P streaming systems. OSSim is characterized by a modular and generic layered design that we derived from the analysis of the three major overlay streaming classes. This design eases the implementation of additional overlay streaming protocols. The simulation incorporates synthetic churn models and allows for the transit-stub model of the underlying network which resemble more closely the characteristics of the Internet.

Using this framework, we implement representative streaming protocols, namely DONet, OST and mTreebone for the three classes of P2P streaming systems, i.e. pull-based, push-based, and hybrid ones, respectively. Our simulation results indicate the validity and accuracy of our framework for the implemented protocols. Furthermore, we assess the performance of OSSim to simulate three classes of P2P streaming systems. We measure the memory consumption and running duration with regard to the number of peers. The results show that OSSim is capable of simulation medium-size systems with several thousands of peers. However, there would be an issue in the running duration when simulating very large networks.

Table 4.4 summarizes which design element or feature enables our simulation framework to fulfill which of the preset design requirement (Section 4.2.1).

Table 4.4: Mapping design elements and features to the requirements

	R1	R2	R3	R4	R5	R6
Validation at module and system levels	x					
Modular design with layers of functionalities		x				
General-purpose and interchangeable modules			x			
Well-defined module interfaces				x		
Reasonable memory consumption					x	
Reasonable running duration					x	
Incorporated the transit-stub network model						x

In the coming chapters, we will use OSSim to study the impact of DoS attacks on P2P streaming systems as well as the countermeasures to DoS attacks.

5

Diversification enforcement to mitigate DoS Attacks

This chapter addresses the issue of DoS attacks on head nodes, which are partners of the source. Those nodes have a critical role in connecting the source to other peers and ensure a continuous delivery of the video stream to them. Attacks on head nodes can potentially disrupt the receipt of the video stream of the whole system. The chapter starts with a brief motivation, in which we recall the potential threats caused by buffer map exchange, an essential operation in pull-based P2P streaming systems. Afterwards, we describe the striping scheme for diversification enforcement. After the evaluation section, we conclude the chapter with the a summary.

5.1 Motivation

Most popular P2P streaming systems in practical deployment, e.g., PPLive [51] and Sopcast [56], can be classified as pull-based. In such systems [32, 39], peers inform others about the chunks that are downloaded and stored in the video buffers via Buffer Maps (BMs). BM exchange is an essential operation in pull-based P2P streaming systems. Peers use received BMs to decide which chunk to request from which partner. In case of failures the mesh topology provides redundant connectivity via alternative source-to-peer paths. Even when one or several partners fail, each peer can quickly react to failures by downloading the video chunks from other partners. For this reason, pull-based systems are inherently *robust* to node churn and peer failures.

However, measurement studies [33, 67] of one of the largest pull-based P2P streaming systems reveal that peers form different tiers in terms of play-out lags. The *stable* tiering effect allows for inferring the flow of the video content distribu-

tion. As stable source-to-peer paths evolve, the topologies established for subsequent chunks become highly similar. This might not affect the robustness of these systems to node churn. However, attacks on the most relevant nodes in the overlay can damage the system severely.

The tiering effect in pull-based systems allows an outsider to gain information on the structure of the whole network and to infer the flow of video chunks among tiers. Attacks by shutting down peers on a certain tier can disrupt the flow of the video distribution. Peers at the downstream tiers stop receiving the video stream completely. When an attacker targets head nodes, which are the source's partners in the overlay topology, the damage can even be more severe. The remaining peers are isolated from the source and therefore they will not receive the video stream until one or several connections with the source are established again.

There are two potential approaches to mitigate the damage caused by attacking head nodes: (*i*) To decrease the direct dependency among peers by increasing the connectivity among them, which consequently increases the number of head nodes; and (*ii*) To undermine the attack's accuracy in identifying head nodes. We reserve the second approach for Chapter 6. In this chapter, we focus on the first approach which allows us to answer a more urgent question: *Assuming that the structure is revealed, what can we do to mitigate the damaging effect when head nodes are attacked?* We also assume that recovery measures, such as rejoining upon isolation or disconnection are always available.

Increasing connectivity among the source and peers is challenging due to resource constraints and inherent behavior of the pull-based protocol. Without increasing the bandwidth of the source and peers, the straightforward method that increases the number of partners does not work. Head nodes might gradually prefer to download chunks directly from the source due to high availability of its chunks. The higher the number of head nodes, the more likely that they have to compete with each other for a fixed source bandwidth. This leads to increasing delays and probably chunk misses since the source cannot respond timely to all chunk requests.

In the next section, striping–our diversification enforcement scheme–will be described in more detail.

5.2 Striping: A diversification enforcement scheme

In this section, we present our striping scheme for pull-based P2P streaming systems that reduces the direct dependency between peers. The scheme mitigates the negative effects of attacking head nodes, which we demonstrate in section 5.3. We begin with the idea of the scheme first. After that, we describe its design and specifications.

5.2.1 Idea to enforce diversification

Current pull-based protocols do not diversify chunk requests exhaustively. That is, peers can steadily download chunks from a few among several partners as long as they respond reliably. This leads to an implicit yet direct dependency between peers. To reduce this dependency, each peer needs to download video chunks from diverse partners. This implies that it needs to send chunk requests to more diverse partners. Towards this end, each peer requests subsets of the required chunks from different groups of partners.

At this point there are three methods to diversify the requests: (*i*) In the first method, each peer alternates between different groups of partners to request chunks at different scheduling cycles. Over the long run, the average number of requested chunks per peer is reduced. However, certain peers might, by chance, receive many chunk requests in a short period. Consequently, local overloading might happen, which affects the overall chunk dissemination. (*ii*) The second method is to split the needed set of chunks by their play-out deadlines, from most to least urgent. On-time delivery of the most urgent chunks is more critical since there might not be enough time to request them again in the next scheduling cycles. When the peer requests the most urgent chunks from a subset of partners that is not reliable, the urgent chunks might not be delivered on time. This leads to more missed chunks. (*iii*) The third method is to divide the video stream in an interleaved manner into stripes. This way, diversification is achieved while avoiding the drawbacks of the above two methods.

Subsequently to dividing chunks into stripes, each peer needs to *locally* separate its partners to different groups. There are two methods: (*i*) In the first method, the grouping is based on partners' identities, e.g., the IP addresses. This partner grouping is inflexible because it depends highly on the fluctuation of the partner list. A group might not have partners with certain identities among the peer's available partners. (*ii*) In the second method, each peer assigns its partners to different logical groups, regardless of partners' identities. Fluctuation of partners in each group can be quickly compensated by adjusting partners among the groups or even by reassigning.

We summarize our idea to enforce diversification as follows:

1. *The video stream is divided into stripes.*
2. *Each peer logically forms separate groups of partners.*
3. *Each peer requests a certain stripe from a certain group of partners.*

By doing that, the chunk downloading demand on a peer from its partners can be efficiently reduced. In conventional pull-based systems, a peer with m partners can, in principle, receive chunk requests of m times the streaming rate in the worst

case. However, the demand for each peer with striping is reduced by a factor of k, the number of stripes, when the number of partners is fixed.

Consequently, the diversification enforcement allows a peer to have more partners, given the same upload bandwidth. Thus, the peer has more source-to-peer paths and at the same time avoids overloading itself. More importantly, the source can significantly increase its number of partners, or the number of head nodes, with the same upload bandwidth. The critical connectivity between the source and the peers is therefore enhanced, thus, potentially strengthens the resilience of the system against both failures and attacks.

In the coming section, we elaborate the idea of diversification enforcement into the design of the striping scheme.

5.2.2 Design of the striping scheme

Following the high-level sketch discussed in section 3.1, this section details the design of the striping scheme. This includes the division of the video stream into stripes and the assignment of partners to different groups.

First, a stripe i consists of chunks whose sequence numbers equal to $i \ mod \ k$. Second, partners of a peer are assigned to k groups, each contains a subset of the partner list. This way, a peer requests chunks of the stripe i from partners of the group i. Figure 5.1 illustrates the design of our scheme for a generic peer. In this example, the video stream is divided into three stripes. Accordingly, seven partners are assigned to three groups.

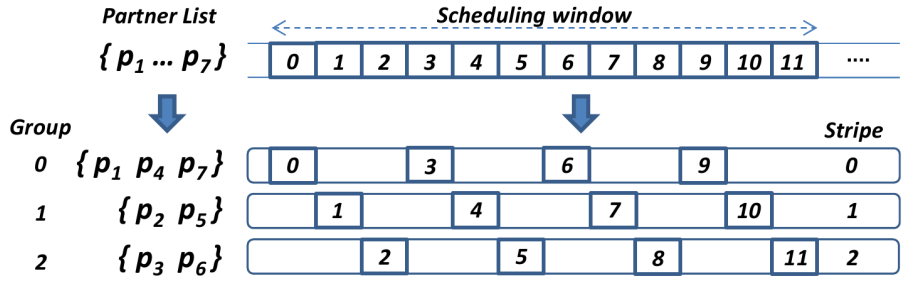


FIGURE 5.1: An example of grouping and striping: partners in each group receives requests for chunks of the respective stripe

The assignment of partners to groups has to satisfy several constraints: (*i*) Every group has at least one partner to ensure the existence of chunk providers for the respective stripe. (*ii*) All partners should be assigned to groups since partners that are not assigned to any group are not considered in requesting chunks. (*iii*) The difference in the number of partners of any two groups should be minimized. Otherwise, chunks in the stripe whose respective group has very few partners have a

lower chance to be requested and delivered successfully. (*iv*) Lastly, the assignment should minimize the difference between the number of groups assigned to each partner. Assigning a partner to several groups increases its chance to be requested more frequently, which might overloads it. We formulate the above assignment problem as follows:
Given the set of partners $P = \{p_1, ..., p_m\}$ and the set of groups $G = \{g_1, .., g_k\}$, and let $a_{ij} \in \{0, 1\}$ ($1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant m$) denote the assignment of partner p_j to group g_i, where $a_{ij} = 1$ if group g_i has partner p_j and $a_{ij} = 0$ otherwise. Define $N_i^P = \sum_{j=1}^{m} a_{ij}$ as the total number of partners assigned to group g_i, and $N_j^G = \sum_{i=1}^{k} a_{ij}$ as the total number of groups to which partner p_j is assigned.

Consequently, the problem of assigning partners to groups is to find a_{ij} such that:

$$z = minimize \left\{ \sum_{i=1}^{k} \sum_{j=1}^{m} a_{ij} \right\} \tag{5.1}$$

$$s.t. \quad \sum_{j=1}^{m} a_{ij} \geqslant 1, \quad i = 1..k \tag{5.2}$$

$$\sum_{i=1}^{k} a_{ij} \geqslant 1, \quad j = 1..m \tag{5.3}$$

$$\underset{i}{\text{argmax}}\{\sum_{j=1}^{m} a_{ij}\} - \underset{i}{\text{argmin}}\{\sum_{j=1}^{m} a_{ij}\} \leqslant 1 \tag{5.4}$$

$$\underset{j}{\text{argmax}}\{\sum_{i=1}^{k} a_{ij}\} - \underset{j}{\text{argmin}}\{\sum_{i=1}^{k} a_{ij}\} \leqslant 1 \tag{5.5}$$

In this formulation, the objective function in (5.1) is to minimize the total number of assignments of partners to groups. The constraints in (5.2) and (5.3) ensure that every partner is assigned to groups and every group has partners. The constraints in (5.4) and (5.5) are used to prevent groups from having too many partners and assigning a partner to many groups.

With the *Round-Robin assignment* of partners to groups, we achieve the optimal solution [74] with $min\{\sum_{i=1}^{k} \sum_{j=1}^{m} a_{ij}\} = max(k, m)$.

Determining parameters

Following the above design, we discuss the constraints for the two system parameters introduced in the design of the striping scheme. The first parameter is the number of stripes k. Its value is constrained by the number of requested chunks C in each scheduling cycle, which is asymptotically proportional to the streaming

rate (in chunks per second) and the scheduling interval. When $k > C$, over-striping happens, i.e. one or more stripes contain no chunks. Consequently, there are groups of partners that are redundant for chunk requests. The striping in this case is not efficient. At the other extreme, when $k = 1$, all partners are in the same group. The striping scheme operates similarly to a conventional pull-based system. The second parameter is the number of partners m. When m is too small, diversification is eventually limited because there are a few options for each chunks request. When m is too large, the necessary communication overhead can overload the system.

In the coming section, we refine the design of the striping scheme with specifications to integrate it into conventional pull-based systems.

5.2.3 Specification

Integrating the striping scheme into current pull-based protocols requires a few modifications. At the source, the number of partners scales up with the number of stripes k. At each peer, the chunk scheduling operation does not search exhaustively for available chunks from all partners. Instead, for a chunk with the sequence number s, the peer only considers partners in the group $s \ mod \ k$. Additionally, the assignment of partners to groups needs to be adapted when there are updates on the partner list. Moreover, peers execute a periodic operation, which also trigger the assignment of partners to group. This operation assures random assignments regardless of the dynamics of other partners. The adjustment should be fast to react promptly to the dynamics of peers to minimize the computational cost.

The simple Round-Robin assignment introduces a low computational cost. Even with a naive implementation, when partners are re-assigned upon each update of the partner list, the cost would be negligible.

In the next section, we integrate the striping scheme with Round-Robin assignment into DONet – a conventional pull-based protocol – and evaluate its performance.

5.3 Evaluation

In this section, the proposed striping scheme for pull-based systems is evaluated with respect to its provided resilience against attacks on head nodes. In addition, the efficiency of the proposed scheme is evaluated in detail. The evaluation aims at answering the following four questions:

1. *What damage does attacking head nodes cause to the performance of the conventional pull-based systems?*
2. *What resilience against the head node attacks is provided by our striping scheme?*

3. *What performance gain in terms of source-to-peer latency does the striping scheme introduce?*

4. *What cost in terms of signaling overhead does the striping scheme introduce?*

5.3.1 Experiment set-up

This section elaborates the set-up of our experiments. First, a description of metrics is given. Next, the attack model on head nodes is detailed. Finally, a representative system and a parameter set are then described.

Metrics From the discussion in chapter 4, we select chunk miss ratio to quantify the impact of DoS attacks. Furthermore, ***Average Miss Ratio***, ***Maximum Miss Ratio*** are used to look at chunk miss ratio from different perspectives. We also use ***Per-chunk Miss Ratio*** which is the fraction of peers missing a certain chunk. This metric quantifies the impact of missing a specific chunk.

Attack model The focus in this chapter is the attack on head nodes. Therefore, the attacker model presented in Chapter 4 is used. Specifically, we assume that the attacker obtains the list of head nodes V_h at the time of the attack. Given the attack budget A, the attacker randomly selects N_a head nodes and shuts down all of them at the same time, where $N_a = min\{A, |V_h|\}$.

Representative pull-based P2P streaming system: We select DONet [82] – a popular deployed pull-based system in the literature – as the benchmark system due to several reasons. (*i*) The design and protocol description of DONet is described in detail, which supports a verifiable implementation of the system in simulation. (*ii*) Its Rarest-First chunk scheduling strategy produces comparable performance to the state-of-the-art algorithms [83]. (*iii*) The simulation model of DONet is also validated in Chapter 4.

Parameters: In all experiments, the following parameters are used unless otherwise stated. The streaming rate is 400 kbps (the observed average video bit rate from PPLive, one of the largest deployed systems [76]). Each video buffer stores up to 30 seconds of video chunks whose sizes are 2500 Bytes, typical values for pull-based systems in the literature. The bit rate is also equivalent to a stream of 20 chunks per second. A peer starts playing out video chunks when the downloaded chunks are equivalent to around six seconds. The upload bandwidth of the source and peers are 8 Mbps and 800 kbps respectively. Even though it is unrealistic to assume that all peers have the same upload bandwidth, it is reasonable as we are only interested

in the resilience of pull-based systems against attacks. Using a homogeneous peer bandwidth eliminates the impact of the peers' characteristics in the results. The simulation duration is 1200 seconds. In the first 500 seconds, no data is collected to avoid unstable system behavior.

5.3.2 Results

In the following, we first summarize our main simulation findings and describe the effects of attacking head nodes. Afterwards, we compare DONet with the striping scheme in adversary scenarios. Finally, we compare DONet with the striping scheme in benign scenarios using two criteria: (*i*) performance in terms of source-to-peer latency; and (*ii*) cost in terms of signaling overhead.

