REDUZINDO O IMPACTO DE PERDAS DE CHUNK EM SISTEMAS P2P DE LIVE STREAMING

JOÃO FERREIRA D´ARAÚJO E OLIVEIRA

REDUZINDO O IMPACTO DE PERDAS DE CHUNK EM SISTEMAS P2P DE LIVE STREAMING

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: Sérgio Vale Aguiar Campos

Belo Horizonte

Dezembro de 2015

JOÃO FERREIRA D´ARAÚJO E OLIVEIRA

LOWERING THE IMPACT OF CHUNK LOSSES IN P2P LIVE STREAMING SYSTEMS

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

ADVISOR: SÉRGIO VALE AGUIAR CAMPOS

Belo Horizonte

December 2015

Oliveira, João Ferreira D´Araújo e

O48L Lowering the Impact of Chunk Losses in P2P Live Streaming Systems / João Ferreira D´Araújo e Oliveira. — Belo Horizonte, 2015
xxxi, 103 f. : il. ; 29cm

Tese (doutorado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação
Orientador: Sérgio Vale Aguiar Campos

1. Computação - Teses. 2. Sistemas de transmissão de dados - Teses. 3. Arquitetura de redes de computador - Teses. 4. Comutação de pacotes (Transmissão de dados) - Teses. I. Título.

CDU 519.6*21(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Lowering the impact of chunk losses in P2P live streaming systems

JOÃO FERREIRA D' ARAÚJO E OLIVEIRA

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

Belo Horizonte, 07 de dezembro de 2015.

Aos meus pais, João e Telma.

Resumo

Sistemas de transmissão de mídia contínua ao vivo estão ficando mais populares a cada dia. Esses sistemas atraem um crescente número de usuarios e, inclusive, alguns importantes canais de TV já disponibilizam alguns conteúdos ao vivo pela Internet. Sistemas par-a-par de transmissão de vídeo em tempo real podem prover uma solução de baixo custo ao problema de transmitir esse conteúdo para um grande número de espectadores. No entanto, estes sistemas não são inteiramente confiáveis e não implementam mecanismos de garantia de qualidade, o que pode levar leva a uma degradação do desempenho do sistema experimentado pelos pares. Essa tese estuda a minimização da perda de pedaços em sistemas P2P de transmissão de vídeo em tempo real, entendendo a razão por trás das perdas, e propondo novas formas de reagir sobre cada uma destas razões, mantendo os custos baixos. Nós desenvolvemos nosso próprio sistema P2P de transmissão de mídia contínua ao vivo e o usamos para caracterizar perda de pedaços. Nós descobrimos uma forma de minimizar a perda de pedaços em cenários com *free riders* ao evitar requisitar pedaços de pares não cooperativos. Nós propomos um mecanismo de suporte para recuperação de pedaços "próximos a serem perdidos", que tem alto potencial para reduzir perdas; ele encaminha requisições emergenciais para fontes fora da vizinhança do par. Nós mapeamos esse mecanismo em um sistema CDN-P2P e desenvolvemos um algoritmo para minimizar dinamicamente o número de fluxos de vídeo sendo plantados na rede P2P ao mesmo tempo em que garante uma disseminação eficiente e evitando requisições emergenciais. Nós avaliamos o algoritmo sob diversas condições nas quais ele chegou a conservar 30% da banda de *upload* previamente usada com requisições emergenciais sem que a qualidade do vídeo fosse afetada.

Palavras-chave: Par-a-Par, Live Streaming, Qualidade de Serviço, Perda de Pedaços.

Abstract

Live streaming systems are becoming increasingly more popular. These systems attract a growing number of users, and some important TV channels already broadcast their live content on the Internet. Peer-to-Peer live streaming systems may provide a low cost solution to the problem of transmitting live content to a large number of viewers. However, these systems are not fully reliable and do not implement quality assurance mechanisms, which leads to a degradation of system performance experienced by peers. This thesis studies chunk loss minimization in P2P live streaming systems, understanding the reason behind losses, and proposing new ways to respond to each reason while keeping low costs. We have developed our own fully capable P2P live streaming system and have used it to characterize chunk loss. We found a way to minimize chunk losses in freeriding scenarios by avoiding to request chunks to uncooperative peers. We proposed a support mechanism to recover "about to be lost" chunks that should significantly reduce losses; it forwards emergency requests to sources outside of a peer neighborhood. We mapped this mechanism to a CDN-P2P system and developed an algorithm to dynamically minimize the number of video streams seeded to the P2P overlay while guaranteeing efficient dissemination and avoiding emergency requests. We have evaluated it under diverse conditions in which it may save up to 30% of upload bandwidth previously used by emergency requests without reducing stream quality.

Keywords: Peer-To-Peer, Live Streaming, Quality of Service, Chunk Loss.

Resumo Estendido

Sistemas de transmissão de mídia contínua ao vivo estão ficando mais populares a cada dia. Esses sistemas atraem um crescente número de usuarios e, inclusive, alguns importantes canais de TV já disponibilizam alguns conteúdos ao vivo pela Internet. Sistemas par-a-par de transmissão de vídeo em tempo real podem prover uma solução de baixo custo ao problema de transmitir esse conteúdo para um grande número de espectadores. À medida em que o potencial para esses sistemas se consolida, provedores como PPLive afirmam ter uma base de usuários de centenas de milhões de telespectadores e permitir que dezenas de milhões de pessoas assistam seus canais simultaneamente.

Transmissão P2P de mídia contínua ao vivo se baseia em dividir a mídia em pedaços e distribuí-los aos participantes da rede P2P. Quando um pedaço não é recebido pelo par dentro do seu prazo de exibição, o tocador de mídia tem duas opções: esperar até que o pedaço chegue, atrasando o par em relação aos outros, ou pular a exibição do pedaço. Para o espectador, ambos os cenários são indesejáveis e indicam má qualidade do sistema. Atualmente a maioria dos sistemas par-a-par comerciais não são inteiramente confiáveis e não implementam mecanismos de garantia de qualidade, o que pode levar leva a uma degradação do desempenho do sistema experimentado pelos pares.

Essa tese estuda a minimização da perda de pedaços em sistemas P2P de transmissão de vídeo em tempo real, entendendo a razão por trás das perdas, e propondo novas formas de reagir sobre cada uma destas razões, mantendo os custos baixos. A ideia é não utilizar os mecanismos padrões do próprio sistema para realizar a recuperação, mas criar novas estratégias para a regeneração do sistema antes mesmo do problema ocorrer, tornando a experiência do usuário perfeita e atraindo empresas de radiodifusão a adotar a tecnologia P2P.

Trabalhos Realizados

Nós desenvolvemos o TVPP, um sistema P2P de transmissão de mídia contínua ao vivo. Seu desenvolvimento foi motivado por restrições ao rodar experimentos em sistemas comerciais, geralmente resultando em dados imprecisos e incompletos. O TVPP pode ser usado para testar uma vasta gama de cenários, alterando parâmetros e algoritmos utilizados pelo sistema. Nós comparamos o TVPP com o SopCast, um sistema comercial bastante conhecido.

Nós utilizamos o TVPP para caracterizar perdas de pedaços. O estudo foi dividido em dois cenários: irrestrito e com restrição de largura de banda. Descobriu-se que mesmo em um cenário irrestrito de largura de banda alguns pares apresentaram perdas significativas. A perda de pacotes no sistema como um todo foi estável ao longo do tempo. Apesar das perdas terem ocorrido, em muitos momentos houve pares que não tiveram nenhuma requisição de pedaços feita a eles. Na maioria das vezes as requisições foram respondidas na primeira tentativa, pelo primeiro par a anunciar que tinha o pedaço. Pedaços que foram perdidos estavam disponíveis em menos candidatos do que pedaços que foram recebidos. Finalmente, perdas de pedaços não são orientadas por uma distribuição ruim de um pedaço específico na rede, nem por grandes sequências de perdas consecutivas. Estes resultados sugerem duas soluções: uma melhoria no método de organização da rede sobreposta e, em última instância, um mecanismos de requisição emergencial para recuperar os pedaços que estão próximos de serem perdidos fora da vizinhança normal do par.

As perdas ficam piores, todavia sustentáveis ainda, uma vez que nós começamos a restringir a largura de banda através da introdução de *free riders*. Observando *free riders* conscientes nós notamos que a perda de pedaços foi qualitativamente similar para um percentual de 50% de *free riders* compondo a rede, enquanto que a latência média foi acrescida de 1 segundo somente. Entretanto, a mesma fração de *free riders* não conscientes causa uma degradação significativamente maior, em grande parte por conta de um crescimento no número de tentativas de realizar uma requisição. Tais tentativas consecutivas desperdiçam banda e aumentam a latência média por pedaço. Para evitar isso, um algoritmo foi desenvolvido para complementar o escalonador de requisições, identificando pares não cooperativos dentro de poucas interações. Obtivemos assim resultados para latência e perda de pedaços similares para as mesmas quantidades de *free riders* conscientes e não conscientes.