Effects of attacking head nodes:

In the following, we answer the question: *What damage does attacking head nodes cause to the performance of the conventional pull-based systems?*

In this experiment, the maximum number of partners of the peers and the source is eight and ten respectively. To perform the attack, all of the ten head nodes are shut down simultaneously at the 900^{th} second, after the system has reached its steady-state. The instantaneous chunk miss ratio per second are plotted in dependence on time.

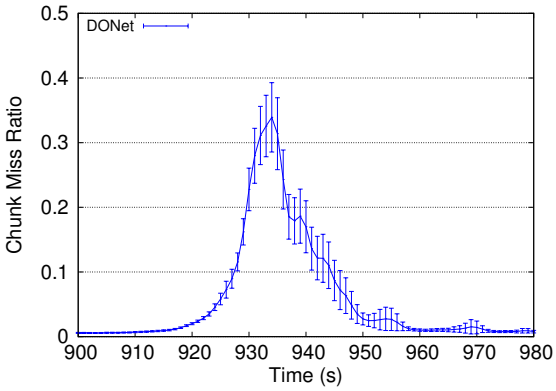


FIGURE 5.2: Instantaneous chunk miss ratio of DONet versus time, after attacking all ten head nodes

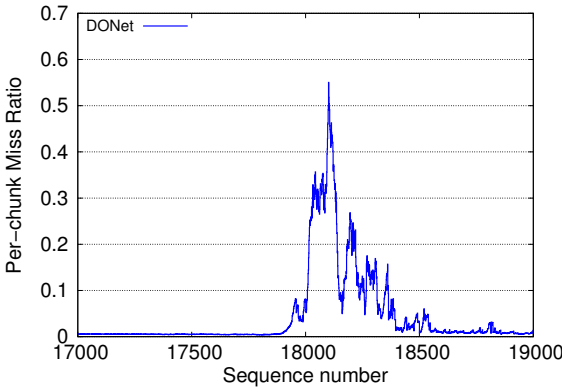


FIGURE 5.3: Chunk miss ratio of DONet versus sequence number of chunks, after attacking all ten head nodes

Figure 5.2 presents chunk miss ratio for DONet under the attack. For clarity, the figure includes only the relevant period after the attacks. It can be seen that, during the first 20 seconds, the chunk miss ratio remains as low as 1% . In the next 20

seconds, the miss ratio increases dramatically to reach its maximum of almost 35%. It then reduces quickly and remain low since the 960^{th} second.

Intuitively, the results have agreed with expectation on the behavior of pull-based protocols and can be explained as follows: Head nodes serve as intermediate sources of video chunks for the rest of the peers. When they are shut down, their partners cannot request chunks from them. However, chunk missing does not occur immediately after the attack, because the peers have large buffers. Chunk missing might start from the peers connecting to the head nodes previously. It then spreads to other peers that locate further away from the source. Chunk missing reduces when the connections between the source and the peers are established again.

To estimate more precisely the impact of missed chunks to the system's performance, we recorded the sequence number of all missed chunks and calculate the chunk miss ratio per chunk's sequence number. The results are plotted in Figure 5.3. The figure shows that the chunk miss ratio increases sharply from around 1% to a maximum of 55% in accordance with an increase in sequence number from around 18000 to 18100. It diminishes steadily for subsequent chunks and remains low for sequence numbers greater than 18500.

The spread of missed chunks to a significant fraction of peers as shown in Figure 5.3 indicates the strong and negative impact to the system's performance. A certain chunk can be more important than others, depending on whether it carries an intra-coded (I), a predictive-coded (P) or a bidirectionally predictive-coded (B) frame. A successfully decoded I-frame is required to decode the P-frames and B-frames in the same Group of Pictures (GoP). Losing an I-frame, therefore, fails the decoding of the GoP, which leads to a perceptible picture degradation at users. More severely, given the small number of head nodes a malicious party can periodically trigger attacks. Perceived quality by users in this case can be further degraded.

Comparing the striping scheme with a conventional protocol:

The second question we answer is: *What resilience against the head node attacks is provided by our striping scheme?*

In this experiment, we compare the effect of attacking head nodes on the conventional DONet ($k = 1$) versus the adapted one with striping ($k = 2, 3, 4$). The number of partners of the source and peers in the striping scheme scales with the number of stripes. The attacker budget in terms of the number of nodes that is attacked varies between 5 and 60. Maximum chunk miss ratio and average chunk miss ratio over a period of 60 seconds have been recorded and plotted.

We expect that the impact of attacks with the same budget is stronger in the conventional DONet than in the striping scheme. Since the number of head nodes in DONet is less than in the striping scheme, the same number of attacked head nodes

reduces a larger portion of the number of connections for distributing video chunks from the source to the peers. Consequently, chunk miss ratio in DONet is larger than in the striping scheme.

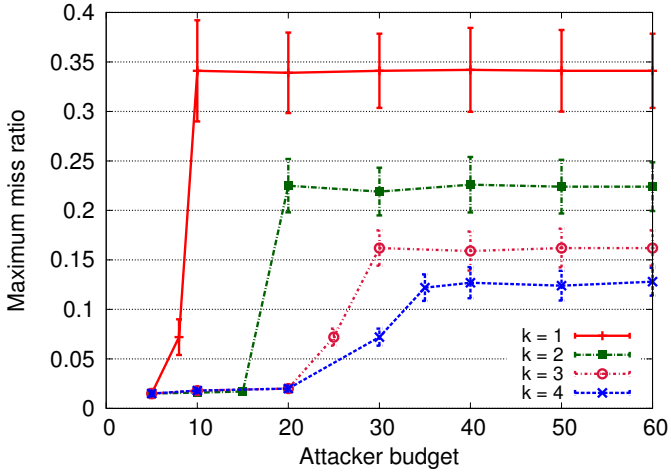


FIGURE 5.4: Comparing the conventional DONet ($k = 1$) to the adapted one with striping ($k = 2, 3, 4$) in terms of maximum chunk miss ratio after attacks

Figure 5.4 plots maximum chunk miss ratio of DONet and the striping scheme in dependence on the attacker budget. The results agree with our expectation. First, it can be observed that when more stripes are applied, the attacker needs significantly larger budget to achieve the same damage to the system. The miss ratios reach their maxima when the attacker budget equals the number of head nodes, at least. Second, it is also shown that even a conservative diversification with two stripes can reduce the maximum chunk miss ratio by around 35%.

To comprehend better the effect of attacks on the system, we plot in Figure 5.5 the average chunk miss ratio over a period of 60 seconds after attacks in dependence on the attacker budget. The figure compares the conventional DONet ($k = 1$) versus the adapted one with striping ($k = 2, 3, 4$). As seen from the figure, the striping scheme reduces the maximum of the average chunk miss ratio from 30% to 50% when the number of stripes varies from two to four. Additionally, the attacker budget has to increase proportionally to the number of stripes to maximize the damage.

One unanticipated but interesting finding in Figures 5.4 and 5.5 is that the striping scheme effectively reduced the maximum damage even when all head nodes are attacked. Note that in all experiments we applied the same process and parameters for recovering a node from isolation. The finding can be explained by two reasons: (*i*) The striping scheme disseminates chunks better due to the diversification enforcement. When chunks are requested in small numbers from diverse partners, they can be quickly downloaded. Consequently, chunk availability in the whole network is improved. (*ii*) The increased number of partners that each peer has also allows it

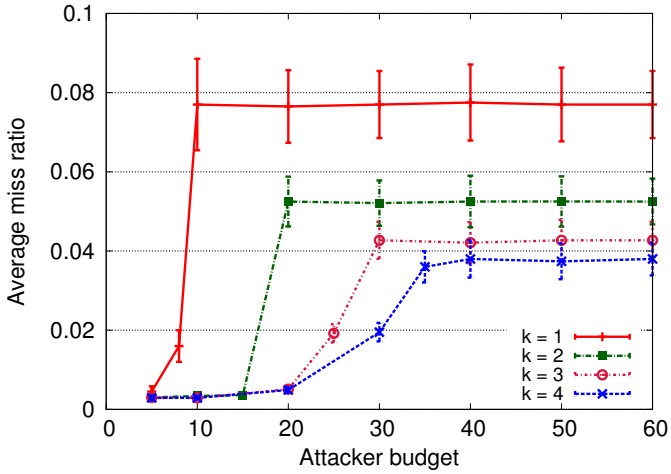


FIGURE 5.5: Comparing the conventional DONet ($k = 1$) to the adapted one with striping ($k = 2, 3, 4$) in terms of average chunk miss ratio over 60 seconds after attacks

to connect to partners that have its required chunks with a higher probability.

Performance of the striping scheme

In this section, we compare DONet with the striping scheme in benign scenarios to answer the question: *What performance gain in terms of source-to-peer latency does the striping scheme introduce?*

We vary peers' upload bandwidth from 800 kbps to 1600 kbps. We expect that the striping scheme yields lower latency since with more partners peers are more likely to connect to others closer to the source. The fewer the number of intermediate nodes between the source and a peer, the quicker video chunks can be delivered to the peer, resulting in lower latency.

Figure 5.6 shows the source-to-peer latency as a function of the peers' upload bandwidth. Both DONet and the striping scheme introduce lower latency when the peers' upload bandwidth increases. As the peers' upload bandwidth increases from 800 kbps to 1600 kbps, the source-to-peer latency decreases from upto 12s to around 7s. The trend is reasonable because the more upload bandwidth peers have, the quicker video chunks can be sent to others, resulting in a reduced latency.

Comparing the two schemes, the striping scheme yields a lower latency compared to that of DONet. With a conservative value of the number of stripes $k = 2$, the striping scheme reduces around 1s of the source-to-peer latency in all cases. The reductions are equivalent to around 6% and 15% when the peers' upload bandwidth equals 800 kbps and 1600 kbps respectively. The more stripes the striping scheme has the lower latency it produces. When $k = 4$ and the peers' upload bandwidth equals

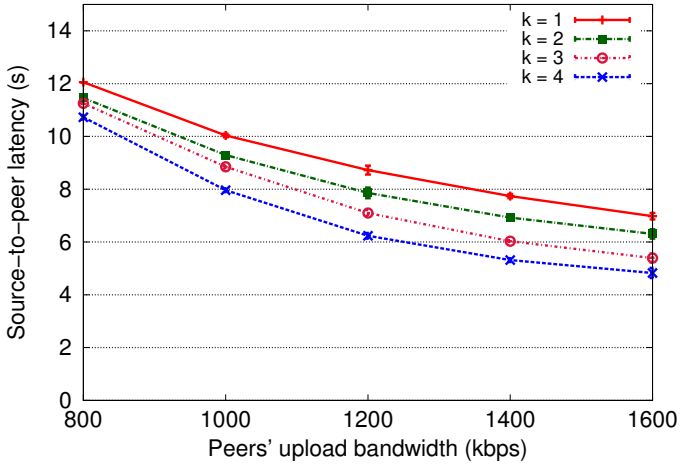


FIGURE 5.6: Comparing the source-to-peer latency of the conventional DONet ($k = 1$) and the adapted one with striping ($k = 2, 3, 4$) in benign scenarios

to 1600 kbps, the striping scheme yields a latency of almost 5s, which is equivalent to a 28% reduction as compared to DONet in the same condition. The capability of the striping scheme in reducing latency is within our expectation. The fact is that the greater the number of stripes, the more number of partners peers are allowed to have, resulting in lower source-to-peer latency as explained earlier.

Cost of the striping scheme

In this section, we would like to answer the question: *What cost in terms of signaling overhead does the striping scheme introduce?* Therefore, we compare DONet and the striping scheme in benign scenarios.

We vary the number of stripes from two to nine. We expect that the striping scheme produces higher signaling overhead since each peer has more partners. The exchanged BMs are increased subsequently. Furthermore, each peer probably sends more chunk requests per scheduling cycle because the peer should diversify the requests to different partners.

Figure 5.7 shows the signaling overhead in dependence on the number of stripes. In this figure, the lower horizontal line represents the average signaling overhead of DONet. As expected, the overhead increases steadily with an increasing number of stripes. Specifically, for each additional stripe, the signaling overhead increases by around one percent. The total signaling overhead is less than 10% when the number of stripes equals to four, which significantly reduces damages caused by attacks. Note that a significant portion of the total signaling overhead would stem from exchanging BMs. This overhead, however, can be strongly reduced by techniques, such as the one described by Li et al. in [42]. The authors propose a scheme that allows each peer

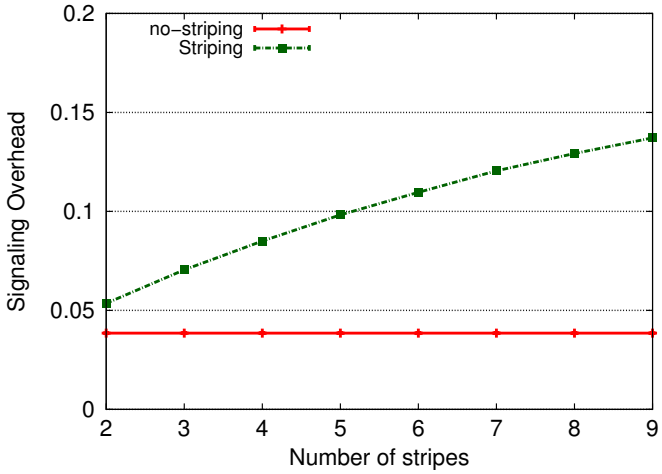


FIGURE 5.7: Comparing the signaling overhead of the conventional DONet (no-striping) and the adapted one with striping when there is no attacks

to send only the difference between consecutive buffer maps. The overall overhead volume is therefore reduced significantly.

5.4 Summary

This chapter investigates the impacts of the attacks on head nodes to pull-based P2P streaming systems. Head nodes are partners of the source. They connect the source with the rest of the peers and consequently have a critical role in disseminating video chunks from the source to the whole system. Attacking the system by shutting down head nodes can affect the system drastically.

Unfortunately, head nodes can be revealed through a fundamental operation of pull-based systems, which is the buffer map exchange. Buffer maps contain the availability information of peers' video buffers. This information is nonidentical between different peers because some peers receive video chunks much earlier than others. More importantly, it is possible to collect buffer maps from peers to infer the overlay structure, subsequently to identify head nodes. They normally receive the earliest video chunks directly from the source. The consequence of the attacks on identified head nodes can badly affect the system.

We conduct simulation studies to answer the urgent question: *How the attacks on head nodes affect pull-based systems?* We assume that the attacker has a global knowledge, thus, all the head nodes are known to the attacker. Using both the maximum and average miss ratios to quantify different aspects of the damage, the extensive simulation results show that such attacks on head nodes cause a maximum chunk miss ratio of up to 40%. More importantly, the missed chunks propagate to a

considerable fraction of peers, of up to 50%.

Subsequently, we develop the striping scheme as a countermeasure against the attacks. Our approach is to decrease the direct dependency of a peer on a particular partner. This allows for increasing the number of partners of each peers, which further improves peers' connectivity. More critically, the system has more head nodes because the source can accept more peers as partners while avoiding to overload itself. To do so, the idea is to enforce peers to diversify their chunk requests to different groups of partners. Each peer assigns its partners into groups in a round-robin manner. This assignment is proven to be optimal in minimizing the dependency of the peer to a particular partners. In the next step, the stream of video chunks is divided into logical stripes using the modular operation of the chunks' sequence numbers. The number of stripes respectively equals to the number of groups. In each scheduling cycle, the peer enforces itself to only request chunks of a stripe from the respective group.

We assess the striping scheme by an extensive simulation study. Results exhibit that in adversary scenarios, the striping scheme effectively reduces both the maximum and average chunk miss ratios by 35% and 30% respectively, even in a conservative setting with two stripes. In benign scenarios, the striping scheme significantly reduces the source-to-peer latency from at least 7% to around 28%. The cost of the striping scheme is, however, an increase in the signaling overhead. The scheme generates around as low as 1% extra signaling overhead with each additional stripe.

Following the study that we present in this chapter, next chapter addresses the challenge to undermine the attack's accuracy in identifying head nodes in the first place.

6

Partner swapping against the inference attacks

In pull-based P2P streaming systems, the tiering effect refers to distinguishable tiers formed by play-out lags of peers. Analyzing peers' buffer maps to infer the overlay structure consisting of those tiers has been reported in the literature [33]. This allows an outsider to infer the overlay structure of the system and the flow of video chunks between tiers. When a similar technique is used by an adversary (or inference attacker), pull-based systems can be vulnerable to attacks. By shutting down peers on a certain tier, the attacker can disrupt the flow of the video distribution. Furthermore, the damage can be severe when an attacker targets head nodes, which are the source's partners. When they are shut down, the remaining peers are isolated from the source, resulting in a disruption of the video dissemination.

In the previous chapter, we have studied the impact of an attacker with global knowledge, which can completely identify head nodes and shut them down. Subsequently, we have developed the striping scheme as a countermeasure to the attack on head nodes. However, as long as the attacker can infer overlay structure, especially the head nodes, pull-based systems are still vulnerable to inference attacks.

This chapter addresses the critical question of how to prevent an inference attacker from accurately identifying head nodes in the first place. To answer the question, we first review the technique used to infer the overlay structure. After that, we describe inference attacker model and SWAP, our countermeasure to the inference attacks. Next, we present our theoretical analysis of the problem, including the attacker model and the countermeasure. After the evaluation section, we conclude the chapter.

6.1 Inferring the overlay structure

The information from buffer maps in a pull-based P2P video streaming system can provide knowledge about the overlay structure of the system [33]. Figure 6.1 illustrates the mapping from the availability of chunks in a video buffer into a buffer map consisting of an array of "0" and "1" bits. The buffer head's sequence number (or *buffer head*, for short) indicates the latest chunk in the buffer. Peers' buffer heads differ from one another in general depending on network conditions (such as jitter) and who their partners are. For example, a peer connecting directly to the source can receive video chunks significantly earlier than others, who do not have direct connections to the source. Comparing the buffer heads of peers buffer maps, therefore, provides their relative position in the overlay structure.

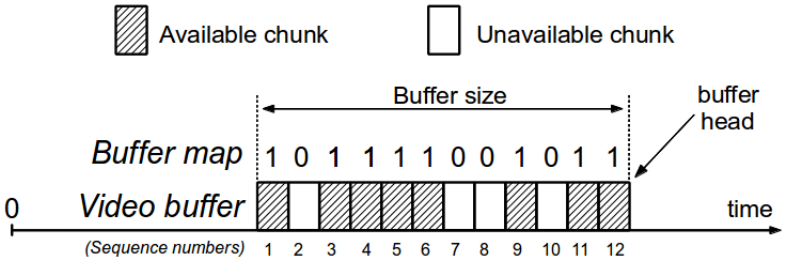


FIGURE 6.1: Mapping of a buffer, partially filled with downloaded video chunks and available for playing back (buffer's head is the newest available chunk)

Let's consider two peers u and v at time t_i with their buffer maps arrived at time $t_{u,i}, t_{v,i}$ containing buffer heads $h_{u,i}, h_{v,i}$ respectively. The relative offset of u with respect to v is defined as follows [33]:

$$\delta_{u,i} := (h_{u,i} - h_{v,i}) - (t_{u,i} - t_{v,i}) \cdot r/l \tag{6.1}$$

where l denotes the chunk size and r the bit rate of the video stream. Next, $\delta_{u,i}$ is calculated for all peers u over time using a common reference point. Peers' relative offsets exhibit a clear and constant ordering. From those offsets, peers closer to the source are clearly distinguished from those farther away.

6.2 The inference attacker

In this section, we describe the attacker model used in the remainder of this section. We derive the model from the inference technique presented in the previous section. That technique has been used by Hei et al. [33] to infer the overlay structure of a

small setup of PPLive, a commercialized pull-based system. In the inference attacker model, we extend the technique to identify head nodes and experiment it on DONet, a conventional yet more generic pull-based system. We select DONet instead of PPLive in our study because PPLive's source code is propriety and its documentation is not available. Whereas, DONet's protocol description is well documented, furthermore, its implementation has been validated.

The attacker's goal is to prevent peers from receiving a continuous video stream for a certain period of time. To achieve this, it attacks a set of peers to break the overlay structure of the system. We assume that an attacker A is able to shut down arbitrary nodes except the source within the system. However, A is limited with regard to the number of nodes x it can shut down at any given time, referred to as the *attacker's budget*. Given this limitation, the attacker must attempt to shut down the most relevant peers, the head nodes, to cause the most damage for some period after the attack.

In the following, we outline the approach followed by the inference attacker. It consists of three steps: *probing*, *inference*, and *attack*.

6.2.1 Probing buffer maps

The attacker collects buffer maps from peers to aggregate the contained information and consequently infer the overlay structure of the system. This accumulation of buffer maps can be performed either by actively requesting them or by storing them from regular exchange operations. While the regular buffer map exchange is a valid operation in all pull-based systems, active requests are only allowed in some systems, e.g., in [33]. Regardless of the way this collection is implemented, buffer maps are periodically gathered in batches, called *probes*. In an ideal attack scenario, the attacker obtains the buffer maps from all active peers in the system for each probe. However, to be more realistic, we assume that the attacker obtains a buffer map from a peer with a probability q ($0 \leqslant q \leqslant 1$) which depends on network latency, congestion, etc. Subsequently, we assume that an attacker performs a total of m probes $p_1, p_2, \ldots p_m$ starting every T_p seconds. These probes should be performed shortly before the attack to ensure up-to-date information.

6.2.2 Inferring the overlay structure

First, the attacker selects a reference peer v, potentially under its control, whose buffer map appears in all probes. Using Eq. 6.1, the attacker calculates the offset $\delta_{u,i}$ for each buffer map from a peer $u \neq v$ collected during the probe p_i. After m probes, the attacker calculates the average offset $\bar{\delta}_u$ of each probed peers u as follows:

$$\bar{\delta}_u := \frac{\sum_{i=1}^{m} \delta_{u,i}}{m} \tag{6.2}$$

6.2.3 Attacking the system

Head nodes receive video chunks before other peers which should results in high values of their averaged offset. Therefore, the attacker sorts the list of all probed peers u in descending order of their averaged offsets $\bar{\delta}_u$. From this sorted list, the attacker selects the top x nodes to attack and thereby shut down. Assuming that these nodes are closest to the source, attacking them should cause the greatest damage possible with the attacker's resources.

In summary, this section presents the inference attacker, a practical attack model that works in three steps: (*i*) collecting information about peers by probing their buffer maps; (*ii*) inferring the system's overlay structure; and (*iii*) executing attacks by targeting the most promising peers. Note that we use a simplified version of this strategy for our theoretical analysis to limit the complexity.

6.3 The SWAP scheme

In this section, we describe the SWAP scheme to counter the inference attacker. Such a countermeasure needs to satisfy the following three requirements:

1. The countermeasure has to reduce significantly the attack's accuracy and therefore effectively mitigate the negative impact of the attacks.
2. The performance penalty should be reasonably low.
3. The additional overhead should be low.

To fulfill the first requirement, there are two possibilities to undermine the attack's accuracy to infer the overlay structure, especially in identifying head nodes. First, we could insert bogus information into the exchanged buffer maps so that the attacker cannot reliably infer the overlay structure. Second, we could enforce the overlay structure to change constantly, so that the overlay structure inferred by an attacker is quickly outdated. Adding bogus information can confuse the attacker, but it confuses benign peers as well, causing inefficient requests for video chunks. Consequently, this reduces the overall system's performance. Hence, this approach directly violates the second requirement and is therefore eliminated.

The second approach requires peers to proactively change their partners. Consequently, the system introduces more dynamics and overhead. However, we have

also learned that pull-based P2P streaming systems are highly robust to the dynamics of peers [40]. Subsequently, there potentially exists a certain level of dynamics that minimizes negative impacts to the system while allowing the system to mitigate damages caused by an inference attacker. Our SWAP scheme follows this approach.

6.3.1 Basic idea behind the SWAP scheme

The basic idea of our SWAP scheme is to enforce peers and especially the source to proactively change their partners. This means to replace an existing partner with another peer (or replacement partner). At this point, there are two follow-up questions for the swap operation:

1. Which partners should be dropped?
2. Which replacement partner should be selected?

The answer to the above two questions should minimize negative impacts on the system. It is however hard to fulfill the requirement if peers have to decide from their local knowledge only. Our strategy is to leverage the information in buffer maps received from a peer's partners. They provide a peer a little more information about its neighboring peers and consequently the local structure between themselves.

To simplify the explanation in this section, we introduce the concepts of *upstream* and *downstream partners* of a peer v, meaning whether they tent to receive video chunks earlier or later than v does. To quantify the timing of received chunks, we use Eq. 6.2 to calculate the average offsets of v's partners with regard to v itself from recently received buffer maps. Let $\bar{\delta}_u$ denote the average offset of v's partner u. Since $\bar{\delta}_v = 0$, u is called an upstream partner of v if $\bar{\delta}_u > 0$ and a downstream partner otherwise. The separation of partners basing on their average offsets is illustrated in Figure 6.2. Assuming that their average offsets satisfy $\bar{\delta}_1 \geqslant \bar{\delta}_2 \geqslant 0 \geqslant \bar{\delta}_3 \geqslant \bar{\delta}_4 \geqslant \bar{\delta}_5$ ($\bar{\delta}_v = 0$ since v is the reference point for itself), Peer v's upstream partners are (u_1, u_2) and its downstream partners are (u_3, u_4, u_5).

Our answer to the first questions is that a peer selects a downstream partner to drop, since dropping an upstream partner reduces the chance of the peer itself in obtaining video chunks. Regarding the second question, a downstream peer should also be selected to replace the dropped partner because it can introduce a potential security issue to do otherwise. Allowing peers to connect to upstream peers enables malicious peers to swap frequently hoping to mount up the overlay structure to block the source node and disrupt the video delivery to benign peers.

To summarize, our idea to defend a pull-based P2P streaming system against the inference attack is to allow peers as well as the source to proactively change their partners. Both the dropped partner and its replacement should be selected from

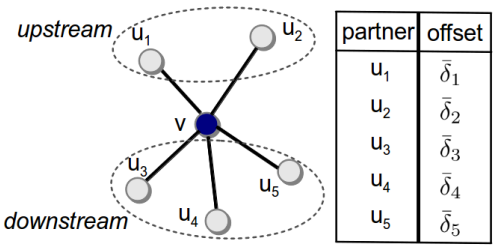


FIGURE 6.2: Peer v's partners: upstream (u_1, u_2) and downstream (u_3, u_4, u_5), given their average offsets satisfy $\bar{\delta}_1 \geqslant \bar{\delta}_2 \geqslant 0 \geqslant \bar{\delta}_3 \geqslant \bar{\delta}_4 \geqslant \bar{\delta}_5$ ($\bar{\delta}_v = 0$ since v is the reference point for itself)

downstream peers. Following the above high-level sketch, the design of SWAP will be detailed next.

6.3.2 Design of the SWAP scheme

The main concern here is the selection of a replacement partner. A straightforward solution is to request a random peer from the membership service, such as a tracker keeping a list of active peers in the system. It is, however, very unlikely that peers can obtain downstream replacement partners as they need. Therefore, preparing a downstream replacement partner for swapping operations requires a collaboration between peers. SWAP realizes the collaboration by the following two primitives, namely *partner nomination* and *nomination forwarding*. After describing those two primitives shortly, we present the swap operation.

Partner nomination

To prepare for swapping operations, every peer should suggest its replacement candidate once the peer is dropped by one of its upstream partners. For this reason, a partner nomination is required, in which a downstream peer is suggested as the replacement partner for the swapping operation. The followup question is how a peer should nominate its partners. In SWAP, a peer nominates its downstream partners to its upstream ones in a Round-Robin manner. For example, in the Figure 6.2, two among the three downstream partners $\{v_3, v_4, v_5\}$ can be randomly nominated to the upstream ones $\{v_1, v_2\}$. In this way, SWAP nominates partners evenly to minimize the chance of a partner to be selected by many peers. Nomination should be performed periodically to ensure a constant and fresh availability of replacement partners.

Nomination forwarding

Accepting nominations from partners immediately only allows peers to swap with nearby peers in the overlay. That cannot help the system from an inference attacker with a large budget. SWAP solves that problem by requiring peers to forward each nomination several times towards upstream peers. This way, peers can connect to more diverse peers whose positions are farther away in the downstream, further hindering the attacker from inferring the overlay structure. Therefore, a counter in each nomination message is introduced. The counter is preset to n_h representing the total number of forwardings until the nomination is eventually accepted. After each forwarding, the counter is decremented. When the counter of a nomination reaches zero, peers should accept that nomination. Decrementing the counter also prevent nomination messages from circulating forever in the system, causing overhead. Figure 6.3 illustrates two examples for swapping, where n_f equals one and two respectively. In the former case, A accepts the nomination from C and connects to this node before dropping its current partner B. In the later case, A accepts the nomination from D after it is forwarded via C and B. Afterwards, A connects to D before dropping B.

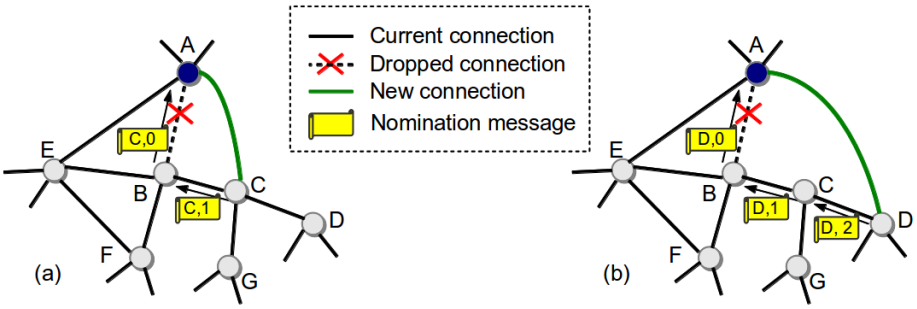


FIGURE 6.3: Swap operation in which peer A: a) accepts C's nomination forwarded via B, connects to C and drops B ($n_f = 1$); or b) accepts D's nomination forwarded via C and B, connects to D and drops B ($n_f = 2$). The nomination message contains the name of the nominated peer and a counter, which is decremented after each forwarding at an intermediate node

Swap operation

Since an attack can happen at any time, the system should be ready against those attacks all the time. The swap operation is therefore executed regularly by each peer. In each swap cycle, a peer selects a few partners and swap them with their respective replacement partners obtained via nominations.

Parameters

Our SWAP scheme is characterized by three parameters for the nomination and swap operations. First, the number of nomination forwardings n_f needs to be specified. The minimum value of n_f is one, meaning to swap with nearby peers in the overlay. The greater n_f is, peers can swap with further downstream peers. A too large n_f, however, can lead to circulated nominations. Second, the swap operation uses two parameters, namely the swap interval T_s and the number of swapped partners m_s in each swap operation. Their combination can influence significantly the resistance to attacks as well as the overall performance of the systems. Swapping too frequently or swapping too many partners at once might help the system to change the overlay structure more swiftly and drastically to avoid inference attacks. However, this also increases dynamics to the system that it might not be able to sustain.

6.3.3 Specification and Implementation

To integrate our SWAP scheme into an existing pull-based P2P streaming system, we describe essential adaptations as follows. To allow for periodic swapping and partner nomination operations, two timers are required for each peer. The first timer is triggered every T_n seconds to update the peer's partners about its nominated peer's address The second timer is triggered every T_s seconds to execute the swap operation.

6.4 Theoretical analysis

The purpose of this section is twofold. First, we aim to show that an adversary can easily identify head nodes in a stable system by frequently performing buffer map probes. In particular, we determine a lower bound on the attack's accuracy, i.e., the fraction of sabotaged head nodes, for m requested buffer maps. Based on this result, one can also determine an upper bound on the number of probes required to achieve a certain accuracy.

Second, we aim to prevent the attacker from determining the head nodes. For this purpose, we proposed to regularly swap downstream nodes. In this section, we determine how to choose the swapping interval such that the attack's accuracy is guaranteed to remain below a certain threshold. In contrast to the the first bound, we now determine an upper bound on the attack's accuracy if the attacker can request up to m buffer map probes in one swapping interval. If we require the attack's accuracy to be below a certain threshold, our results enable us to determine a number m of buffer map probes the attacker may be allowed to request in order to confirm with the required threshold. The length of the proposed swapping interval is determined by multiplication of m with the time between two subsequent requests for buffer map probes.

6.4.1 Identifying Head Nodes

We aim to derive a lower bound on the attack's accuracy without topology changes. Such a lower bound indicates that the attacker can identify critical nodes with high accuracy at a low cost. First, we formalize the problem of identifying the head nodes in a stable system. Based on the formalization, we then simplify the mathematical model to relate to a problem in probability. Last, we obtain the desired bound for the simplified problem.