Finalmente, nós trabalhamos com o Hive, uma solução comercial CDN-P2P para transmissão de mídia contínua ao vivo. No Hive, um par que está prestes a perder o prazo de um pedaço solicita-o emergencialmente para o servidor da CDN. Observamos

que tais requisições garantem qualidade de serviço, mas entregam pedaços perto do seu prazo de exibição, o que deixa pouco tempo para que eles sejam retransmitidos na rede P2P. A solução evita a perda de pedaço, mas é ineficiente em resolver um eventual problema da disponibilidade do dado na rede P2P. Nós desenvolvemos um mecanismo que dinamicamente adéqua a quantidade de dados alimentada à rede P2P pela CDN de forma a garantir uma disseminação de pedaços eficaz e reduzir o numero de requisições emergenciais. Nós avaliamos a solução sob diversas condições e encontramos potencial para reduzir o consumo de banda da CDN em 30% mantendo a qualidade da mídia.

Em resumo, nós descobrimos que a maioria das perdas acontece por razões muito específicas para serem pontuadas e tratadas. Todavia, uma importante razão identificada foi a requisição de pedaços a pares não cooperativos. Nós propomos um método simples para identificá-los e evitar tais requisições. Para as demais razões, nós propomos um serviço onde pedaços que estão prestes a serem perdidos ganham uma última chance de serem recuperados fora da vizinhança do par. Essa solução tem potencial para garantir uma entrega de pedaços próxima a 100% dependendo da forma que for implementada. Apesar disso, tal método pode incorrer em custos de infraestrutura adicionais. Embora isso pareça inevitável, propusemos um método para minimizar esses custos. Finalmente, com este trabalho obtivemos uma melhor compreensão de por que os pedaços são perdidos na transmissão P2P de mídia contínua ao vivo, e como minimizar essas perdas. Um sistema de transmissão usando estas técnicas será significativamente mais eficiente para transmitir fluxos sem aumentar custos, tanto quanto possível.

Extended Abstract

Live streaming systems are becoming more popular each day. These systems attract a growing number of users, and some important TV channels already broadcast their live content on the Internet. P2P live streaming systems may provide a low cost solution to the problem of transmitting live content to a large number of viewers. As the potential for P2P live streaming consolidates itself, providers, such as PPlive, claim to have a user base of hundreds of millions of viewers and allow dozens of millions to watch their channels simultaneously.

P2P live streaming relies on splitting the stream into chunks and spreading them to the overlay participants. Whenever a chunk misses its playback deadline either the player will have to wait until it arrives, delaying the peer in relation to others, or it will skip it. For the viewer, both scenarios are undesirable, annoying and, indicatives of the system's bad quality. Yet today most commercial P2P systems are unreliable, as quality assurance mechanisms are still in a developing stage, or even completely ignored, and several scenarios can cause degradation of system performance experienced by peers.

This thesis studies chunk loss minimization in P2P live streaming systems, understanding the reason behind losses, and proposing new ways to respond to each reason while keeping low costs. The principle is not to use common system mechanisms already present upon recovery, but rather to create new strategies so systems could preemptively identify problems and regenerate, making the user experience flawless, and attracting broadcasters to embrace P2P technology.

Work Accomplished

We have developed TVPP, a P2P live streaming system. Its development has been motivated by impairments to run experiments with commercial systems and tools, often resulting in incomplete or imprecise data. It can be used to test scenarios under a wide set of parameters and algorithms. We have compared TVPP with SopCast, a well-known commercial system.

We have used TVPP to characterize chunk losses. The study has been performed in two scenarios: resourceful and bandwidth-constrained. It has been found that for a resourceful scenario few peers exhibit significant losses. System-wide average chunk loss has been stable in time. While losses do occur, there are many moments where peers had no requests made to them. Mostly, requests receive replies in the first attempt, by the first peer to announce that it has the chunk. Chunks that have been lost have had less candidates to request from than chunks that have been received. Finally, chunk losses are not driven by a bad spread of a specific chunk or by big loss sequences. These results suggest two solutions: an improved overlay organization method and, ultimately, an emergency request mechanism to request "about to be lost" chunks outside of a peer's partner set.

Losses get worse, yet are still sustainable, once we start to reduce bandwidth through the introduction of free riders. Observing conscious free riders we have noted that chunk losses have been qualitatively similar with less than 50% free riders while the average latency had increased by 1s at most. However, the same fraction of oblivious free riders causes significantly worse degradation, mostly triggered by an increase in retries. Consecutive retries waste bandwidth and increase average chunk latency. To avoid this, an algorithm has been developed to complement the request scheduler which identifies uncooperative peers within few interactions. Using it we have obtained similar chunk loss and latency results for the same amount of conscious and oblivious free riders.

Finally, we have worked with Hive, a CDN-P2P commercial live streaming solution. On Hive, a peer that is about to miss a chunk playback deadline issues an emergency request to the CDN. We have found that emergency requests guarantee quality of service, but delivering a chunk too close to its playback deadline leaves little to no time for redistribution in the P2P overlay. Such requests avoid chunk losses but are inefficient to solve P2P data availability problems. We have developed a mechanism that dynamically minimizes the number of video streams seeded to the P2P overlay while guaranteeing efficient dissemination and avoiding emergency requests. We have evaluated it under diverse conditions in which it may save up to 30% of upload bandwidth previously used by emergency requests without reducing stream quality.

In summary we have found out that most losses happen for reasons too specific to be pointed out and have it treated. One important reason that has been identified, however, was requesting chunks to uncooperative peers. We have proposed a simple method to track this behavior and avoid it. For other reasons, we have also proposed a service where chunks that would be lost could have a last chance to be retrieved outside of the peer's neighborhood. This solution has the potential to guarantee close to 100% delivery depending on how it is implemented. Nevertheless, such a method to neutralize losses might incur in additional infrastructure costs. While this may be unavoidable, we have proposed a method to minimize these costs. Finally, in this work we have obtained a better understanding of why chunks are lost in the transmission of live P2P streams, and how to minimize these losses. A transmission system using these techniques will be significantly more efficient to transmit streams without increasing costs as much as possible.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In the last decade the Internet has grown significantly through the expansion of audiovisual content sharing. We have seen the Internet achieve its success evolving in size and applications through Web 1.0 - with text and static pictures -, instant messaging, file sharing and the mp3 musical revolution, real time voice transmission (as if we were using a phone), Web 2.0 and video on demand. Services such as Napster and YouTube were iconic examples that boosted Internet usage and popularity. Nowadays, the Internet flagship is video transmission; its traffic is still growing as video services offer new resolutions, such as 4K UHD [Amazon, 2014]. Globally, consumer internet video traffic will be 80 percent of all consumer Internet traffic in 2019, up from 64 percent in 2014 [Cisco, 2015]. Yet, the next imaginable evolution, video transmission in real time (or live streaming), has not yet emerged as a great success. There is no iconic live streaming application that stands out worldwide.

Audiovisual content transmission in real time over the Internet has yet to become popular. Television broadcast is the technology most similar to live streaming, and what can be extracted from the television scenario is that **1)** ordinary viewers do not have a habit of sharing real time videos or producing their own shows, and **2)** broadcasters are not interested on abandoning their current reliable and scalable Ultra High Frequency (UHF) transmission structure. There is a business opportunity, however, given that the Internet faces no geographical boundaries. During the FIFA 2014 World Cup in Brazil, Globo TV network live streamed all games achieving impressive numbers. During the stream of a single game, Globo achieved five hundred thousand simultaneous viewers. The accumulated duration of all streams during the event was approximately 1600 years in total [Moreira, 2015].

Moreover, individual producers of specific content have a way to reach interested viewers without having to fit to a predetermined program schedule like on TV. Services such as TwitchTV [Twitch TV, 2015] and Hitbox [Hitbox, 2015] specialize in broadcasting live content related exclusively to video games. Some championships reach hundreds of thousands of simultaneous viewers. Another highlight is Periscope, a live streaming application which was released in late March, 2015. The developing startup was acquired for US$100 million by Twitter even before reaching the public. One million users joined Periscope in its first 10 days on the iPhone app store. Yet, the number of viewers watching each stream is still relatively low; CNN Royal Correspondent Max Foster, who covered the birth of Duchess of Cambridge's second child live through the application, reported a maximum of one and a half thousand viewers during his streams [CNN, 2015]

The traditional model used for streaming content on the Internet is the client-server model. A server opens one connection and performs transmission for each user interested in the content. With live streaming, the server continuously transmits to each user the video at the same rate it is produced. In scenarios where there are many users interested in a dense video, such as high quality videos, hte client-server model stands out negatively by its low scalability. Content servers may eventually become bottlenecks that affect response time. This model is reliable and with some planning can be structured to comply with quality of service (QoS) requirements. One can replicate and distribute content across multiple well-located servers over the Internet and ensure increased availability and performance. That solution is called content delivery networks [Dilley et al., 2002], or CDNs, however, its structure demands costly investments.

Alternatively, if each user interested in the content would initially consume it – as a client –, store it and then supply it – as a server –, it would be possible to distribute the total load among these "partners". From that idea arises the term that defines this type of transmission: peer-to-peer (P2P), which became popular from 1999 with systems such as Napster [Shirky, 2001], Gnutella [Ripeanu, 2001], Emule [Kulbak and Bickson, 2005], BitTorrent [Cohen, 2003] and Skype [Guha et al., 2006]. In particular, this work studies P2P live streaming systems.

P2P live streaming has gained some notoriety over the last decade for its rise and numbers. There are several systems that hold millions of users connected daily [Hei et al., 2007a; Wu et al., 2007; Huang et al., 2008; Sentinelli et al., 2007]. This indicates that users are shifting their desire for live broadcasted content to the Internet, a natural technological exchange. Some see more aggregated value to this service such as greater choice of content, potential for interactivity and mobility at lower prices [CNET, 2010] and others belong to a new class of users who sees the possibility of creating and producing content by themselves with low-cost distribution [Sentinelli et al., 2007; Silva, 2009].

P2P live streaming fits these demands. System scalability as the number of users grows, load distribution and transfer rates achievable through P2P technology are fundamental to ease the burden and infrastructure costs related to dedicated servers. Yet, there are open problems in the P2P live streaming scenario. P2P brings deficiencies to live streaming that, if not treated properly, may hinder the service popularity. Some of the main challenges are related to management and performance, turnover of partners (churn) and lack of guarantees on quality of service. Refinement of these latent issues is essential to improve P2P live streaming. Thus, the client-server model is still preferred because **1)** there is a lack of maturity to P2P live streaming systems, and **2)** client-server provides more guarantees as servers can be pressured to offer an acceptable quality if the investment is high enough.

Commercial success of any live streaming service depends on the quality provided to end users. Nowadays, P2P live streaming viewers watch streams with acceptable quality. However, this is not true at all times, nor for all users. Most commercial systems are unreliable, as quality assurance mechanisms are still in a developing stage, or even completely ignored, and several scenarios can cause degradation of system performance experienced by users [Moltchanov, 2011].

Yet, quality is a broad term. There is a special class of videos, such as soccer matches, where stream latency is very important. Those videos cannot afford to be delayed by dozens of seconds or more. But, for most contents, continuity and perfect stream decoding is more important, and that is achieved by controlling losses of chunks of streamed content. Whenever a chunk misses its playback deadline either the player will have to wait until it arrives – delaying the peer in relation to others or eventually freezing exhibition – or it will skip the chunk – exhibiting badly decoded frames depending on system design. For the viewer, watching a show that keeps stuttering or with broken frames is undesirable, annoying and, indicates system's bad quality.

Chunk loss may depend on a number of system design peculiarities including network topology, scheduling, incentive mechanism, etc [Moltchanov, 2011]. Such systems are also subject to scenarios that increase chunk loss, such as peers that not share (free riders), local neighborhood bandwidth constraints or the unavailability of a chunk in the neighborhood within its deadline. Moreover, chunk loss scales with the relation between network bandwidth distribution and video resolution. Although several studies try to improve or solve these problems by working on policies for chunk scheduler [Zhao et al., 2009; Birke et al., 2011] or peer selection mechanisms [Traverso et al., 2014; Simoni et al., 2014; Roverso et al., 2013], few studies focus on ensuring chunk delivery, specially while facing particularly bad scenarios. Most current systems are not well prepared to act in face of chunk loss.

1.1 Problem

Acting upon chunk losses is essential. A single loss may disrupt the stream decoding for seconds. Losses may affect in different ways the momentary displayed error depending on decoder's error concealment capabilities. High quality motion codecs such as H.264 [ITU-T, 2015] are designed to take advantage of redundancies between video frames. If a loss affects a key frame (I-frame), the effect will propagate through every frame until the next key frame arrive. Besides, loss of consecutive chunks may generate a signal interruption for several seconds. Stream disruption and chunk loss are considered direct and measurable impacts. Indirectly, these failures lead to **1)** decreased confidence of users in the system, **2)** the eventual abandonment of these users and **3)** a system collapse.

In this work we study the problem of chunk loss and methods to avoid/eliminate it. We give an extra focus at bandwidth constrained scenarios where chunk loss problems become massive. Finally, although some methods are costless, final methods to neutralize losses incur in infrastructure cost increase and, since P2P architectures are attractive for their low costs, it is also important to track and minimize the tradeoff of ensuring chunk delivery.

1.2 Objectives and Contributions

This thesis main goal is to study chunk losses in P2P networks used for live streaming and propose new methods to avoid/eliminate them. We employ harsh bandwidth conditions not only to create these methods but also to improve them. Below we will briefly describe our contributions and further detail them in the following paragraphs.

- Development of a P2P live streaming system with academic purposes and design similar to commercial systems;

- Characterization of chunk loss on a generic P2P mesh-pull live streaming network, extracting failure-related information – which peers, for what reasons, how much impact – with and without bandwidth constraints;

- Analysis and development of a non-punitive algorithm to reduce the impact of free riding – as a bandwidth constraint;

- Proposal of a solution to assist and complement traditional P2P transmission mechanisms aimed at ensuring chunk delivery to all peers obeying time restriction requirements;

- Analysis and development of an algorithm to reduce costs associated with ensuring chunk delivery;

In practice, no live streaming system has 100% delivery guarantees, especially P2P ones. There are a number of reasons behind chunk losses in P2P live streaming systems. Chunk availability bottlenecks may occur, external factors may reduce peer's upload capacity, churn makes everything less reliable, among other problems. Ultimately, the underlying Internet architecture is best-effort. Underlay connection problems may arise or network cables can accidently be disconnected which make the delivery infeasible. Nevertheless, there are chunks which could have been delivered but are lost because of system design inefficiency. There are dozens of studies that deal with chunk scheduling mechanisms, neighborhood management, or server placement aimed at optimize and solve these problems [Birke et al., 2011; Traverso et al., 2014; Moltchanov, 2011; Simoni et al., 2014; Roverso et al., 2013; Zhao et al., 2009]. Even these optimizations can fail however. These algorithmical optimizations tend to improve overall quality but do not treat specific client problems. In P2P systems each peer strives to perform his part, but that does not necessarily generate a collective guarantee. How can a system step further and increase the quality assurance for individual peers beyond those solutions? We address this feasible delivery aiming to eliminate chunk loss.

We start this work by describing TVPP [Oliveira et al., 2013b] (Chapter 2). TVPP is an academic P2P live streaming system designed to provide a similar service to popular proprietary commercial systems but with a few advantages for researchers. TVPP's local development has been motivated by the need for a P2P live streaming system in which one could run many sorts of experiments, with freedom to modify the source code, and easier access to experimental result logs. The tool mimics the behavior of commercial systems such as SopCast. Through TVPP description we introduce the major mechanics and theory behind peer-to-peer live streaming up to their details. Moreover, Section 2.4 presents an emergency requesting mechanism proposal for "pure" P2P live streaming systems which has the potential to drastically reduce chunk losses. Peers monitor their own missing chunks and contact the Emergency Request Service once they understand that a specific chunk has only one more opportunity to be requested before it misses its playback deadline. The Emergency Request Service forwards the received requests to handlers which are peers selected from within the overlay. Handlers respond to original emergency request sources with the missing chunk.

After that, we characterize chunk loss on TVPP under different bandwidth constraints (Chapter 3). In particular, it is important to understand the characteristics of lost chunks and peers that lost it. We have identified a set of reasons behind chunk losses, the relative likelihood for these to happen and what solutions can be applied. We start from a generic and resourceful environment, and we add constraints modifying the parameters as follow-up questions arise. We decrease the amount of bandwidth available through the rise of free riders – peers that refuse to upload data. It can be observed that system-wide chunk loss highly depends on network conditions and system's designs. Finally, we discuss solutions to avoid chunk losses and propose Simple Unanswered Request Eliminator – SURE (Section 3.4.3). It is a modification to the chunk scheduling in which a simple and local evaluation identifies unresponsive neighbors and avoids sending requests to them.