In order to obtain lower bound on the accuracy an attacker, we propose one attacker strategy and then compute a lower bound for the proposed strategy. Note that we do not claim that the proposed attack strategy inevitably offers the maximum accuracy. However, as we are interested in a lower bound on the achievable accuracy, analyzing one strategy is sufficient.

Recall that N is the total number of nodes in the system, N_h the number of head nodes, and x the attacker's budget. Furthermore, the attacker requests m probes of buffer maps. A node's buffer map is contained in a probe with probability q. Let $Y_{m,p}$ be the random variable denoting the attack accuracy, i.e., the fraction N_x/N_h with N_x denoting the number of head nodes the attacker sabotages. Formally, we thus aim to determine a lower bound on $\mathbb{E}(Y_{m,p})$, with the expectation being defined over all possible buffer map probes.

In order to solve the problem, we first introduce some additional notation. Let $ord(i, j)$ be the j-th node to receive the i-th chunk. We enumerate the buffer maps probes with $\alpha = 1 \dots m$, and let $C(v) \in \{1, \dots, m\}$ indicate the probes containing v's buffer map. Note that C corresponds to a function from the set of nodes V into the power set $\mathcal{P}(\{1, \dots m\})$, i.e., the set of all subsets, of possible buffer map probes indices $\{1, \dots, m\}$. Furthermore, for the α-th probe, let $lci(\alpha, v)$ denote the index of the latest chunk in the buffer map of v if $\alpha \in C(v)$. We can now formalize our assumptions and results in an adequate manner.

We make the following assumptions to obtain in our mathematical model for deriving the attack's accuracy. First, we assume that the order in which nodes receive chunks is the same for all chunks, i.e., $ord(j) = ord(i_1, j) = ord(i_2, j)$ for all i_1, i_2. As a consequence, we can enumerate nodes $v_1, \dots, v_N$ such that $ord(j) = v_j$. Furthermore, we assume that a probe of buffer maps only contains buffer maps from one specific time t_0, rather than buffer maps from varying points in time. In this manner, v_j has received at least as many chunks as v_l for $j < l$, i.e., if $\alpha \in C(v_j) \cap C(v_l)$, then $lci(\alpha, v_j) \geqslant lci(\alpha, v_l)$. Last, we assume that all head nodes receive chunks before non-head nodes, i.e., the head nodes correspond to the nodes $v_1, \dots, v_h$. In practice, network dynamics, inhomogenous latencies, and network jitter preclude the above assumptions. Our model could account for such an instability, as long as the variation is small, by including probabilistic changes in the order. However, the drastically increased model complexity is disproportional to the expected gain.

We assume that the attacker identifies head nodes based upon the 'intuition' that their latest chunk is of a higher index. The above assumptions merely formalize this intuition in a straightforward manner.

Now, we describe the attacker strategy for which we derive the lower bounds. First, the attacker only includes nodes for which it received any buffer maps in its list of potential head nodes. So, let $V_C = \{v \in V : C(v) \neq \varnothing\}$ be the set of such nodes. Furthermore, let $v \prec_a u$ denote the fact that the adversary perceives v to receive chunks earlier than u. Let $M_C(v) = \{u : v \prec_a u\}$ be the set of nodes the adversary perceives to receive chunks later than u for the buffer map probes defined by the sets $C(v)$. We suggest the attacker to sabotage x nodes $v \in V_C$ for which $|M_C(v)|$ is maximized, i.e., the attacker sabotages nodes for which the number of nodes which are guaranteed to receive chunks later is maximized. The motivation for the strategy is the increased probability that such nodes are likely to be close to the source as many nodes receive chunks later. It remains to detail how the adversary derives $M_C(v)$, or more specifically the partial order relations, given the buffer map probes. Our key observations for determining partial order relations between v and u for one probe are

1. $v \prec_a u$ if $\alpha \in C(v) \cap C(u)$ and $lci(\alpha, v) > lci(\alpha, u)$, and
2. $v \prec_a u$ if there exists a node w with $v \prec_a w$ and $w \prec_a u$.

The first condition holds because clearly v_j received one chunk before u and thus by consistency of the order always receives chunks first. The second condition follows by the transitivity of a partial order. Thus by successively considering each probe and applying the above rules, the attacker can determine $M_C(v)$ for all $v \in V_C$ and thus select its x nodes, breaking ties randomly. With regard to the proposed attack strategy, we can now derive a lower bound on the expected attack accuracy. Note that this attack strategy is different to the one suggested in Section 6.2. The reason lies in the fact that the order of receiving chunks is not necessarily strictly constant in practice due to network jitter. Nodes with similar delays to the source receive chunks in a different order. Thus, the buffer map probes in practice do not result in a partial order, so that a more complex strategy is required to overcome such differences in the retrieval order. However, in our simplified model, the above attack strategy is sufficient and reflects the main idea of both attack strategy: identify head nodes due to their low delay to the source. By using a simplified, possibly less accurate attack strategy, we obtain a good lower bound and focus on the main ideas for the attack's effectiveness.

For simplification, we assume that u and v's buffer maps always allow us to determine the partial order relation between the nodes, i.e., we have either $lci(\alpha, u) > lci(\alpha, v)$ or $lci(\alpha, v) < lci(\alpha, u)$. In practice, the above assumption clearly does not hold. As the number of pairs such that $lci(\alpha, v) = lci(\alpha, u)$ depends on the

streaming rate and the latencies, allowing for this possibility increases the model complexity and reduces the generality of the approach by incorporating system-dependent parameters. Thus, we first consider the simplified problem and then extend the model.

Proposition 1. *Let*

$$\forall j \neq l, \alpha \in C(v_j) \cap C(v_l) \implies lci(\alpha, v_j) > lci(\alpha, v_l). \tag{6.3}$$

Then a lower bound on the expected attack's accuracy is

$$\mathbb{E}(Y_{m,q}) \geqslant \sum_{C:V \to \mathcal{P}(\{1,\dots,m\})} \frac{\{j \leqslant N_h : |V_C| - |M_C(v_j)| < x\}}{N_h} \prod_{i=1}^{m} q^{|C(v_i)|} (1-q)^{m-|C(v_i)|} \tag{6.4}$$

with $M_C(v_j) = \{v : v_j \prec_a v\}$ denoting the set of nodes the attacker perceives to receive chunk later than v_j.

Proof. Consider one realization of buffer map probes uniquely defined by a function C. We derive a lower bound $acc(C)$ on the accuracy of the attacker for the buffer map probes C assuming that the attacker utilizes the above strategy. Afterwards, we determine the probability $P(BM = C)$ for this buffer map probe. Formally, we hence obtain the bound by

$$\mathbb{E}(Y_{m,q}) = \sum_{C:V \to \mathcal{P}(\{1,\dots,m\})} acc(C) P(BM = C). \tag{6.5}$$

In order to determine $acc(C)$, recall that the attacker sabotages x nodes v for which the sets $M_C(v)$ is of maximal size. Consider that if less than x nodes are contained in $V_C \backslash M_C(v_j)$, there are also less than x nodes v with $|M_C(v)| \geqslant |M_C(v_j)|$. We shortly prove the above statement. Let $v \in M_C(v_j)$, i.e., we have $v_j \prec_a v$. Furthermore, for all nodes $w \in M_C(v)$, $v \prec_a w$ holds. Hence, by the definition of $M_C(v)$, we have $M_C(v) \subset M_C(v_j)$. Due to the anti-symmetry of $\prec_a$, $v_j \notin M_C(v)$ and thus the subset $M_C(v)$ is a real subset of $M_C(v_j)$. So, $|M_C(v)| < |M_C(v_j)|$ for all $v \in M_C(v_j)$. Thus, a head node v_j is guaranteed to be sabotaged if all but x nodes are contained in $M_C(v_j)$, resulting in the lower bound

$$acc(C) \geqslant \frac{\{j \leqslant N_h : |V_C| - |M_C(v_j)| < x\}}{N_h} \tag{6.6}$$

on the attack's accuracy.

Now, we compute the probability $P(BM = C)$ of certain buffer map probes. Note that each probe contains the buffer map of a node v with probability q. Thus,

the probability of v's buffer map to be contained in the probes specified by $C(v)$ is $q^{|C(v)|}(1-q)^{m-|C(v)|}$. As buffer maps are probed independently, we indeed get

$$P(BM = C) = \prod_{i=1}^{N} q^{|C(v_i)|}(1-q)^{m-|C(v_i)|}. \tag{6.7}$$

The claim follows by inserting Eq. 6.6 and Eq. 6.7 in Eq. 6.5. □

Note that Eq. 6.4 require computation cost exponential in the number of requested buffer maps. Thus, we apply Monte Carlo sampling when computing the desired bounds for our evaluation.

6.4.2 Swapping Interval

In this section, we determine an upper bound on the expected attack accuracy if all nodes switch m_s downstream partners each T_s seconds. From such a bound, we can obtain the swapping interval T_s and the number of swapped nodes f per interval in order to maintain an attack's accuracy below a certain threshold. We assume that the attacker continuously requests buffer maps and then decides to sabotage x nodes at some point in time. We bound the attack's accuracy of this attack. Note that our bound holds regardless of the attacker's strategy.

Proposition 2. *Let T_p be the interval at which the attacker requests buffer map probes, q be the probability of node's buffer map to be probed, let $T_s = m \cdot T_p - T_{init}$ be the swapping interval with T_{init} denoting the time a new head node requires to catch up to the old head nodes with regard to the number of received chunks and m_s be the number of swapped nodes. Let A_x denote the event that newly selected head nodes are not chosen from the x nodes currently marked for sabotage. An upper bound on the expected attack accuracy $Z_{m,q,f}$ at any point in time is given by*

$$\mathbb{E}(Z_{m,q,m_s}|A_x) \leqslant \frac{1}{m}\sum_{j=1}^{\infty}\left(1-\frac{m_s}{N_h}\right)^{j-1}\frac{m_s}{N_h} \sum_{i=0}^{m-1}\left(1-(1-q+q(1-q)^x)^{mj+i}\right). \tag{6.8}$$

Proof. The main idea of the proof is that the attacker has to be able to compare the buffer maps of new head nodes with its current x candidates for sabotage before it considers the node as a new head node. This need for comparison is independent of the actual attacker strategy. We first derive an upper bound on the probability that a head is detected using γ buffer map probes and secondly derive the probability that the attack requested γ buffer maps since the head node caught up with the other head nodes. Formally, let I be the event that a random head node is correctly

identified and R the number of buffer map probes requested by the attacker after the new head node caught up.

$$\mathbb{E}(Z_{m,q,f}|A_x) = \sum_{\gamma=1}^{\infty} P(I|R=\gamma)P(R=\gamma). \tag{6.9}$$

In the following, we determine $P(I|R=\gamma)$ and $P(R=\gamma)$.

When considering $P(I|R=\gamma)$, note that for one probe, i.e., $\gamma = 1$, the probability that a comparison is not possible is given by $1-q+q(1-q)$. The probability follows because either v's buffer map is not contained in the probe or v's buffer map is contained but none of the buffer maps of the x nodes. As the probes are chosen independently, we obtain

$$P(I|R=\gamma) = 1-(1-q+q(1-q)^x)^\gamma, \tag{6.10}$$

the complementary probability of the event that a comparison is not possible for all γ probes.

It remains to determine $P(R=\gamma)$. Note that $R = m \cdot R_1 + R_2$ with R_1 denoting the number of swaps and R_2 denoting the number of buffer map probes considered since the last swap. R_2 is uniformly distributed in $0,\ldots,m-1$. For determining R_1, note that the probability to swap a certain head node is $\frac{m_s}{N_h}$, the fraction of swapped head nodes per swap. Thus, the number of swaps since a certain node was last swapped is given by a hypergeometric distribution with parameter $\frac{f}{N_h}$. Rewriting $\gamma = j \cdot m + i$, we obtain

$$P(R=\gamma) = P(R_1=j, R_2=i) = \left(1-\frac{f}{N_h}\right)^{j-1} \frac{f}{N_h}\frac{1}{m}. \tag{6.11}$$

Thus, we have derived the missing terms in Eq. 6.9. The claim in Eq. 6.8 follows by inserting Eq. 6.10 and Eq. 6.11 in Eq. 6.9 and elementary mathematical operations. □

We have derive the desired bounds on the attack's accuracy for our (simplified) model. In the following, we compare these bounds to the attack's accuracy in a simulation study and relate the attack's accuracy to the quality of service, measured by the chunk miss ratio.

6.5 Evaluation

In this section we investigate the impact of the inference attacker on pull-based systems and SWAP. Specifically, we would like to answer the following three questions: (*i*) How accurate does the inference attacker identify head nodes? (*ii*) To which

extent does SWAP increase the resilience of pull-based systems against the inference attacker? (*iii*) At which cost does SWAP perform in benign situations? We start by describing the metrics and our simulation first.

The stream source is constantly fed by a stream of 2500-Byte video chunks. The streaming bit rate is set to 400 kbps, which is a typical average rate reported in the literature [76]. Peers with video buffers storing up to 30 seconds of video chunks start playing out when roughly 20 percent of their buffers are filled. The upload bandwidth of the source and peers are 8 Mbps and 1 Mbps respectively. Even though unrealistic, the assumption of homogeneous peers are reasonable since it eliminates the impact of the peers' upload bandwidth in the results and focus only on the resilience of pull-based systems against attacks. The simulation duration is 1200 seconds and the attacks are performed at the $800th$ second, when the system already reaches its steady state. We repeat each simulation setting 30 times.

In order to evaluate the strength of the attack and our defense mechanism, we consider the impact of the probability q and the number of probes m influence the accuracy of the the attack. The probing interval T_p equals 0.5 seconds in all settings. Next, we vary q between 0.1 and 1.0, while the value of m was varied between 1 and 5. The attacker's budget x is chosen between 5 and 50 in steps of 5. This completes our set-up for the evaluation. Due to space constraints, we only show selected parameter settings. However, the remaining results are similar and entail the same conclusions.

6.5.1 Results

In the following, we first summarize our main findings on the characteristics of the inference attacker. Afterwards, we evaluate our SWAP scheme in mitigating the negative impact of the inference attacker. This also includes the costs of our scheme.

On the accuracy of the inference attacker

In this section, we seek the understanding about the dependence of the attack's accuracy to the different parameters of the inference attacker model. Specifically, we would like to answer the question: How the probability q and the number of probes m influence the attack's accuracy. We compare our simulation results with the theoretical bound given by Eq. 6.4.

We expect that an increased value of q can significantly improve the attack's accuracy. Moreover, the higher the number of probes is used to infer the overlay structure of the system, the better the attack's accuracy is.

Figure 6.4 displays the dependency of the attack's accuracy on the probability q and the number of probes m. The results show the following: (i) The lower bounds from the model predictions and the simulation results are very close to each other.

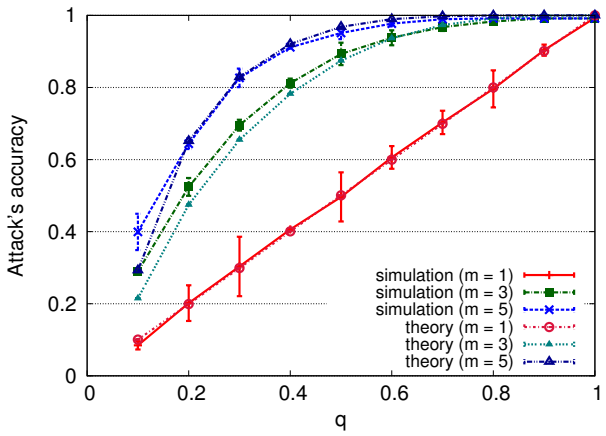


FIGURE 6.4: The dependence of the attack's accuracy to the probing interval and the number of considered probes

Thus, the impact of network jitter on the delay order, which is ignored in our model, does not seem to have a significant impact, as the model's result are validated by the simulations. Indeed, the simplified model presents a lower bound on the actual attack's accuracy. The reason for the difference between model and simulation is given by the model's simplified attack strategy. The attacker only relates two nodes if it can establish a partial order. A head node v_h is only considered identified if the number of nodes without a clear partial order relation to v_h is less than the attacker's budget. In contrast, the more complex attack strategy introduced in Section 6.2 can compare any pair of nodes. A head node might thus be compared against nodes for which no clear order can be determined. Even so the head node v_h might thus not be detected as receiving chunks earlier for certain, the attacker might still sabotage v_h as it is an equally likely targets as other nodes. Thus, the theoretical bound is slightly lower than the actual result for $m > 1$. (ii) The attack's accuracy is improved with an increased q and m. Specifically, when $m = 1$, the attack's accuracy increases almost linearly with an increase of q. However, with $m > 1$ the increase in the attack's accuracy is larger for a smaller q, while the increase is less for a larger q. As we can see from the figure, the results agree with our expectation. There are, however, some interesting observations. First, if the attacker can probe buffer maps of all peers in the system, it can identify all head nodes accurately with only one probe. Second, when the probability to get a probed buffer map is small, probing several times can significantly improve the accuracy of the inference attacker.