At last, we study an alternative to reduce chunk losses that uses CDN architecture. We evaluate chunk loss elimination at Hive Streaming [Hive, 2015] (Chapter 4). Hive Streaming is a commercial solution for media distribution based on SmoothCache [Roverso et al., 2015], a CDN peer-assisted live streaming system. Started in 2007 as a spin-off from the Swedish Institute for Computer Science and the Royal Institute of Technology in Stockholm, its company maintains a strong focus on research and development. We were given access to SmoothCache code and have studied chunk loss in several scenarios. The benefit of experimenting with this tool is that SmoothCache is a commercial application with several implemented optimizations, including a concept that ensures chunk delivery – typical of CDN peer-assisted systems. The concept states that all data is acquirable from CDN; those who are interested in the content form a support P2P overlay, which should be queried first for data. Thus, SmoothCache peers know CDN servers addresses and, ultimately, will request chunks to the CDN if a chunk cannot be found or a request fail to be responded by its partners. We mapped this behavior as an equivalent of an emergency request. Therefore, we stop evaluating chunk loss as an overlay distribution efficiency metric to analyze how much data has been received through the source, emergency requests, or peers.

We have analyzed emergency requesting tradeoff, i.e. costs or amount of effort expended to acquire the missing chunks. Although emergency requests allow the retrieval of nearly missed chunks, they deliver a chunk close to its deadline, which leaves no time for dissemination through the P2P overlay. This may create a negative feedback loop, where emergency requests compromise P2P dissemination, lead to more emergency requests, and further compromise P2P dissemination. We have developed the Adaptive Emergency Request Optimization – AERO – (Section 4.6), a mechanism that dynamically optimize the number of video streams that servers should seed to P2P overlay – adapting servers' out-degree – while guaranteeing efficient dissemination and avoiding emergency requests. This is especially important on SmoothCache because CDN servers respond as both stream source and emergency request handler, and AERO manages to reduce servers' load by seeding a bit more to reduce the large number of emergency requests. We evaluate AERO under diverse overlay conditions and show that it maintains or reduces servers upload traffic. Although developed using SmoothCache, which is a CDN peer-assisted system, AERO can be applied to "pure" P2P overlays reducing traffic on emergency requests handlers by adjusting handlers' or source's out-degree.

In summary we have found out that most losses happen for reasons too specific to be pointed out and have it treated. One important reason that has been identified, however, was requesting chunks to uncooperative peers. We have proposed a simple method to track this behavior and avoid it. For other reasons, we have proposed a service where chunks that would be lost could have a last chance to be retrieved outside of the peer's neighborhood. This solution has potential to guarantee close to 100% delivery depending on how it is implemented. Nevertheless, such method to neutralize losses might incur in additional infrastructure costs as an external entity will have to handle that chunk delivery. While this may be unavoidable, we have proposed a method to minimize these costs. Finally, in this work we have obtained a better understanding of why chunks are lost in the transmission of live P2P streams, and how to minimize these losses. A transmission system using these techniques will be significantly more efficient to transmit streams without increasing costs as much as possible.

The remainder of this thesis is organized as follows. Chapter 2 explains peer to peer live streaming systems through TVPP. Chapter 3 address the problem of chunk loss and the characterization performed. Chapter 4 discusses reduction of costs caused by chunk loss recovery through emergency requesting. Chapter 5 summarizes the results and describe possible future works.

Chapter 2

Peer-To-Peer Live Streaming: The TVPP Design

In this chapter we present TVPP [Oliveira et al., 2013b], an academic P2P live streaming system designed to be similar to popular proprietary commercial systems, such as SopCast, but with a few advantages for researchers. Through the description of TVPP we introduce the major mechanics and theory behind Peer-To-Peer Live Streaming up to their details. We also provide an experimental comparison between TVPP and SopCast. TVPP has been a key tool for the development of this thesis as an experimental platform.

2.1 Introduction

TVPP is a research-oriented P2P live streaming system. We have built TVPP from the need to simplify, control and get more detailed data from experiments. It has features that are useful for researchers, such as:

- easy data acquisition: no need for additional network traffic analyzers
- configurability: several key parameters to experiment with
- modularity: simply plug in and test new algorithms
- collection of arbitrary data: one can change code to output what he needs

Most popular live systems, such as TvAnts[1], UUSee[2], SopCast[3], PPLive[4] and PPStream[5] are commercial applications, with no publicly available source code, which

[1]tvants.en.softronic.com; [2]www.uusee.com; [3]www.sopcast.org; [4]www.synacat.com; [5]www.ppstream.com

makes it hard for researchers to gather useful data or log files. Most studies targeting these systems deal with a black box and must rely on educated guesses with regard to system architecture, protocols and internals. Crawling the network, analyzing traffic, recreating partial network graphs are examples of methods frequently used to infer system structure and behavior [Vieira et al., 2009; Horvath et al., 2008; Silverston et al., 2009; Ali et al., 2006; Tang et al., 2009]. Moreover, in such a scenario where systems do not facilitate third party testing or reverse engineering, control of key protocol parameters, such as partnership or bandwidth limits, media buffer size, chunk scheduling and partner maintenance strategies, which would allow deeper experimentation and analysis, are absent.

The TVPP design uses similar mechanisms to those of the most popular platforms [Sentinelli et al., 2007]. It is a mesh-pull P2P live streaming system. In order to better understand this kind of systems some key mechanisms are particularly interesting to explore in further detail, such as overlay maintenance, chunk scheduling, emergency requesting, and logging (Sections 2.2 to 2.5). Another important feature is that TVPP has been designed to be expandable, as described in Section 2.6, since it aims to experiment with a wide range of questions, and therefore must be able to include additional algorithms, mechanisms, monitoring metrics or experimental parameters. Section 2.7 explores a full list of configurable parameters present on TVPP. Section 2.8 describes an experimental comparison about traffic and network related metrics between TVPP and SopCast.

2.2 Overlay Construction and Maintenance

In mesh-pull applications, peers are organized in a mesh-like network, without hierarchical roles. Peers have partners from which they request data or send data to. This design flexibility has the advantage of being scalable and fault resilient [Fodor and Dan, 2007; Hei et al., 2008] while compared to tree-based approaches. Besides topology concerns, peers need to know which other peers exist in the overlay, a feature known as peer sampling. We describe tree-based, mesh-based and hybrid topologies, peer sampling methods, and then the TVPP design in details.

2.2.1 Tree-based topology

In tree-based topologies [Deshpande et al., 2002; Tran et al., 2004; Castro et al., 2003; Kostić et al., 2003], peers form an hierarchical structure similar to a tree or a multi-tree graph. Content is transmitted from the root – data source – to its children, and so on until it reaches the leafs.

Benefits are that, since data is been pushed down the tree during chunk exchange, traffic flows are predictable and delay is proportional to the number of transversed hops from the source. Limitations are that the quality delivered to each branch is limited by the upload capacity of its individual root, the distribution potential of many leaves might be wasted [Picconi and Massoulié, 2008; Magharei and Rejaie, 2006], and attacks or even churn can disrupt the flow availability, partition the overlay, and overload the network with overlay repair requests. Some examples of tree-based system are Overcast [Jannotti et al., 2000], Climber [Park et al., 2008], ZigZag [Tran et al., 2003], NICE [Banerjee et al., 2002], SplitStream [Castro et al., 2003] and Orchard [Mol et al., 2007].

2.2.2 Mesh-based topology

The mesh-based topology tries to overcome the tree topology constraints [Hefeeda et al., 2003; Zhang et al., 2005; Magharei and Rejaie, 2006]. It incorporates swarm techniques for content distribution, inspired by mechanisms such as BitTorrent [Cohen, 2003], allowing most peers to actively contribute with their upload bandwidth. Mesh network's basic principle is that participants form an overlay through random connections generating non-deterministic topologies. Lack of hierarchy and of tight structure allows a peer to receive chunks from several sources, which makes media distribution unpredictable.

Benefits are that overlay construction and maintenance are less complex and expensive, and a greater connectivity reduces the likelihood of performance and bandwidth bottlenecks while increases churn resilience [Moltchanov, 2011]. As for drawbacks, control traffic generated is usually significant given that peers need to frequently exchange status information for deciding which are the best peers to download chunks from, and each data chunk is treated as a separate delivery unit and it might be received through a different route, i.e., paths and delivery times are not predictable and are highly variable [Picconi and Massoulié, 2008]. Yet, Magharei et al. [2007] show that a mesh-based structure has a superior performance then a structure based on multiple trees. Some examples of mesh-based system are SopCast, DONet/Coolstreaming [Zhang et al., 2005], Chainsaw [Pai et al., 2005], BiToS [Vlavianos et al., 2006] and PULSE [Pianese et al., 2007].

2.2.3 Hybrid topology

Tree-based and mesh-based approaches can be combined to boost benefits from each other, and to better utilize bandwidth. Hybrid topologies construct trees with peers logically closer to the source, and meshes connecting these with the rest of the network. They get tree's predictability at the early hops while keeping mesh's churn resilience for the rest of the overlay.

CliqueStream [Asaduzzaman et al., 2008] creates clusters of peers using delay and locality metrics. From each cluster, one or more peers are elected to form a tree connecting the clusters to each other and to the source.

mTreebone [Wang et al., 2010] elects a subset of stable peers to form its tree, and all peers request chunks from the tree. Upon failure to receive data from the tree, peers use auxiliary mesh connections. The limitation of this approach is that few stable peers might get congested while others do not contribute with their upload bandwidth.

2.2.4 Peer Sampling

Peer sampling is the act of supplying peers with a restricted sample of overlay members which they may connect to, and it can be done in a centralized or distributed fashion. The centralized method relies on a service which can be contacted for peers samples while the distributed method relies on gossiping [Voulgaris et al., 2005; Dowling and Payberah, 2012; Roverso et al., 2013] which draw peers samples from within peer-to-peer overlay. Heuristics can be used while selecting a sample to improve its relative quality, e.g., sampling peers geographically closer to the requester.

The centralized method mechanic is straightforward, peers request samples directly to a centralized service and receive a subset of all overlay members. Benefits include a view over all members at all times, view's freshness, and immediate convergence of sample selection heuristics. However, as any centralized method, it is a single point of failure and it could become a bottleneck.

The distributed approach to peer sampling is gossiping. Peers keep a fixed-size restricted view of the overlay, periodically select a target node from that view, and exchange a number of items from the view with the target [Jelasity et al., 2007]. The quality of such protocols is associated with their heuristics which are applied for node selection, nodes replacement at target's view, and if peers just push views or if they trade them (enabling partial view swapping). Benefits include solving centralized service issues but at the expense of freshness and of convergence time. Moreover, bad implementations might lead to overlay partitioning. Some examples are Cyclon [Voulgaris et al., 2005], Croupier [Dowling and Payberah, 2012], and WPSS [Roverso et al., 2013].

2.2.5 TVPP implementation

Stream distribution starts from a *channel* creation. The channel concept is similar to television channels. The channel is the system entry point for users. When a peer decides to watch a stream, it searches for other peers that are watching the same content in order to associate itself with. This entry point is managed by a special node called *bootstrap*.

In TVPP, as in SopCast, mesh construction and maintenance relies on a centralized bootstrap server. For each channel c, the bootstrap stores a list of connected peers, $\mathcal{P}_c$, the source peer address, and the last generated chunk ID. In order to keep $\mathcal{P}_c$ up to date, peers send a *ping* packet to the bootstrap periodically. A channel *source*, or server peer, also sends ping messages, which update the last generated chunk ID value for that channel. This value is used in a further moment to initialize or reset peers. Peers are removed from their respective $\mathcal{P}_c$ if they fail to report their existence to the bootstrap for a configurable period, typically a few seconds.

A channel's source peer (one that splits a stream into chunks and starts their distribution) announces to the bootstrap its intent to create a channel. Other peers join the *overlay* sending to the bootstrap a peer request message asking for peers that are watching an existing channel. After receiving a peer request message, the bootstrap selects a subset of peers from $\mathcal{P}_c$ and sends it back to the requester. This message also contains the last generated chunk ID so that new peers know from which chunk to start asking.

Upon receiving the requested subset from the bootstrap, peers store them in a *candidate peer list*. To keep the candidates list freshness, peers will send new peer request messages to the bootstrap at regular intervals.

Until a peer reaches its partnership size limit, they will periodically select candidates to try to establish a partnership with, turning that candidate into a *partner*. TVPP considers directed partnership links. Each peer p has a set of *input partners*, denoted $\mathcal{I}_p$, and *output partners*, denoted $\mathcal{O}_p$. It receives data from the input partners and sends data to the output partners. Each peer tries to fill its own input partner set; consequently, its output partner set is filled by other peers doing the same. Table 2.1 summarizes the notation.

Partnership links are maintained through pings between partners which signalize that each end is still alive. Further details about extra load on pings are discussed

Table 2.1. Definitions and notation.

VAR.	DEFINITION
$\mathcal{P}_c$	Set of peers connected to channel c.
$\mathcal{I}_p$	Set of input partners of peer p.
$\mathcal{O}_p$	Set of output partners of peer p.
R	Video streaming rate.

in the next section for its importance. Partners may face connection issues or leave the network. In these cases they can be turned back into candidates or be dropped. Another feature is that, periodically, partnerships can be selected to be undone.

Selection strategies are an important part of a system [Traverso et al., 2014]. Peer selection strategies are configurable and at any time new strategies can be implemented. They are used to select candidates to become partners or to select partnerships to be undone. A few ways to do peer selection are physical distance, logical distance (round-trip delay time), upload capacity or randomly.

2.3 Chunk Scheduling

The chunk scheduling mechanism of TVPP is inspired by the Coolstreaming data-driven model Zhang et al. [2005]. According to this moel, all peers only request data to partners that actually have the data. This is possible by making peers broadcast periodically to their output partners which chunks they have within the ping messages. This design generates more control overhead, but it prevents request/transmission redundancy.

The broadcast ping contains a *buffer map* which has an integer indicating the newest chunk of media present on the peers's own buffer and a bit map where each position determines the presence or not of a previous chunk. The nth position of the map represents the id of the newest chunk minus n. Thus buffer maps cover a dynamic range of *chunk IDs*.

Knowing which partners have which chunks is only part of the solution. There are three common approaches for chunk exchange: pull, push and hybrid. In the pull approach, data is sent as a response to explicit requests done by a destination peer to a source peer. In the push approach, peers forward data based on their partners needs without the explicit requests. The push approach has less control overhead, but has a great potential for redundant traffic since more than one source can push the same content to a given destination. Typically, tree-based topologies use the push approach while the mesh-based topologies use the pull approach. It is unclear which technique is the best, and commercial systems use them both. Moreover, these can be mixed into a hybrid approach in which sometimes peers push and other times they pull chunks.

As previously mentioned, TVPP is a mesh-pull system. The channel source will naturally be the first peer to have any chunk available. It splits the stream into chunks and associate a chunk ID to each of them. The source announces new chunks to its partners through buffer map messages.

Buffer map messages spread data availability over the entire network. Once any peer announces the presence of a chunk, its partners might try to request it, if they do not have it. Every few milliseconds each peer compares its own buffer map with its partners's maps looking for the newest chunk to request. The peer then creates a request to that chunk, selects a partner to serve this request using a configurable selection strategy, and adds the request to the request list signaling its intent to ask that partner for that particular chunk. After receiving a chunk, the respective request is removed from the request list, the chunk is stored, and the buffer map sets it as present.

There are many approaches to choose which chunk to request first and whom to ask. TVPP request scheduling rule follows the earliest deadline first (EDF) rule, in which chunks closer to meet the deadline are requested first. Other options are requesting the rarest chunk first considering partners buffer maps, or the chunk that more peers need. Jian [2009] discusses benefits of each and how they fit each streaming scenario. As for whom to ask the chunk, the same peer selection strategies described before can be used over the reduced set of peers that have the chunk.

Requests have a configurable expiration timer with the default value set as half second. After a request timer expires a *retry* – a new selection for a serving peer – is held. The number of retries for each request is also configurable. If all tries fail, the request is removed from the request list.

2.4 Emergency Requesting

In this section we discuss the Emergency Request Service, an envisioned design that allows the retrieval of nearly missed chunks. Yet, this design has not been implemented in TVPP and it figures as a proposal. Whenever the peer identifies that it is about to miss a media chunk playback deadline, it issues an emergency request to a reliable source. The design is inspired in fundamental premisses of CDN peer-assisted systems, which are mainly client-server architectures with a P2P support [Roverso et al., 2012, 2015; Zhao et al., 2013; Yin et al., 2009; Mansy and Ammar, 2011; Lu et al., 2012]. In

CDN peer-assisted systems all data is acquirable from CDN servers, but a support P2P overlay exists between those who are interested in the data which should be queried first for it. Once a chunk cannot be found on the overlay or a request fails to be responded by other peers, the request is made to the CDN servers. CDNs are well suited for such a solution as they are built to handle a variable load and can guarantee quality of service up to the contracted capacity. We propose a solution for "pure" P2P live streaming systems which has the potential to drastically reduce chunk losses using resources available over the P2P overlay.