In the coming section, we will investigate how the SWAP scheme helps improve the system's resilience by undermining the inference attacker.

Comparing the resilience of SWAP and DONet

In this experiment, we would like to answer the question: To which extent does SWAP increase the resilience of pull-based systems against the inference attacker? We compare our simulation results with the theoretical bound given by Eq. 6.8.

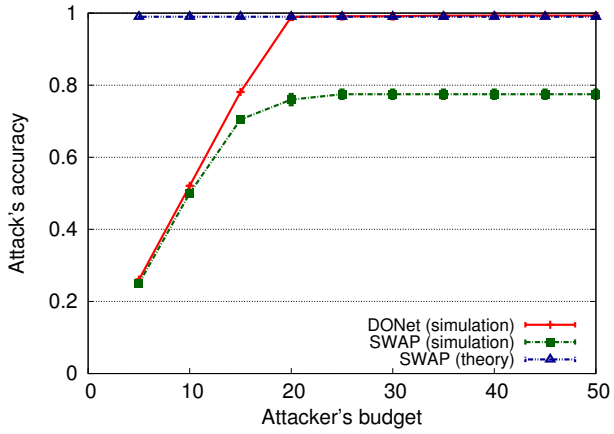
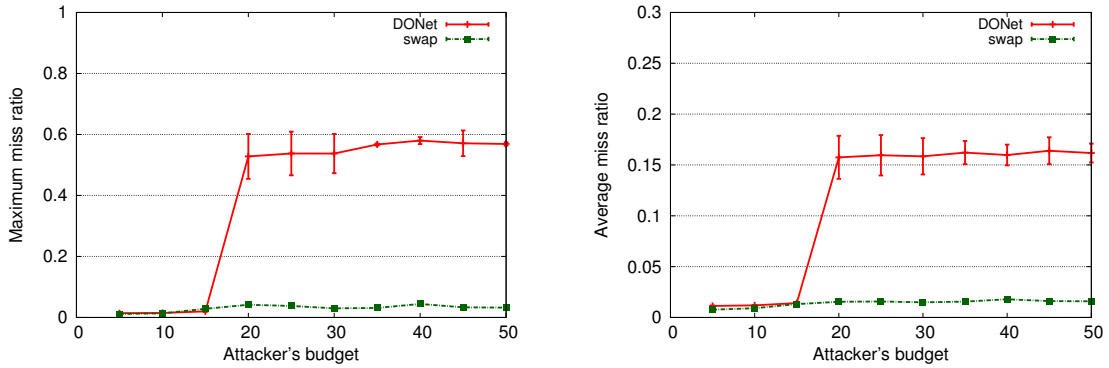


FIGURE 6.5: Comparing the attack accuracy when attacking DONet and SWAP, with respect to the attacker's budget

As plotted in Figure 6.5, the attack's accuracy increases drastically from around 0.25 to reach almost 0.8 and 0.7 in cases of DONet and SWAP respectively when x increases from 5 to 15. However, when $x \geqslant 20$, SWAP limits a stable attack's accuracy upto 0.8 while DONet loses most of its head nodes. Here, the theoretical upper bound is only of limited usefulness as it overestimates the attack's accuracy. For all parameter settings, the theoretical bound is close to 99 %. The reason for such a high difference between model and simulation lies in the generality of the model. The upper bound assumes that a new head node is detected as soon as it is first observed together with any of the previous candidate nodes. However, due to the new node's need to catch up to the other head nodes, the head nodes are generally identified much later than predicted. Nevertheless, the theoretical model is of interest as it gives a guaranteed upper bound, independent of the attacker's strategy. Even though a large fraction of head nodes of SWAP are correctly identified, the inference attacker still misses a significant fraction of them.

To explore the consequences of the attack's accuracy on system's resilience, we compare the chunk miss ratios caused by the attacks in both cases of DONet and SWAP. Subsequently, we collect the maximum and average chunk miss ratios. Results of both the SWAP scheme and DONet are plotted in Figure 6.6(a) and Figure 6.6(b), respectively.

When $x \geqslant 20$, the inference attacks cause extremely high damages on DONet with the maximum and average miss ratios of almost 60% and around 15% respectively.



(a) (b)

FIGURE 6.6: Comparing the resilience of SWAP to DONet against the inference attacks in terms of a) the maximum miss ratio and b) the average miss ratio, with respect to the attacker's budget

Whereas, under similar attacker's budget, SWAP's maximum and average miss ratios remain below 3% and 2% respectively. Those results are consistent with the attack's accuracy and can be explained as follows. When some of the head nodes are not attacked, either by lacking attacker's budget or by incorrectly being identified, they still connect the source and the remaining peers. The stream delivery flow is affected but remains connected. Therefore, the chunk miss ratios are very small. However, when all the head nodes are correctly attacked, the remaining peers are unable to obtain video chunks from the source, causing extremely high chunk miss ratios.

Impact of the number of forwarding on SWAP's performance

In this experiment, we would like to answer two questions: (*i*) How the SWAP scheme performs when there is no attack? and (*ii*) What are the costs of the SWAP scheme?

In all settings, we set the swap interval to 1.0 second and the number of swaps to 2 while the number of forwarding is set to 2, 4 and 6. The upload bandwidth of peers varies between 800 and 2000 kbps. We collected chunk loss ratios of all peers, as well as the source-to-peer latency to quantify the performance of the SWAP scheme. Furthermore, we also calculated the average head offset to understand the underlying reasons of the resulting performance. Finally, we collected and calculated the signaling overhead to estimate the costs of our SWAP scheme. We expect that an the SWAP scheme produce as much chunk loss ratio as the conventional DONet, since pull-based P2P streaming systems are highly tolerate to the dynamics of peers.

Nevertheless, we got interesting results. SWAP outperforms DONet in terms of chunk miss ratio. Figure 6.7 plots the average chunk miss ratio on dependence with

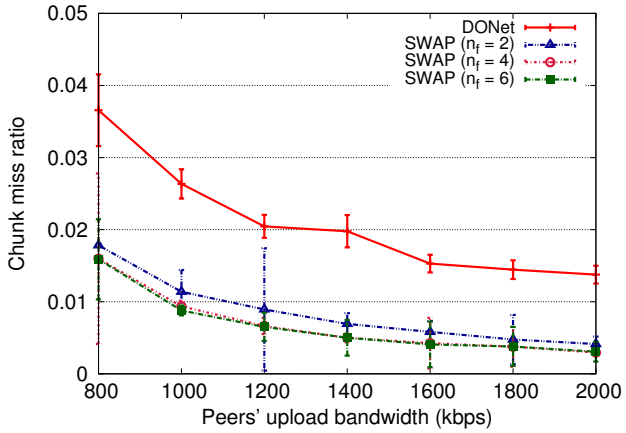


FIGURE 6.7: Comparing the performance of DONet and SWAP with different values of the number of forwarding, in terms of the average miss ratio, with respect to peers' upload bandwidth

the peer's upload bandwidth. While DONet loses 1.5% and 4% of chunks, SWAP loses less than 2% in all settings. This, however, can be explained as the following. Upstream peers with better downloading quality, when proactively connect to downstream ones can provide them more reliable connections. When downstream peers connect to random peers as in the conventional pull-based protocol, they more likely end up at another downstream one, since they are in large quantity as compared to the upstream peers. This suggests that by allowing an upstream peer to proactively connect to peers which are farther away from the source can help improve system's performance.

Latency

On the coming figures, we plot the source-to-peer latency and the average head offset on dependence with the upload bandwidth of peers. We expect that the SWAP scheme can reduce the source-to-peer latency, since more downstream peers have a chance to connect to and download video chunks directly from the upstream ones.

Figure 6.8 shows that the results agree with our expectation. The source-to-peer latency of the SWAP scheme is less than that of DONet in all settings. More interestingly, a conservative value of 4 for the number of forwarding n_f can already reduce the latency significantly. A larger value of n_f does not contribute to the reduction of the latency.

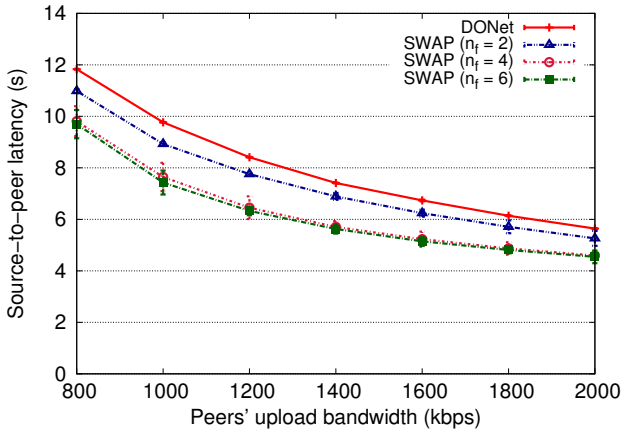


FIGURE 6.8: Latency of DONet and SWAP (with different values of n_f) with regard to peers' upload bandwidth, in benign scenarios

Cost of the SWAP scheme

Finally, we would like to answer the question: At which cost does SWAP performs in benign situations? Consequently, we discuss our finding on the cost of SWAP in terms of the signaling overhead in dependence with the peers' number of partners. The results are plotted on Figure 6.9. When the number of partners increases from 6 to 16, the signaling overhead of DONet ranges from 2.0% to almost 4.5%. On top of that, SWAP generates less than 1% extra overhead in each configuration. Considering the SWAP alone, a larger number of forwardings n_f reduces the overhead. This is due to the fact that, the smaller the value of n_f the more peers are able to trigger swap operations, resulting in more dynamics and signaling overhead in the system.

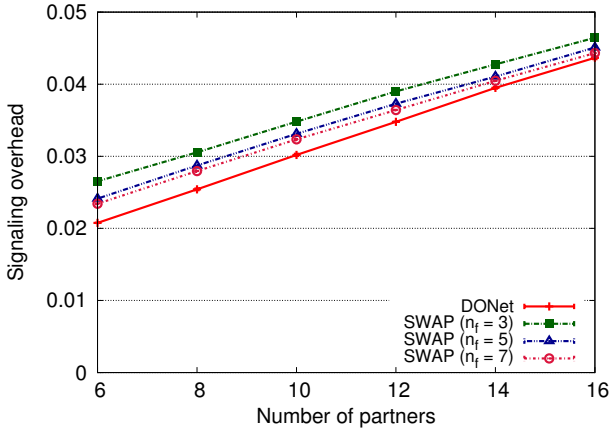


FIGURE 6.9: Signaling overhead of DONet and SWAP (with different values of n_f) with regard to the number of partners, in benign scenarios

We have presented the evaluation of our SWAP as a countermeasure to the in-

ference attacker. Extensive simulation results show that the SWAP scheme reduces both maximum and average chunk miss ratios drastically as compare to DONet, the conventional pull-based system. Furthermore, the SWAP maintains a lowed average chunk miss ratio and lower source-to-peer latency in scenarios without attacks. In return, SWAP induces slightly more signaling overhead due to the increased dynamics of peers. However, the overhead is well below 4% even with a reasonably large number of forwarding.

6.6 Summary

DoS attacks on head nodes can affect a pull-based system drastically in terms of maximum and mean chunk miss ratios. Unfortunately, head nodes can be identified accurately and rather easily by an inference attacker. The attacker collects buffer maps (BMs) from peers and uses the information contained in those BMs to infer the overlay structure of the system, including the head nodes.

To mitigate the attacks, our approach is to undermine the attacker's capability in identifying accurately head nodes. Consequently, this chapter starts by modeling the inference attacker. In this attacker model, the attacker probes the system, meaning to collect BMs from peers, several times before attacking it. The attacker distinguishes peers by the average offset metric. The metric is used in the literature to successfully infer the overlay structure of PPLive, a running pull-based system. Afterwards, the attacker selects peers with the highest averaged offsets to shut down. Subsequently, we identify important parameters of the attacker model, which influence the attack's accuracy. They are the number of probes m and the probability q that the attacker can get a buffer map from a peer. Finally, we formally derive the lower bound of the attack's accuracy for the inference attacker. We use this bound to validate the simulation model of the attacker.

Our extensive simulation studies suggest that an inference attacker can accurately identify head nodes. Interestingly, if the attacker can always get the buffer maps from active peers in the system, it can identify all head nodes using only one probe. However, if the probability q is small, increasing the number of probes m can significantly increase the attack's accuracy in identifying head nodes.

To undermine the attack's accuracy in identify head nodes, we introduce SWAP. The scheme's idea is to enforce peers to proactively change their partners. The swap operation needs to happen periodically to prepare for attacks that can happen at an unknown instance. In each swap cycle, each peer drops a few of its partners and connects with their replacement peers. To minimize the negative impact of the swap operation, peers should have a strategy in selecting dropped partners. Without global knowledge, peers have to use the best out of its local knowledge to decide. Fortunately, the average offsets calculated from a peer's received buffer maps provides

it a relative position of the peer and its partners as compared to the source. Using the average offset metric, we introduce the concepts of upstream and downstream partners. Subsequently, downstream partners should be selected for dropped and replacement partners. The peer sends its nomination containing the replacement partner to all upstream partners. For peers to avoid swapping locally with their partners, thus, to avoid possible collusion between malicious peers to eclipse the system. To overcome this issue, the nominations are forwarded several times. This means that a current partner should be swapped with a peer that positions farther in the downstream. This also allows peers to actively decide alternative partners in a more structured manner.

The SWAP scheme is assessed by extensive simulation studies. Simulation results reveal the following remarkable findings. (*i*) The SWAP scheme effectively undermines the attack's accuracy in identifying head nodes. As the result, SWAP effectively reduces both the maximum and average chunk miss ratios drastically. (*ii*) In benign scenarios, SWAP reduces chunk miss ratio and source-to-peer latency. (*iii*) However, SWAP causes extra signaling overhead due to the the nomination forwarding and additional churn introduced by swap operations. Nevertheless, the signaling overhead only increase slightly and remains well below 5% of the total traffic.

At this point we conclude our chapter on the SWAP scheme as a countermeasure against the inference attackers. We also conclude our measures to counter the DoS attacks in pull-based P2P streaming systems. In the next chapter, we present our study on the resilience of hybrid P2P streaming systems against DoS attacks.

7

Resilient backbone construction

This chapter addresses the problem of improving system's performance while at the same time maintaining the resilience of P2P streaming systems. A brief motivation will be followed by the description of the resilient backbone construction scheme. This section elaborates the idea, design and specification of the scheme respectively. The evaluation will continue the chapter with simulation setup and key findings. Finally, we conclude the chapter with the a summary.

7.1 Motivation

Hybrid P2P streaming systems, such as mTreebone [68], have evolved recently. These systems combine the advantages of an efficient push-based with a more resilient pull-based content dissemination. In this manner, they offer low latency as well as an increased robustness to failures as a result of node churn. However, the overlay construction of current hybrid systems is vulnerable to misbehaving nodes and deliberate attacks. By taking central positions in the hybrid overlay, malicious nodes can perform extremely harmful Denial-of-Service (DoS) attacks.

In this chapter, we propose a novel backbone construction scheme for hybrid P2P streaming that is highly resilient against DoS attacks while maintaining similarly fast content dissemination as current state-of-the-art streaming systems. Our construction relies on two steps. First, stable peers are securely identified by using only *local knowledge* about the participation time of other peers. Second, those peers are incorporated into a manipulation-resistant multiple-tree backbone overlay, which is resilient against both attacks and node churn. Extensive simulation experiments indicate that our scheme outperforms the state-of-the-art in being more resilient against

attacks at the price of a slightly increased overhead. Specifically, three sub-problems are identified: (*i*) minimizing information leak that allows a malicious party to infer the structure of the overlay to attack the system, (*ii*) constructing a resilient backbone against DoS attacks, and (*iii*) efficiently coordinate the reserved bandwidth for pull and push operations.

7.2 System Design

In this section, we introduce RBCS, our hybrid P2P streaming system. The proposed system combines a general pull-based system with a multiple-tree push-based approach used only by the *stable peers* making up the backbone. Additionally, we introduce our *invitation-based* scheme to replace the self-promotion scheme, which both enables selfish behavior potentially decreasing the resilience of the system and facilitates large-scale attacks. We first informally discuss our design, then give a formal system model, before discussing potential problems and limitations.

7.2.1 Idea

We integrate a backbone of stable nodes in the form of a push-based multiple-tree into the pull-based mesh. The key components of the stable backbone are *i*) the identification of stable peers and *ii*) the construction of the multiple-tree for content dissemination between stable peers.

Our idea for identifying new stable peers is to allow existing stable nodes to continuously calculate the connection durations of their partners. The decision if a node is invited into the set of stable peers is hence based on *local observations* only. Our belief that such *local observations* are sufficient to detect stable nodes despite the frequent changes of partners is based on studies from Wang et al. [68, 67]. They show peers usually keep a core of partners which are hardly dropped, so that indeed partners are observed for a sufficiently long time to determine if they are stable. Partners whose peering durations exceed a certain threshold are invited to join the backbone consisting of stable peers. The backbone evolves gradually as more peers join the system and are added to the backbone. Initially, the source is the first stable node in the backbone. It then invites more stable nodes to join the backbone. These stable ones, in turn, start observing the peering durations of their partners and invite partners if they have been connected for long enough.

We choose a multiple-tree scheme for the stable backbone, multiple-tree schemes being the push-based schemes with the highest resilience. The scheme *magnifies* the connectivity between peers in the backbone, more importantly, between peers and the source. It is therefore harder to disrupt the backbone. One direct concern is on the cost of fixing the topology in a multiple-tree overlay when peers join,

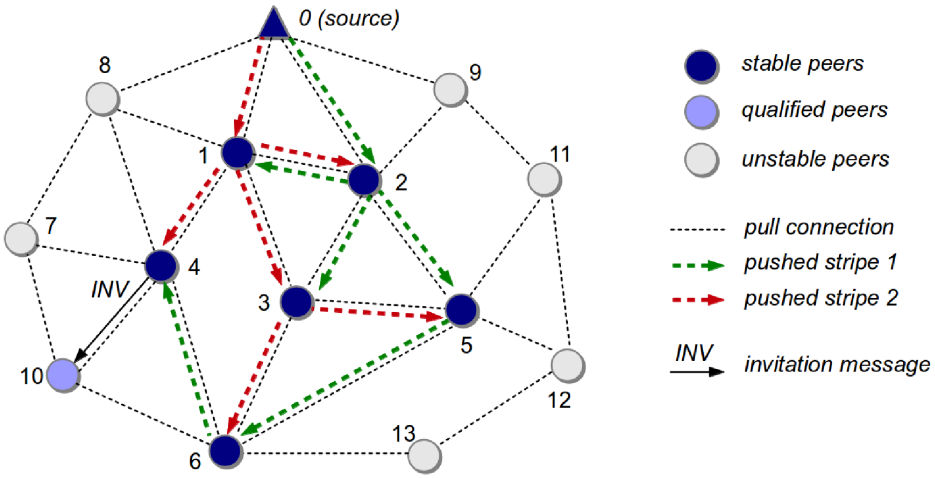


FIGURE 7.1: The overlay construction of RBCS: All peers are inter-connected in a pull-based overlay. The stable peers form a push-based backbone with two stripes. The backbone gradually expands by inviting newly qualified stable peers. Each number denotes the join time of the respective peer

leave or are attacked. However, the problem might not be as serious as it seems for the following reasons: *First*, the multiple-tree backbone consists of more stable peers whose dynamics are probably less frequent. *Second*, since the multiple-tree backbone is maintained by stable peers only, the maintenance cost of the backbone should be small. *Third*, the overlays in later multiple-tree protocols are built in such a way that a peer which is an inner node in one tree is a leaf node in all other trees. Those significantly reduce the topology-repairing cost when an inner node fails or is attacked. Our abstract design allows for any multiple-tree scheme, our implementation is based on the Optimally Stable Topology (OST) [15], as detailed in Section 7.2.3. We illustrate the above strategy in Figure 7.1, depicting how stable peers in a pull-based overlay form a backbone. It builds two spanning trees to deliver the two stripes of the video stream. The backbone gradually expands when the stable peers *invite* newly qualified ones.

7.2.2 Formalization of Invitation Scheme

We now formalize the evolution of the stable backbone, identifying the components governing the time until a node is invited. Formal descriptions of the pull-based system and multiple-tree systems are available in [82] and in [15], respectively. Here, the different functionalities, such as the time nodes have to remain connected, are only described in an abstract fashion, a realization is discussed in Section 7.2.3. The formalization of the invitation scheme is based on the model the *partner selection and management*, which was presented in Chapter 4, because it depends on the topology

and its management.

In the following, we use $\inf A$ to denote the infimum, i.e., the greatest lower bound on A, and $\sup A$ to denote the supremum, the lowest upper bound on A, for a set A of real numbers.

As introduced in Section 4.1, each peer $v \in V_t$ has a *partner set* $N_t(v)$ of peers it can exchange content with at time t. We restrict the number of partners to be at most M in agreement with common pull-based streaming systems. When joining, each peer v is informed of the session duration T. Now, assume that v has a partner u at time t. The duration of v and u's partnership at time t is given by $len_t(v,u) = \sup\{t - \tau : \forall z \geqslant \tau, u \in N_t(v)\}$. The quantity $len_t(v,u)$ is determined by the time peers remain in the system and the partner refinement strategy R. We adapt the refinement strategy of DONet [82]: Each s seconds new partners are suggested. If the peer already has M partners, one partner is dropped and the most suitable of the suggested partners according to some *neighbor selection criterion* L is added. For example, L can specify to select a neighbor randomly or based on some reputation scheme implemented within the system. Otherwise, if the peer has less than M partners, multiple peers can be added as partners and none of the existing partners are dropped. The decision which partner to drop depends on the exchanged traffic during the last y seconds. Let $download_t(v,u)$ be the download rate from partner u to v at time t, and correspondingly $upload_t(v,u)$ the upload rate to partner u. Then the exchanged traffic during the last y seconds is given by

$$perf_t(v,u) = \int_{t-y}^{t} [download_x(v,u) + upload_x(v,u)]dx.$$

The partner with lowest performance $perf_t(v,u)$ is dropped, considering only partners the peer has been connected to for at least y seconds. Basing the partner selection on the exchanged traffic and periodically replacing partners has been shown to be highly effective in pull-based streaming, hence we refrain from changing it in favor of more stable partnerships.

Invitation into the set of stable nodes now works as follows: The time for a node to be invited to join the backbone is governed by a function $w(t)$. The function w should satisfy that for any $t_1 > t_2$, $t_1 + w(t_1) > t_2 + w(t_2)$, i.e., an earlier obtained partner is also invited into the set of stable peers sooner. When a peer v becomes a partner of another peer u at time t_0, it is invited at time $t_s = t_0 + w(t_0)$ given that (i) v and u have been partners from time t_0 to t_s; (ii) u is a stable peer at time t_s; and (iii) u is able to integrate v into the multiple-tree. Depending on the multiple-tree topology, integration might not always be possible. We denote by $poss_{t_s}(v,u)$ the fact that v can be integrated into the scheme by u at time t_s. Note that the time t_0 of v adding a partner u can be expressed in terms of the current time t as $t_0 = t - len_t(v,u)$, the difference of current time and the time the peers have been partners. Let $s(v) \in \mathbb{R}_+ \cup \{\infty\}$ be the time a peer v becomes stable. By the above,

$s(v)$ is given by

$$s(v) = \inf\{t \geqslant 0 : \exists u \in N_t(v), len_t(v,u) \geqslant w(t - len_t(v,u)) \\ \wedge\, s(u) \wedge poss_t(v,u)\}, \tag{7.1}$$

the first point in time that any stable partner has been connected long enough to invite the peer to the set of stable peers. In other words, when a peer v connects to a peer u at time t_0, v is invited into the set of stable peers at time $t_s = t_0 + w(t_0)$ if the following five conditions hold:

- u has to be a stable peer at time t_s
- u has to stay online until at least time t_s
- u does not drop v until at least time t_s
- v does not drop u until at least time t_s
- u has to be able to integrate v into the multiple-tree

Clearly, $s(v)$ is not solely depending on a node's join time, as is the case for the current self-promotion scheme. Rather, $s(v)$ depends on the stability of a node's neighbors and their willingness to keep v as a partner. By considering v's suitability as a partner, the time until invitation is influenced by download and upload bandwidths, preventing nodes with a low performance to join the backbone. In this manner, nodes in the backbone are expected to be not only stable but also highly performant, thus achieving a fast as well as reliable content distribution along the backbone. Note that remaining online for a long period does not guarantee invitation into the backbone. If the node is unable to connect to already stable nodes for a sufficiently long time, it will not be invited to join. Therefore, it is not evident that a stable backbone indeed evolves. Deriving theoretical bounds on the time for node to become stable and for a backbone to evolve is complicated by the dependence of the above five conditions. Furthermore, the evolution time depends on various features, such as the online time distribution S, the upload and download distribution, the session duration, and parameters, such as the number of partners M, the refinement periods s and y, and the function w determining the required time to become stable. So, we choose to perform a simulation study, allowing us to model all of the above component of the complex system, rather than neglecting any of these factors to obtain insufficiently accurate theoretical bounds.

7.2.3 Realization

The choice of the function w, determining the required duration for being invited, and the neighbor selection criterion L, used for choosing new partners, are essential for evolution of the backbone and hence the performance of the system. Furthermore,

we specify the multiple-tree scheme used in our evaluation, and discuss its non-trivial integration into the push-based overlay. In particular, the flag $poss_t(v, u)$, stating if a peer v can be added to the backbone by a peer u without violating the constraints of the scheme, requires careful consideration. Finally, we discuss the coordination between the pull and push operations.

Invitation scheme In mTreebone, the function is chosen as $w(t) = f(T - t)$ for $f = 0.3$, i.e., a peer becomes stable after it has been online for 30% of the remaining session length. This means that the stable backbone first evolves slowly until a very stable core has formed, then peers at the lower levels of the tree are added more quickly. We will vary the factor f to counteract the fact that $w(t)$ is only a lower bound on the time to become stable in our scheme. For the criterion L, most pulled-based systems prefer a random selection to achieve diverse meshes.

Multiple-tree backbone For the construction of the multiple-tree backbone, we select the Optimally Stable Topology (OST) scheme [15] to make the backbone overlay highly resilient for multiple reasons: *First*, OST strives to minimize the dependency of a peer on a single predecessor. As a consequence, it is guaranteed that a minimum number of stripes is affected by an individual peer failure or shut-down. *Second*, trees of a high depth are avoided, leading to a fast propagation of content and a higher chance of being unaffected by predecessors due to failures or attacks. *Third*, OST achieves a global optimum based on local knowledge only, preventing attackers from gathering information about the overall topology and identifying influential peers. More details about the system can be found in [15]. Integrating the OST scheme into a pull-based system requires some slight adaptations: The original OST scheme cannot be used immediately, most notably due to the availability of knowledge about other stable peers. More precisely, a newly invited peer is only aware of a few stable peers in the backbone due to its purely local view of the network. The options of a peer to join the backbone and recover from isolation due to the leaving of its parents is drastically impacted by this restricted view. Here, isolation refers to the absence of any known nodes within the multiple-tree approach, potentially resulting from parents leaving the system. Therefore, we modify the scheme slightly as follows:

1. A currently stable peer sends an invitation to a qualified partner only if it can accept at least one more child in one of the stripes that it is forwarding. Hence, the function $poss(v, u)$ is defined to return true in case there exists a stripe for which u can accept another child and zero otherwise.

2. When a peer v receives an invitation from a peer u, v sends join requests for all stripes to u. If only a subset of the requested stripes can be served, u replies suggesting alternative parents for v to connect to.

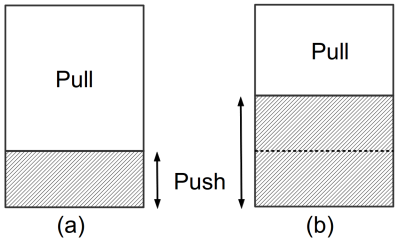


FIGURE 7.2: Coordination of the dedicated bandwidth for the pull and push operations: a) Less bandwidth for push when the peer has less children; b) More bandwidth for push when the peer has more children

3. During the online duration of a peer v, v might receive several invitations but only needs the earliest one to join the backbone overlay. The contact information of the remaining inviting peers are stored for later use, e.g., when recovering from isolation. If at the event of isolation, none of the stored partners can accept v as a child, v waits until it receives a new invitation, temporarily losing its stable status.

Coordination between push and pull functions A hybrid streaming system requires coordinating between the pull-based mesh overlay and the push-based multiple-tree overlay. In particular, the bandwidth dedicated to each overlay needs to be specified. As a minimum, the trees can be maintained if the dedicated bandwidth equals the bandwidth required by one video stream. However, this restriction entails disadvantageously high trees, diminishing the system's performance due to high latency and low resilience. On the other extreme, reserving too much bandwidth for the push operation would leave the pull operation with insufficient bandwidth, resulting in congestion. This problem is especially severe in the presence of missed chunks due to leaving or attacked neighbors.

To solve the bandwidth coordination issue, we introduce an individual flexible bandwidth reservation scheme. Let $bw_{pull}(t)$ and $bw_{push}(t)$ be the bandwidth portions for pull and push operations at time t respectively. Then, the bandwidth $bw_{push}(t)$ is restricted to be at most $bw_{max} \leqslant bw_{up}$, whereas the bandwidth $bw_{pull}(t) = bw_{up} - bw_{push}(t)$. Before a peer is invited into the backbone at time t_S, it uses the whole upload bandwidth for the pull operation, i.e., $bw_{push}(t) = 0$ and $bw_{pull}(t) = bw_{up}$ while $t \leqslant t_S$. After joining the backbone, the peer increases $bw_{push}(t)$ as it receives subscription requests. However, children are only accepted as long as $bw_{push}(t) \leqslant bw_{max}$. If its number of children in the tree declines, the peer releases the reserved bandwidth $bw_{push}(t)$ to increase the bandwidth $bw_{pull}(t)$. The coordination of the bandwidth is illustrated in Figure 7.2.

We have now introduced our system, presenting both the high-level abstract model and a concrete realization. The key concern is to maintain a distribution of

time $s(v)$ for a peer to be invited such that a stable backbone indeed involves. In the following, we see that for a carefully selected set of parameters, we can indeed have a stable backbone, while at the same time eliminating the drawbacks created by self-promotion.

7.3 Evaluation

In this section, we assess how effectively our RBCS scheme can mitigate the damage caused by the attacker with a greedy attacking strategy (or greedy attacker, for short) in comparison to the state-of-the-art hybrid system mTreebone. In particular, we focus on three questions: (*i*) What is the damage caused by an attack at different attacker's budgets? (*ii*) What is the impact of the peers upload capacity on the attacker damage? (*iii*) How much signaling overhead is introduced by RBCS compared to mTreebone?

In our experiments, we use the following parameters: The video stream with the rate of 400 kbps is divided into equally sized chunks of size 2500 bytes. The video buffer at every peer stores up to 30 seconds of video chunks. A peer starts playing out video chunks when the downloaded chunks are equivalent to around six seconds. The upload bandwidth of the source is $bw_{up}(s) = 8Mbps$ in all experiments. The upload bandwidth of peers is chosen homogeneous to allow for characterize the impact of bandwidth on the attacker performance, using $bw_{up}(u) = 2Mbps$ for all nodes $u \neq s$. The download bandwidth of peers is always chosen as $bw_{down}(u) = 2.5Mbps$. Furthermore, the number of partners M is 12. Each stable peers dedicates two times the streaming bit-rate to forward chunks to others in the backbone.