Received chunks are fed to a player for exhibition. The player maintains an exhibition buffer to store these chunks. A gap at the player buffer may occur once that a requested chunk is not received before its due time to be played. There are several reasons that might lead to that, such as network failures, peers failures, churn, bandwidth bottlenecks or data unavailability over the partnerships. Timing constraints of live transmission make this a particularly serious problem. A peer might try to request several times to the P2P network, however, in a given moment it will have only enough time to request once more. At this point, requesting to a more reliable source or just outside the established partnerships might help.

We introduce the design for the Emergency Request Service describing its possible implementation on TVPP. The service address is published to peers through the peer list messages provided by the bootstrap. A new kind of request message is necessary, the emergency request message. Its function is to behave as a normal request that has to be responded not to its source but to a secondary address indicated over the request header. A peer that is about to miss a chunk playback deadline issues an emergency request to the service with its own address added to the message header.

As the emergency request service receives emergency requests, it must forward them to emergency handlers – peers that are capable of responding. An emergency handler that receives a forwarded emergency request checks its buffer for the chunk and sends it directly to the emergency request original message source. The response with a chunk is treated as any other normal response.

Many implementations can be done to define which peers will be emergency handlers, a few are discussed below.

- The channel source might be the only one that receives the forwarded requests, similar to CDN peer-assisted systems;

- A peer might initialize itself pointing out its willingness to be part of the emergency network, in which case the emergency request service would have a peer list composed of such peers;

- The emergency request service could elect a number of peers to be listed on as emergency handlers based on those peers features, such as upload capacity or data availability.

Selection strategies are important in implementations with multiple handlers to select from. Strategies can be as simple as random or round-robin. The emergency request service will not even need to know handler's buffer maps. Success is determined then by how buffers are configured and how synchronized peers playbacks are.

Finally, emergency requests are to be used as a support mechanism, a last resort and avoided at maximum. They aim at solving specific problems but it is a costly operation and that deliver a chunk close to its playback deadline. Upon the arrival at the peer this chunk will have less opportunity to be disseminated compared to chunks obtained through normal requests. In Chapter 4 we discuss in details this problems using a CDN peer-assisted system by mapping requests made to the CDN as emergency requests.

2.5 Logging

Ping messages that are sent to the bootstrap can carry peer performance data. Using the bootstrap as the logging server increases the traffic at the bootstrap, but it has the advantage that there is no need for a special server to store log data.

These special pings – or *log* messages – include the following performance data: the number of chunks generated, sent and received per second, requests sent and received per second, duplicated and missed chunks since the last log message, average hop and tries count for each chunk received, input and output partnership size. The list can be easily extended, as new measures are envisioned and required. A more detailed chunk performance data is also sent in each message. This chunk data relates to the last *sample chunk* received before sending the log message. For simplicity and comparison between peers we consider one sample chunk every *buffer size* chunks received. Its data is composed by its chunk ID, hop and tries count, and timestamps indicating when it was generated or consumed. All that data is used to get useful insights about latency, playback deadline miss rate, and the distance that a chunk travels before being delivered. These are especially interesting because they are very hard (impossible in some cases) to be captured from commercial systems.

Moreover, there is a system option to enable client-side logs. These logs contain information about every chunk expected by the client. It reports if the chunk was either received normally, missed, received late or received more than once. Also, there are details about to whom that chunk was requested at each try and from whom it was received (in case of receiving), and when each request/receive happened.

Another feature is that each peer periodically sends to the bootstrap a list of which other peers they are connected to. This is done through a different message with two lists of peer IDs (input and output partners) and a timestamp upon bootstrap receival. With those reports, one may track the evolution of the overlay connections over time.

2.6 New Modules/Algorithms

TVPP's object-oriented design includes generic interfaces for many modules. These interfaces provide flexibility to implement new features or algorithms.

For instance, it is easy to create a new kind of message to be exchanged between peers since the send/receive methods receive a Message object as parameter and all messages inherit from Message. So, to create a new kind of message, one must only create a class that extends from Message, add a new entry at the group of message kinds, describe the structure of the new kind of message through an abstract method inherited from Message and introduce a method to handle the reception of that message.

Other mechanisms that are useful to alter are the scheduler policy, the bootstrap peer selection and the connect/disconnect partner selection algorithms. All of them have been implemented using the *strategy* design pattern [Gamma et al., 1994]. To extend these mechanisms, one can develop an algorithm as an extension of a strategy interface and patch the new strategy in with a few lines of code on headers and at the parameter handler. These selection algorithms mainly sort objects by a specific key value, and many strategies can yet be implemented.

Adding a periodical event is possible by extending the Temporizable class. One can create an event, set its period, and push it to the Temporizable list, which triggers it periodically. A few examples of features currently implemented using this are: the peer list request to the bootstrap, the partnership connect and disconnect mechanic and the upload limiter.

Implementing a new peer performance metric and logging it is also simple. Since the log message is basically a wrapper, one can add the new metric at the log message, to the chain that constructs it, and make sure that it will be unwrapped and written at the log file on the other side, in this case the bootstrap.

2.7 Parameters

Some mechanisms described throughout the chapter are essential to understand the basics about P2P live streaming. However, there are additional details that impact systems performance and, consequently, quality of service.

Input and output partnership sizes are factors that impact the overlay connectivity and churn resilience, the local chunk availability, the amount of control messages and the duplicated reception probability (depending on scheduling choices). When combined with the total network size, it affects the diameter of the network and, thus, latency, since chunk storage, processing, and forwarding periods are significant. Several system internal timers are also influential, such as timers between buffer map exchanges and request retries, and partnership link removal timeout. It is also important to balance peer buffer sizes between avoiding big buffer map messages and keeping a desirable storage capacity at peers, so it can offer chunks to its partners. Peer upload capacity is another relevant aspect because it might introduce bottlenecks and overlay average upload capacity must be sufficient to spread the content through the entire network.

Finally, TVPP provides some configurable parameters. Currently, one can fix the following set of parameters:

- *mode*, defines peer main behaviour (e.g., source or a viewer) or can be used to create special kinds of peers (e.g., a free rider or an emergency handler);

- *buffer size*, affects buffer map message sizes and the peer capacity to hold chunks;

- *maximum number of input/output partners*, relates with the overlay, more partners means more chunk sources, a more connected network, but it also increases buffer map messages throughput since they are periodically broadcast to all output partners;

- *request limit*, defines the amount of chunks that can be simultaneously requested;

- *request timeout*, defines how many milliseconds a request must wait before being retried;

- *retry limit*, defines the amount of times that a chunk can be requested if previous requests fail;

- *unresponsive input/output partnership timeout*, define how many seconds a peer must wait before removing an unresponsive input/output partners from its lists;

- *upload and download limits*, restrict a peer to send or receive a maximum number of bytes using a leaky bucket;
- *bootstrap's peer subset selection algorithm*, the subset of peers returned by the bootstrap can be random, oriented by IP distance or RTT (round-trip time) between peers, but one can create his own selection strategy;
- *partnership candidate selection algorithms*, for connecting or disconnecting a partner, these algorithms currently are the same as above since they are also peer selection strategies;
- *chunk scheduler algorithm*, another peer selection strategy but during chunk scheduling.

2.8 Comparison with SopCast

We have compared TVPP with SopCast. The latter is a known commercial system which has been targeted by several studies [Vieira et al., 2009; Horvath et al., 2008; Ali et al., 2006; Tang et al., 2009]. As in previous studies, we also have crawled the network, analyzed traffic, and recreated network graphs to obtain SopCast data. Our results indicate that both systems behave similarly which, in turn, shows that our system is as efficient and it can produce reliable, accurate and more comprehensive analyses of P2P live streaming systems. We have compared traffic and network metrics such as server and peers loads, control overhead rates, network size and diameter, partnership sizes, and average shortest paths.

2.8.1 Experimental Setup

Firstly, we have conducted a SopCast experiment with 60 minutes duration using approximately 500 unreliable geographically-dispersed peers. Peers have been partially synchronized, and using *cron* we have made all peers join and leave the overlay simultaneously. We have streamed a 100 minutes video, which looped continuously, and with an average stream encoding rate, R, of 400Kbps. As a commercial system, SopCast does not allow further setup.

Then we have conducted the same experiment over TVPP trying to emulate SopCast behavior. We have limited TVPP server output partners to 10 peers based on Tang et al. [2009]. Limits for $\mathcal{I}_p$ and $\mathcal{O}_p$ are 50 partners for each p based on SopCast experiment observations, as shown in Section 2.8.3. Selection and scheduling algorithms have been setup to the random policy.

To record traffic for each peer we have used *tshark*, a console version of Wireshark[6] which is a network protocol analyzer based on *tcpdump*. We have chosen to use these logs from both systems while comparing them to avoid biasing results by using different data acquiring methods. However, TVPP performance data has been used to validate its logs. Through traffic logs, we have analyzed both traffic and network behavior. We have dropped the initial and final 300 seconds of each log to avoid any interference from peers joining/leaving the overlay.

We have calculated peers download and upload rates for both control and data packets. We have considered a conservative threshold of 200 bytes to classify control packets on SopCast in accordance with Tang et al. [2009]. On TVPP we have set this threshold to 1300 bytes, since no control packet is larger than this. Section 2.8.2 presents a histogram for packets size, a comparison table with server overhead and load, and cumulative distribution functions of upload and download rates for each peer for each second.

Network metrics cannot be directly extracted since they are calculated over overlay graphs. Peer traffic logs have been merged into a time-sorted log. To represent the partnership graph through time, we have extracted graph snapshots from the sorted log. Vertices and edges are representations of peers and partnerships, respectively. Interarrival time of keep-alive messages on SopCast is less than 2.5 seconds in more than 95% of the cases [Tang et al., 2009]. That assumption has been made also valid for TVPP through buffer map message exchanges every second and a 3 second partnership timeout setup. For both systems, we have considered that a partnership starts after any message is exchanged between peers, and it ends after they stop exchanging packets for 3 seconds or more. Thus, we have analyzed overlapping snapshots within 3 seconds for every observed second, i.e., snapshot t is formed by any peer interaction between time t and $t+2$, $\forall t = [300, 3298]$ seconds. Section 2.8.3 presents network size, diameter and average shortest path, and cumulative distribution function of peers degrees (partnership size) and peers's shortest path to server. These metrics are further detailed below.

Network size is the number of vertices present in a snapshot. Degree of a vertex v is the number of edges which touch v [Skiena, 1991]. A shortest path, or distance, $d(u,v)$ between two vertices u and v of a finite graph is the minimum length of the paths connecting them [Diestel, 1997]. Diameter is the longest shortest path (i.e., the longest geodesic) between any two vertices (u,v) of a graph $(max_{u,v}d(u,v))$ [Skiena, 1991].

[6]http://www.wireshark.org/

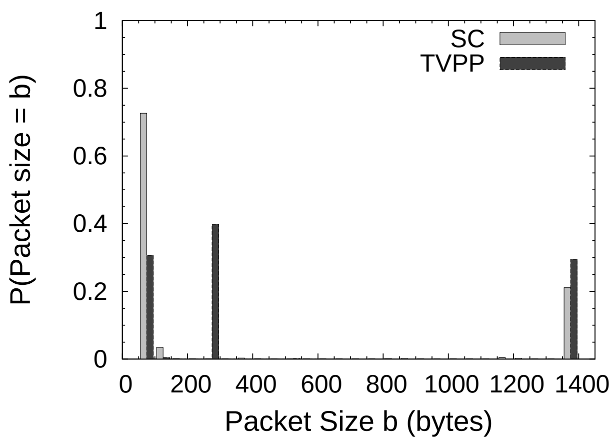


Figure 2.1. Pdf of packet size for each system.

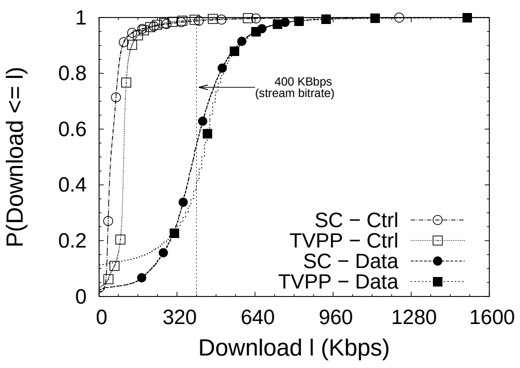
Table 2.2. Traffic-related Metrics

	Download Overhead	Upload Overhead	Server Contribution
SopCast	14.71%	18.47%	3.05%
TVPP	21.32%	24.45%	2.30%

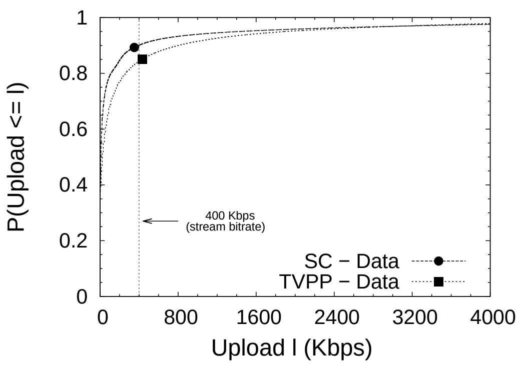
2.8.2 Traffic Analysis

Figure 2.1 shows packet size distribution for SopCast (SC) and TVPP. We have grouped data in 50 bytes buckets. Each system has a different distribution, which indicates that these systems use different packet types. SopCast shows a very high probability of packets below 150 bytes, a smaller probability of packets above 1350 bytes and minor probabilities of packets with size between 150 and 1350 bytes. The first two types might be control and full media packets [Tang et al., 2009; Ali et al., 2006], however there is no consensus about the nature of the others. On TVPP three types of packets occur: small control packets (communication with bootstrap, partnership agreement and chunk requests) below 100 bytes, medium control packets with near 250 bytes and data packets with more than 1350 bytes.

Medium control packets are buffer maps and their actual size will vary according to buffer size setup. Overhead caused by these control messages also scale with output partners size. According to Table 2.2 and Figure 2.2(a), TVPP control packets have been responsible for a greater overhead compared to SopCast. Two ways of achieving lower overhead would be: to increase the period between buffer map messages proportionally to output partners size, which could compromise the scheduling algorithm, to group chunks so a buffer map index would represent n chunks instead of one, reducing map's size, or to reduce output partners limit, reducing overlay connectivity.



(a) Download grouped by control and data packets



(b) Upload restricted to data packets

Figure 2.2. Cdf of download and upload rates for each peer for each second for each system. R is $\approx$400Kbps.

Figure 2.2 shows a cumulative distribution function of download and upload rates for each peer for each second. It can be observed that both systems have similar distributions of control and data rates. Figure 2.2(a) shows that TVPP and SopCast peers have similar behavior while downloading and they both tend to have a download rate near R. Moreover, data curves extremes show that peers in both systems eventually receive nothing or download in bursts.

Figure 2.2(b) shows upload rates distributions. In both systems, Wireshark logs show that there is a considerable number of peers that download what they need but do not contribute uploading to the P2P system in a similar fashion. This is called free-riding. In the other extreme, a few peers contribute a lot, supporting almost the entire network through uploading. As Figure 2.2(b) shows, generally TVPP peers have higher upload rates than SopCast. However, at extreme upload rates (on the right tail not shown) SopCast's curve surpasses TVPP's. Peak upload rates for SopCast and TVPP have been as high as 150 and 120 times R, respectively. This indicates that fewer peers share the upload burden on SopCast compared to TVPP.

Finally, Table 2.2 shows server upload contribution. SopCast server has been responsible for serving 3.05% of all traffic while TVPP server have contributed with almost 2.30%. As peers join the overlay increasing its size, they help to reduce the burden of uploading data imposed to the server. However, Figure 2.3 shows that TVPP had almost 10% less peers than SopCast. Since the server had the same partners limit (around 10) in both runs, TVPP server partners have directly asked less data to the server.

2.8.3 Network Analysis

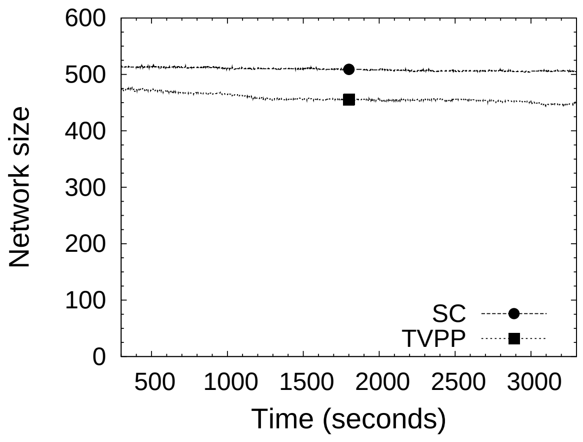


Figure 2.3. Network size in each snapshot.