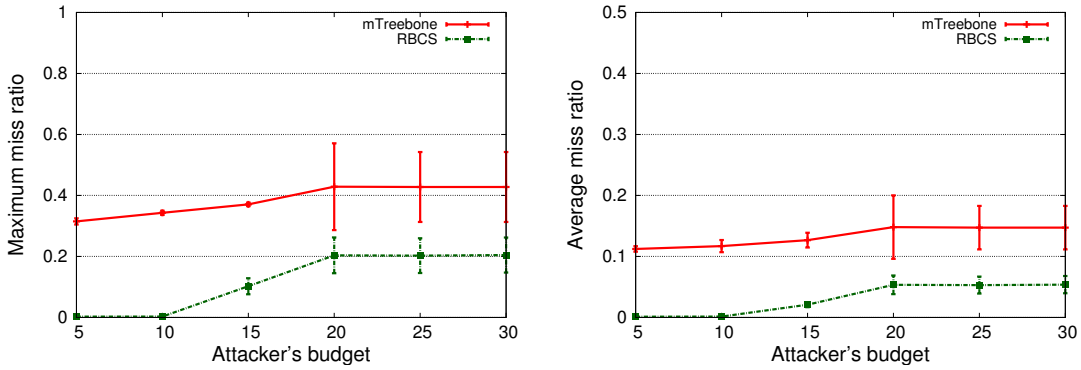
We simulate a system with a total of 2000 peers using one source node for the duration of $T = 3000$ seconds. The attack takes place at 1500 seconds, allowing the system to evolve first, before considering the reaction to an attack. The attacker shuts down nodes after roughly half of the session duration passes. This ensures that some stable peers already join and maintain the backbone. We vary the attacker's budget x between $5, 10, 15, 20, 25$, and 30. All simulation results are averaged over 30 runs.

7.3.1 Results

We now present our simulation results concerning the three research questions.

Impacts of the attacker's budget

We expect the chunk miss ratio to increase with the attacker budget. Figure 7.3(a) and Figure 7.3(b) show the resulting chunk miss ratios with regard to the attacker's



(a) (b)

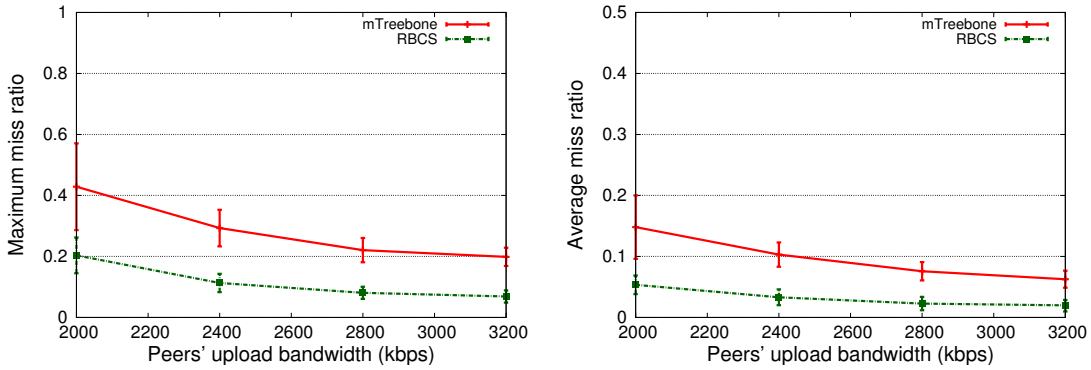
FIGURE 7.3: Impact of attacker's budget on (a) maximum chunk miss ratio and (b) average chunk miss ratio to mTreebone and RBCS

budget for both mTreebone and RBCS schemes. Indeed, both the maximum and the average chunk miss ratios increase with the attacker's budget. There is a strong increase between 10 and 20, after that the caused damage remains nearly constant. In particular, the miss ratio in mTreebone is not negligible because a miss ratio of this extent usually affects consecutive chunks and thus severely reduces the users' quality of experience. RBCS peers miss 10% less chunks than mTreebone, with the same attacker's budget, because RBCS peers have more connections to other backbone peers. When the source is isolated, because all of its neighbors are removed, some RBCS peers are still able to reconnect to the source using previous invitations, which helps RBCS recover more quickly than mTreebone.

Impacts of peers upload bandwidth

In this experiment, we evaluate how the peers' upload capacity can mitigate the negative impact of the greedy attacker to mTreebone and our RBCS scheme. We consider an attacker budget of $x = 20$. We expect that the more upload bandwidth each peer has the better the system can recover from disruption. Due to the larger upload bandwidth, one peer can serve more partners, increasing the chance of finding a partner able to provide the desired chunk. When peers have a large upload bandwidth, missing chunks can be delivered quickly before their play-out deadlines.

It can be seen from the Figure 7.4(a) and Figure 7.4(b) that an increase of the upload bandwidth reduces chunk miss ratios drastically for mTreebone. So, the maximum miss ratio reduces from 40% down to around 20%, whereas the average miss ratio reduces from almost 15% down to around 5%. For RBCS, the average chunk miss ratios are well below 10% in all settings. These results confirm our expectation that RBCS is highly resilient to attacks due to its multiple-tree topology. It distributes the load more balanced, positioning more nodes close to the source than



(a) (b)

FIGURE 7.4: Impact of peers' upload bandwidth on (a) maximum chunk miss ratio and (b) average chunk miss ratio to mTreebone and RBCS

mTreebone, even with less dedicated bandwidth for the backbone.

Signaling Overhead

In this experiment, we are interested in how much additional signaling overhead the RBCS scheme introduces. Both mTreebone and our RBCS scheme are evaluated without attacks. We vary the number of partners m from 8 to 24. For the RBCS scheme, the number of stripes are varied between 2 and 4. We expect that RBCS introduces additional signaling overhead due to the maintenance of the multiple-tree overlay for the backbone. Figure 7.5 shows the resulting signaling overhead in dependence on peers' number of partners for both mTreebone and RBCS schemes. In all cases, the signaling overhead increases with the number of partners and stripes. However, the increase in overhead for each additional stripe is less than one percent, so that RBCS is only slightly more costly than mTreebone using only one stripe.

In summary, RBCS offers an increased resilience to attacks at the price of a modest additional signaling overhead.

7.4 Summary

This chapter tackles the problem of providing both resilience to DoS attacks and high performance in terms of low latency for pull-based P2P streaming systems. Our approach is to securely identify and efficiently utilize stable peers, which are long-staying ones in the systems. Those stable peers form a backbone of the overlay, which disseminates majority of video chunks with low latency. Specifically, three sub-problems are identified: (*i*) minimizing information leak that allows a malicious party to infer the structure of the overlay to attack the system, (*ii*) constructing a

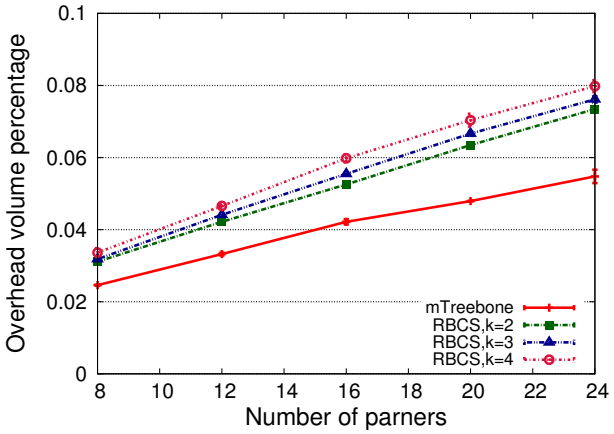


FIGURE 7.5: Dependence of signaling overhead to the number of partners in mTreebone and RBCS with 2, 3, and 4 stripes

resilient backbone against DoS attacks, and (*iii*) efficiently coordinate the reserved bandwidth for pull and push operations.

Subsequently we present RBCS, a resilient backbone construction scheme for hybrid P2P streaming. The scheme consists of the following three parts. First, RBCS incorporates a local manipulation-resistant stabilization process, which prevents an attacker from quickly learning much about the overlay structure of our system, thus considerably reducing its chances to attack the most relevant nodes. Second, the scheme constructs an attack-resistant multiple-tree backbone, which is proven to be resistant to the optimal attack with global knowledge. Third, RBCS introduces a coordination strategy to seamlessly assign bandwidth to pull and push operations. A minimum bandwidth is reserved for pull operation to ensure effective pull operations and the connectivity of the underlying mesh topology. As soon as there is unused bandwidth from the push operation, due to e.g., a disconnecting child node, that bandwidth is used for pull operations.

We evaluate our solution via extensive simulation studies. The results show that RBCS achieves a considerably higher attack resistance than the state-of-the-art. Moreover, our scheme uses significantly less bandwidth to achieve comparable performance as the state-of-the-art. The cost of RBCS is the increased overhead due to signaling messages resulted from maintaining multiple trees at the backbone. However, the overall signaling overhead only constitutes a slight increase.

8

Conclusion and Outlook

This chapter reviews the problem of streaming videos over the Internet in general and the resilience of P2P streaming systems against DoS attacks in particular. Furthermore, the chapter recaps the contributions of this thesis. Thus, Section 8.1 summarizes the developed schemes and discusses the main evaluation results of the schemes and provides subsequent insights into P2P video streaming systems. Finally, Section 8.2 sketches future work that can be directly extended from this thesis.

8.1 Summary

The consistently increasing demand to watch live events over the Internet has placed video streaming systems under a high bandwidth demand to satisfy large audiences. The Peer-to-Peer (P2P) streaming architecture mitigates the bandwidth demand at the streaming source by incorporating peers bandwidth in the delivery of video streams. To do so, a P2P streaming system needs to build an overlay interconnecting peers. However, the overlay can be broken by churn, meaning the leaving and failing of peers. Disconnected peers from the overlay are unable to receive the stream any more. Furthermore, the dependents of the disconnected peers cannot receive the stream either. Therefore, the challenge of P2P streaming systems is to provide peers a continuous stream with a low latency and a low signaling overhead by constructing an overlay resilient to both churn and Denial-of-Service (DoS) attacks.

The pull-based systems have demonstrated their robustness to churn with several real-world working deployments to date. A pull-based system establishes a mesh topology, providing peers with multiple paths to the source. To obtain the stream, peers need to frequently request video chunks from their partners. To do so, peers

need to periodically exchange buffer maps (BMs) which are small packets containing the availability information of the peers' video buffers. Pull-based systems are robust to churn because even when a peer loses some partners, due to churn, it still can request video chunks from its remaining partners. Nevertheless, the common drawback of pull-based systems is the high latency and the high signaling overhead because of the frequent buffer map exchange and the chunk requests. Because of that hybrid systems are proposed. A hybrid system bootstraps from a pull-based one and subsequently builds a backbone to deliver chunks without requests. The backbone consists of a small number of stable peers which already stay sufficiently long in the system.

Unfortunately, both classes of pull-based and hybrid systems are vulnerable to DoS attacks. First, pull-based systems are exposed to attacks on head nodes (or the source's partners). Attacks by shutting down those nodes can disrupt the delivery of the video stream from the source to the remaining peers, drastically affecting the system. More critically, those head nodes can be identified accurately and pretty easily by an inference attacker. Such an attacker collects peers' buffer maps and infers the system's overlay structure including head nodes. Second, a hybrid system is vulnerable to DoS attacks on its backbone. The backbone of a hybrid system has a single-tree topology, thus, fragile to attacks. Since the backbone delivers majority of video chunks, disrupting the stream delivery at the backbone can badly affect the whole system. Furthermore, the current strategy to discover stable peers for the backbone is insecure because it allows peers to promote themselves. This self promotion allows an adversary to easily obtain the list of stable peers to attack the system.

This thesis addresses the above two vulnerabilities of pull-based and hybrid systems. We strive to mitigate the attacks on head nodes first because that is the common problem of both classes of systems. Subsequently, we address the first problem by two contributions: mitigating the attacks even when the adversary knows all the head nodes and preventing the adversary from accurately identifying head nodes in the first place. After that, we mitigate the attacks on the backbone of hybrid systems.

To mitigate the damage caused by attacks on head nodes even if the attacker has a global knowledge, we develop the striping scheme. Our approach is to increase the number of partners each peer has, consequently increasing the number of head nodes. Therefore, the system has more chance to maintain its connections between the source and the peers under attacks, thus, reducing the negative impacts of the attacks. However, when having more partners, a peer might overload itself by chunk requests from its partners. Because of that the striping scheme enforces peers to diversify their chunk requests. To do so, we introduce two techniques: First, the video stream is divided into k stripes, which are partial streams. Second, each peer assigns its partners into k groups. Consequently, in each scheduling cycle, every peer

has to request chunks of a particular stripe from the respective group of partners only.

To further mitigate the destructive impacts of the attacks on head nodes, we strive to undermine the capability of the inference attacker in accurately identifying head nodes in the first place. Subsequently, we assume a practical attacker with local knowledge and build the attacker model. Before attacking the system, the inference attacker probes peers to collect their buffer maps and then uses them to infer the overlay structure of the system. To counter the inference attacker, our idea is to enforce peers to regularly and proactively change their partners. As the result, the attacker is unable to accurately identify head nodes any more. Consequently, we introduce the following three primitives: First, to prepare for the swapping operations, every peer should suggest its replacement candidates. Furthermore, the nominations should be forwarded several times, allowing peers to connect to more diverse new partners. Second, to prevent the nomination message from being infinitely forwarded, we introduce a counter. Third, to be ready for attacks, peers periodically replace a subset of their partners by the respectively nominated ones.

After strengthening the resilience of pull-based systems to DoS attacks, we address the vulnerability of hybrid systems. Specifically, to protect hybrid systems from the attacks on the backbone, we develop the Resilient Backbone Construction Scheme (RBCS). The backbone only includes stable peers that stay sufficiently long in the system. The scheme is built from three primitives. First, to identify stable peers, peers use their local observation only. If a stable peer is discovered it is then invited to join the backbone. By using invitations, the system does not reveal peers in the backbone, thus, lowers the risks of attacks on the backbone. Second, the backbone employs the Optimally Stable Topology (OST) to include peers into a multiple-tree overlay. OST has theoretically proven its resilience to DoS attacks, thus, strengthening the backbone against attacks. Third, RBCS introduces an adaptive bandwidth reservation scheme to flexibly coordinate the bandwidth of peers when they push chunks in the backbone and respond to chunk requests at the same time.

The evaluation of the developed schemes requires a framework that allows us to compare fairly different P2P streaming systems. Since such a framework has not been available so far, we develop OSSim, our general-purpose simulation framework for P2P video streaming. The framework allows for simulating all classes of P2P streaming systems. Specifically, representatives of each class of systems are instantiated, namely DONet, OST and mTreebone for pull-based, push-based and hybrid respectively. We validate the implementations of those systems by comparing the simulation results with the published ones.

On top of OSSim, we implemented our striping, SWAP and RBCS schemes. In addition to that, we develop and implement several attacker models and novel resilience metrics. We conduct extensive simulation studies on the developed schemes.

The results show that those schemes effectively mitigate the damage of DoS attacks. In benign scenarios, the developed schemes also decrease the source-to-peer latency while only increase the signaling overhead slightly.

8.2 Future work

This section discusses the future work subsequent to this thesis.

First of all, a combination of the developed schemes in this thesis should be studied. Potentially, this allows for a concrete hybrid P2P streaming system which is resilient to both DoS attacks on head nodes and DoS attacks on the backbone. Toward this goal, the impacts of combining the striping and the SWAP schemes require a further investigation. However, we believe that the integration would be rather straightforwards since both schemes are complementary to one another. The striping scheme increases the number of partners each peer has. The increased number of partners allows the SWAP scheme to swap more partners in each swap cycle. The SWAP scheme also magnifies the benefits of the striping scheme. By swapping partners, the system might not require as many head nodes to mitigate the DoS attacks as the striping does alone. This reduces the signaling overhead of the system. Nevertheless, more comprehensive study on combining the string and the SWAP scheme is required to understand the resilience gain and the respective cost.

After that, it is interesting to integrate RBCS to work on top of the above two schemes. Among those, combining RBCS and SWAP can potentially be more challenging. SWAP introduces more dynamics to the system by enforcing peers to change their partners while RBCS prioritizes stable partners whose partnership span for sufficient durations. Combining the two seemingly contradicting strategies into one system can be very hard. However, we have a few initial ideas for the problem. For the SWAP scheme, we can modify the swap strategy so that stable partners are less likely to be selected for swapping. This helps the RBCS in identifying stable peers for the backbone. At the RBCS scheme, it is worth to try the swapping peers inside the backbone trees. We expect that the OST can tolerate a certain extent of peers' dynamics. Nonetheless, comprehensive studies should be conducted to assess the modified strategies and possibly to find the tradeoffs between the resilience and the performance of the system.

Second, the question is to re-examine the fundamental implications of key system-wide parameters, such as the number of partners. When diversification enforcement is used via the striping scheme, the partners of a peers is divided into several groups. A sufficient number of partners is required to allow for partner grouping and ensuring the selection of partners in each group to request video chunks. A large number of partners also allows for several source-to-peer paths, which increase the connectivity between peers and consequently increase the resilience of the system. On the other

hand, more partners also increase the signaling overhead drastically due to buffer map exchange. A study on the dependency of the system's performance with regard to the number of partners and consequently to the number of stripes would help understand an optimal value of such a system-wide parameter and provide guidelines for the system design.

Third, the impacts of the inference attacker on pull-based systems with locality awareness need to be examined. When peers prefer to select close-by ones as their partners, the system's performance can be improved and the overall traffic demand to the core network can be reduced. However, the system can also become more clustered with more connections within each cluster while there are less connections between the clusters. Consequently, the system can become more fragile and easily fragmented into isolated segments by attacks. A direct follow-up step from our work is to examine the possibility to incorporate the locality information into our developed mechanisms, including the partner selection process, diversification enforcement, swapping and resilient backbone construction schemes. The challenge is how to incorporate the locality information into the those schemes in order to optimize the system's performance and resilience at the same time.

Bibliography

[1] Abeni, L., Kiraly, C., Lo Cigno, R.: Sssim: a simple and scalable simulator for p2p streaming systems. In: IEEE 14th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks. pp. 1–6 (2009)

[2] Alstrup, S., Rauhe, T.: Introducing octoshape-a new technology for largescale streaming over the internet. EBU technical review 303 (2005)

[3] Apostolopoulos, J.G., Tan, W.t., Wee, S.J.: Video streaming: Concepts, algorithms, and systems. HP Laboratories, report HPL-2002-260 (2002)

[4] Apostolopoulos, J.G., Wee, S.J.: Video Compression Standards. John Wiley & Sons, Inc. (2001)

[5] Armstrong, M., D, F., Hammond, M., Jolly, S., Salmon, R.: High frame-rate television,. BBC White Paper WHP 169, (September 2008)

[6] Aurrecoechea, C., Campbell, A.T., Hauw, L.: A survey of qos architectures. Multimedia Systems 6(3), 138–151 (May 1998)

[7] Awiphan, S., Su, Z., Katto, J.: Tomo: A two-layer mesh/tree structure for live streaming in p2p overlay network. In: CCNC. 7th IEEE Consumer Communications and Networking Conference. pp. 1–5 (2010)

[8] Banerjee, S., Lee, S., Bhattacharjee, B., Srinivasan, A.: Resilient multicast using overlays. IEEE/ACM Transactions on Networking 14, 237–248 (April 2006)

[9] Baumgart, I., Heep, B., Krause, S.: Oversim: A flexible overlay network simulation framework. In: IEEE Global Internet Symposium. pp. 79–84 (2007)

[10] Bhaskaran, V., Konstantinides, K.: Image and Video Compression Standards: Algorithms and Architectures. Kluwer Academic Publishers, Norwell, MA, USA, 2nd edn. (1997)

[11] Biernacki, A.: Simulation of p2p tv system using omnet++. In: EuroView (2010)

[12] Birrer, S., Lu, D., Bustamante, F., Qiao, Y., Dinda, P.: Fatnemo: Building a resilient multi-source multicast fat-tree. In: Chi, C.H., Steen, M., Wills, C. (eds.) Web Content Caching and Distribution, Lecture Notes in Computer Science, vol. 3293, pp. 182–196. Springer Berlin Heidelberg (2004)

[13] Bishop, M., Rao, S., Sripanidkulchai, K.: Considering priority in overlay multicast protocols under heterogeneous environments. In: INFOCOM. The 25th IEEE International Conference on Computer Communications. pp. 1–13 (April 2006)

[14] Brinkmeier, M., Fischer, M., Grau, S., Schäfer, G., Strufe, T.: Methods for improving resilience in communication networks and p2p overlays. PIK. Praxis der Informationsverarbeitung und Kommunikation 32, 64–78 (2008)

[15] Brinkmeier, M., Schafer, G., Strufe, T.: Optimally dos resistant p2p topologies for live multimedia streaming. IEEE Transactions on Parallel and Distributed Systems 20(6), 831–844 (June 2009)

[16] Byun, H., Lee, M.: Hypo: A peer-to-peer based hybrid overlay structure. In: ICACT. 11th International Conference on Advanced Communication Technology. vol. 01, pp. 840–844 (2009)

[17] Calvert, K., Doar, M., Zegura, E.: Modeling internet topology. IEEE Communications Magazine 35(6), 160–163 (June 1997)

[18] Castro, M., Druschel, P., marie Kermarrec, A., Nandi, A., Rowstron, A., Singh, A.: Splitstream: high-bandwidth multicast in cooperative environments. In: Symposium on Operating Systems Principles. pp. 298–313 (2003)

[19] Chan, K.H., Chan, S.H., Begen, A.: Optimizing substream scheduling for peer-to-peer live streaming. In: CCNC. 2010 7th IEEE Consumer Communications and Networking Conference. pp. 1–5 (Jan 2010)

[20] Chu, Y., Rao, S.G., Seshan, S., Zhang, H.: A case for end-system multicast. IEEE Journal on Selected Areas in Communications special issue on Network Support for Multicast Communications 20(8) (October 2002)

[21] Cisco: Cisco vni global ip traffic forecast, 2014–2019 (May 2015), `http://www.cisco.com`

[22] Cohen, B.: Incentives build robustness in bittorrent. In: Workshop on Economics of Peer-to-Peer systems. vol. 6, pp. 68–72 (2003)

[23] Day, K.: Characterization of node disjoint paths in arrangement graphs. Tech. rep. (1991)

[24] Deering, S.E.: Multicast routing in internetworks and extended lans 8(2), 55–64 (May 1988)

[25] Diot, C., Levine, B., Lyles, B., Kassem, H., Balensiefen, D.: Deployment issues for the ip multicast service and architecture. IEEE Network 14(1), 78–88 (January 2000)

[26] Feng, Y., Li, B.: Peer-assisted media streaming: A holistic review. In: Intelligent Multimedia Communication: Techniques and Applications, vol. 280, pp. 317–340. Springer (2010)

[27] Fischer, M., Grau, S., Nguyen, G., Schaefer, G.: Resilient and underlay-aware p2p live-streaming. Computer Networks 59(0), 122–136 (2014)

[28] Flierl, M., Girod, B.: Generalized b pictures and the draft h.264/avc video-compression standard. IEEE Transactions on Circuits and Systems for Video Technology 13(7), 587–597 (July 2003)

[29] Floyd, S., Fall, K.: Promoting the use of end-to-end congestion control in the internet. IEEE/ACM Transactions on Networking 7(4), 458–472 (August 1999)

[30] Grau, S., Fischer, M., Brinkmeier, M., Schaefer, G.: On complexity and approximability of optimal dos attacks on multiple-tree p2p streaming topologies. IEEE Transactions on Dependable and Secure Computing 8, 270–281 (March 2011)

[31] Gu, Y., Zong, N., Zhang, Y., Piccol, F., Duan, S.: Survey of p2p streaming applications (October 2014)

[32] Hei, X., Liu, Y., Ross, K.: Iptv over p2p streaming networks: the mesh-pull approach. IEEE Communications Magazine 46(2), 86–92 (February 2008)

[33] Hei, X., Liu, Y., Ross, K.: Inferring network-wide quality in p2p live streaming systems. IEEE Journal on Selected Areas in Communications 25(9), 1640–1654 (December 2007)

[34] Helder, D.A., Jamin, S.: Banana tree protocol, an end-host multicast protocol. Submitted for publication (2000)

[35] Jannotti, J., Gifford, D.K., Johnson, K.L., Kaashoek, M.F., O'Toole, Jr., J.W.: Overcast: Reliable multicasting with on overlay network. In: Proceedings of the 4th Conference on Symposium on Operating System Design and Implementation

[36] Kiraly, C., Abeni, L., Lo Cigno, R.: Effects of p2p streaming on video quality. In: ICC. IEEE International Conference on Communications. pp. 1–5 (May 2010)

[37] Kobayashi, M., Nakayama, H., Ansari, N., Kato, N.: Robust and efficient stream delivery for application layer multicasting in heterogeneous networks. IEEE Transactions on Multimedia 11(1), 166–176 (January 2009)

[38] Lei, J., Shi, L., Fu, X.: An experimental analysis of joost peer-to-peer vod service. Peer-to-Peer Networking and Applications 3(4), 351–362 (2010)

[39] Li, B., Wang, Z., Liu, J., Zhu, W.: Two decades of internet video streaming: A retrospective view. ACM Transactions on Multimedia Computing, Communications, and Applications 9(1s), 33:1–33:20 (October 2013)

[40] Li, B., Keung, G., Xie, S., Liu, F., Sun, Y., Yin, H.: An empirical study of flash crowd dynamics in a p2p-based live video streaming system. In: GLOBECOM. IEEE Global Telecommunications Conference. pp. 1–5 (December 2008)

[41] Li, B., Xie, S., Qu, Y., Keung, G., Lin, C., Liu, J., Zhang, X.: Inside the new coolstreaming: Principles, measurements and performance implications. In: INFOCOM. The 27th IEEE International Conference on Computer Communications. pp. 1031–1039 (April 2008)

[42] Li, C., Chen, C., Chiu, D.: Buffer map message compression based on relevant window in p2p streaming media system. CoRR abs/1108.6293 (2011)

[43] Liang, J., Nahrstedt, K.: Dagstream: locality aware and failure resilient peer-to-peer streaming. vol. 6071, pp. 60710L+. SPIE (2006)

[44] Liu, J., Rao, S., Li, B., Zhang, H.: Opportunities and challenges of peer-to-peer internet video broadcast. vol. 96, pp. 11–24 (January 2008)

[45] Liu, Y., Guo, Y., Liang, C.: A survey on peer-to-peer video streaming systems. Peer-to-Peer Networking and Applications Vol. 1, No. 1, 18–28 (2008)

[46] Minoli, D.: IP Multicast with Applications to IPTV and Mobile DVB-H. Wiley-IEEE Press (2008)

[47] Moshref, M., Motamedi, R., Rabiee, H., Khansari, M.: Layeredcast - a hybrid peer-to-peer live layered video streaming protocol. In: IST. 5th International Symposium on Telecommunications. pp. 663–668 (December 2010)

[48] Nguyen, G., Fischer, M., Strufe, T.: Ossim: A generic simulation framework for overlay streaming. In: Summer Computer Simulation Conference (July 2013)

[49] Nguyen, G., Fischer, M., Strufe, T.: On the resilience of pull-based p2p streaming systems against dos attacks. In: Felber, P., Garg, V. (eds.) Stabilization, Safety, and Security of Distributed Systems, Lecture Notes in Computer Science, vol. 8756, pp. 33–47. Springer International Publishing (2014)

[50] Nguyen, G., Roos, S., Strufe, T., Fischer, M.: Rbcs: A resilient backbone construction scheme for hybrid peer-to-peer streaming. In: LCN. IEEE International Conference on Local Computer Networks. pp. 1–8 (October 2015)

[51] PPLive: (2015), `http://www.pptv.com`

[52] PPStream: (2015), `http://www.pps.tv`

[53] Riley, G., Henderson, T.: The ns-3 network simulator. In: Wehrle, K., Güneş, M., Gross, J. (eds.) Modeling and Tools for Network Simulation, pp. 15–34. Springer Berlin Heidelberg (2010)

[54] Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: 18th IFIP/ACM International Conference on Distributed Systems Platforms. pp. 329–350. Springer-Verlag (November 2001)

[55] Seyyedi, Y.: Denacast: A p2p video streaming simulator. Available: http://denacast.org/ (2010)

[56] Sopcast: (2015), `http://www.sopcast.org`

[57] Strufe, T., Wildhagen, J., Schäfer, G.: Towards the construction of attack resistant and efficient overlay streaming topologies. Electronic Notes in Theoretical Computer Science 179(0), 111–121 (2007)

[58] Tian, R., Zhang, Q., Xiang, Z., Xiong, Y., Li, X., Zhu, W.: Robust and efficient path diversity in application-layer multicast for video streaming. IEEE Transactions on Circuits and Systems for Video Technology 15(8), 961–972 (August 2005)

[59] UUSee: (2015), `http://www.uusee.com`

[60] Vakali, A., Pallis, G.: Content delivery networks: status and trends. IEEE Internet Computing 7(6), 68–74 (November 2003)

[61] Varga, A.: The inet framework (2012), `http://inet.omnetpp.org`

[62] Varga, A., Hornig, R.: An overview of the omnet++ simulation environment. In: Simutools. Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems. pp. 1–10. ICST (2008)

[63] Veloso, E., Almeida, V., Meira, W., Bestavros, A., Jin, S.: A hierarchical characterization of a live streaming media workload. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment. pp. 117–130. ACM (2002)

[64] Venkataraman, V., Yoshida, K., Francis, P.: Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In: ICNP. Proceedings of the 14th IEEE International Conference on Network Protocols. pp. 2–11 (2006)

[65] Vishnumurthy, V., Francis, P.: On heterogeneous overlay construction and random node selection in unstructured p2p networks. In: INFOCOM. The 25th IEEE International Conference on Computer Communications. pp. 1–12 (April 2006)

[66] Vu, L., Gupta, I., Liang, J., Nahrstedt, K.: Measurement and modeling of a large-scale overlay for multimedia streaming. In: QSHINE. The Fourth International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness. pp. 3:1–3:7. ACM (2007)

[67] Wang, F., Liu, J., Xiong, Y.: Stable peers: Existence, importance, and application in peer-to-peer live video streaming. In: INFOCOM. The 27th IEEE International Conference on Computer Communications. pp. 1364–1372 (April 2008)

[68] Wang, F., Xiong, Y., Liu, J.: mtreebone: A collaborative tree-mesh overlay network for multicast video streaming. IEEE Transactions on Parallel and Distributed Systems 21(3), 379–392 (March 2010)

[69] Weingartner, E., vom Lehn, H., Wehrle, K.: A performance comparison of recent network simulators. In: ICC. IEEE International Conference on Communications. pp. 1–5 (June 2009)

[70] Whitaker, A., Wetherall, D.: Forwarding without loops in icarus. In: IEEE Open Architectures and Network Programming Proceedings. pp. 63–75 (2002)

[71] Wichtlhuber, M., Richerzhagen, B., Ruckert, J., Hausheer, D.: Transit: Supporting transitions in peer-to-peer live video streaming. In: IFIP Networking Conference. pp. 1–9 (June 2014)

[72] Wiegand, T., Sullivan, G., Bjontegaard, G., Luthra, A.: Overview of the h.264/avc video coding standard. IEEE Transactions on Circuits and Systems for Video Technology 13(7), 560–576 (July 2003)

[73] Wildhagen, J., Strufe, T., Schäfer, G.: Netzwerkeffizienz stabiler overlay-streaming-topologien (network efficiency of stable overlay streaming topologies). Information Technology 49(5), 304–311 (2007)

[74] Wilkinson, B.; Allen, M.: Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Prentice-Hall, Upper Saddle River, NJ,, 2nd edn. (2005)

[75] Williamson, B.: Developing IP Multicast Networks. Cisco Press (1999)

[76] Xie, S., Li, B., Keung, G.Y., Zhang, X.: Coolstreaming: Design, theory, and practice. IEEE Transactions on Multimedia 9, 1661–1671 (2007)

[77] Zatoo: (2015), `http://zattoo.com`

[78] Zegura, E.: Gt-itm: Georgia tech internetwork topology models (1996)

[79] Zhang, M.: p2pstrmsim: Peer-to-peer streaming simulator (2009), `http://media.cs.tsinghua.edu.cn/~zhangm/`

[80] Zhang, M., Sun, L., Fang, Y., Yang, S.: igridmedia: The system to provide low delay peer-to-peer live streaming service over internet. Peer-to-Peer Networking and Applications 3(3), 175–185 (2010)

[81] Zhang, M., Zhang, Q., Sun, L., Yang, S.: Understanding the power of pull-based streaming protocol: Can we do better? IEEE Journal on Selected Areas in Communications 25, 1678–1694 (2007)

[82] Zhang, X., Liu, J., Li, B., Yum, Y.S.: Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In: INFOCOM. The 24th IEEE International Conference on Computer Communications. vol. 3, pp. 2102–2111 (March 2005)

[83] Zhou, Y., Chiu, D.M., Lui, J.C.S.: A simple model for chunk-scheduling strategies in p2p streaming. IEEE/ACM Transactions on Networking 19, 42–54 (February 2011)