Figure 2.3 shows the number of peers present in each experiment through time. TVPP overlay have had fewer peers than SopCast's because experiments were conducted on different days and a different subset of active peers have been available for each day. SopCast experiment have started with nearly 520 peers and ended with approximately 500 while TVPP's went from 475 to 450. Size reduction for both systems has been similar. Curves noise results from snapshot creation methodology, meaning that any peer might eventually not be present in a snapshot despite being still connected to the overlay.

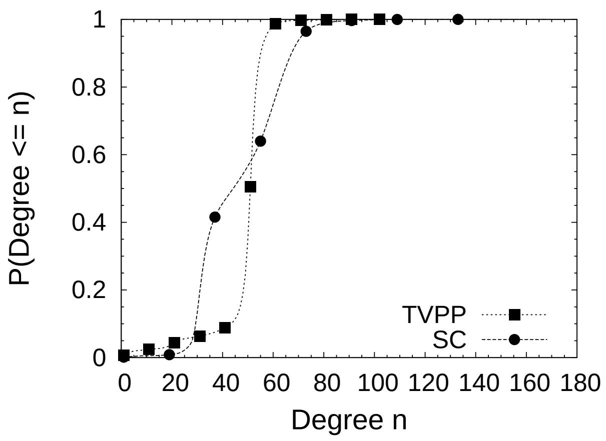


Figure 2.4. Cdf of degree of each peer in all snapshot. TVPP was limited in 50 partners but even if a peer has reached that limit, it still can receive partnership requests.

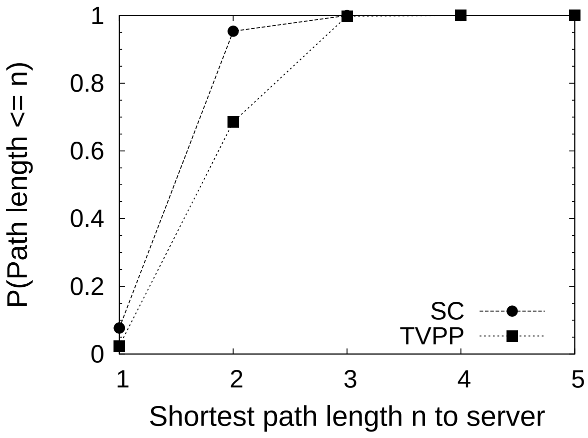


Figure 2.5. Cdf of shortest path lengths from each peer to the server in all snapshot.

Partnership limit parameter choice has direct implication on graph connectivity and impacts several network metrics. Having more partners increases the probability that a peer reaches any other peer with a smaller number of hops or that more partners are connected between themselves. However, there is a trade-off between this parameter and overhead, as already explained. Also, if all peers have been connected between themselves (making a fully connected graph) certainly an even smaller group of peers would concentrate the upload task. Figure 2.4 shows distributions of peer degrees which approximate partnership size. SopCast peers mostly had around 30 to 80 partners. This have motivated us to setup TVPP $\mathcal{I}_p$ and $\mathcal{O}_p$ to accept a maximum of 50 partners. Even though we fix the parameter, a few peers had less than 40 or more than 60 partners through snapshots. These errors are inherent to the data acquisition methodology used for both systems, which was Wireshark logs concatenation. Although a peer has been limited to 50 partners, any other peer could have sent a message trying to connect to a "full" peer, which would be identified as a control message between two peers and, consequently, considered a link between then. Thus creating the top right tail on TVPP's curve. Furthermore, a few traffic log files could have been corrupted or missing causing edges or vertices to be lost.

Figure 2.5 shows a cumulative distribution function of lengths for shortest paths between each peer and the server. Assuming that all connections are free of errors, with no bandwidth limit or link delay, shortest paths indicate the best path to take, as fewer hops from server to peer lead to lower latency. Peers closer to the server receive chunks first, early announce this data availability and are more likely to be the ones forwarding chunks to their partners. The intuitive trade-off is that, in general, a peer closer to the server receives more requests. Systems should be concerned with

Table 2.3. Network-related Metrics

	Diameter		Average Shortest Path	
	Mean	σ	Mean	σ
SopCast	3.692	±0.022	1.952	±0.004
TVPP	4.361	±0.023	2.318	±0.054

fairness, aiming at egalitarian load distribution among participants. This helps to explain upload distribution seen before (Figure 2.2(b)). Nevertheless, connections cannot be treated as ideal. Traffic can easily surpass bandwidth and physical links can become bottlenecks. Thus shortest path lengths are an imprecise estimative of how many hops data travels until arrives at peers.

Table 2.3 presents the average diameter and average shortest path for all snapshots of each experiment, with a confidence interval of 99%. Smaller diameter and average shortest path are better in ideal scenarios but do not necessarily represent the best network organization or paths to take in real world. Despite the small difference between the values for each system, both metrics are slightly higher for TVPP than SopCast which might lead to higher latencies.

Finally, we have observed that under traffic measurement TVPP and SopCast performed similarly. Thus, TVPP is capable of behaving much alike SopCast regarding chunk delivery. Some metrics above have shown that network graphs formed by each system are statistically different. Unfortunately, usual traffic analyzis and offline graph reconstruction methods lack depth to fully understand implications - such as latency impacts - of this differences. Also, these methods might provide incomplete or imprecise information which makes controlling systems verbosity an advantage. Such control on TVPP is responsible for delivering extra statistics such as chunk hop count, chunk loss and latency which could not be extracted from SopCast for comparison.

2.9 Conclusions

We have presented TVPP, an academic research-oriented P2P live streaming system designed to provide a similar service to popular proprietary commercial systems, such as SopCast. We have exposed several details of TVPP design and architecture in the light of overlay construction and chunk scheduling theory. We have also described the design of an emergency request feature. We have shown what TVPP is currently able to log, and some ways on how it can be expanded for other researches. We have introduced the current parameter set with the options that can be fixed at runtime. Finally, we have highlighted benefits of TVPP usage through an experimental comparison with SopCast about traffic and network related metrics.

Chapter 3

Chunk Loss Characterization

In this chapter we evaluate chunk loss in several different scenarios. Characterization of chunk loss plays a key role in our objectives. It allows us to understand **1)** what are the reasons for failure, **2)** if the reasons are protocol, system or network related, **3)** what is the probable frequency or impact of each fault, and **4)** what approaches can be used to solve them. We draw many insights to chunk losses both in resourceful and bandwidth constrained networks. From these, we highlight SURE, a mechanism that significantly reduces performance issues associated with overlooking unresponsive peers. Results obtained with the characterization make it easier to propose and develop other solutions to prevent losses.

3.1 Introduction

A chunk loss happens at a peer whenever the given chunk misses its playback deadline. Once it occurs, either the player will have to wait until that chunk arrives – delaying the peer in relation to others or eventually freezing exhibition – or it will skip it – exhibiting badly decoded frames depending on system design and video encoding algorithm. For the viewer, watching a show that keeps stuttering or with broken frames is undesirable, annoying and, indicates system's bad quality. These failures lead to decreased viewers confidence in the system, viewers abandonment and system collapse.

Chunk loss may depend on a number of system design peculiarities including network topology, scheduling, incentive mechanism, etc [Moltchanov, 2011]. Furthermore, systems are also subject to runtime dynamic conditions, such as peers that not share content (free riders), local neighborhood bandwidth constraints or the unavailability of a chunk in the neighborhood within its deadline.

We have done a chunk loss characterization for experiments run on TVPP. The goal was to identify common reasons and patterns on delivery and failure situations. We have addressed questions such as "Do chunks losses occur?", "How frequently?" and "Why?", and we expand those to grasp the influence of available bandwidth, load at the peers, neighborhood, and time, over chunk losses. We explore different features of chunk loss as we investigated results for both resourceful, and bandwidth constrained scenarios. Section 3.2 describes default parameters and scenarios used throughout those experiments.

At the resourceful scenario (Section 3.3), we have investigated chunk loss in three different scopes: system, peer and chunk. We have drawn conclusions about system-wide losses, partnership bottlenecks, clustering of losses in time and in peer groups, and more.

At the bandwidth constrained scenario (Section 3.4), we reduce available bandwidth through the increase of free riding in the overlay. We setup two types of free riders (Section 3.2.2) – conscious and oblivious – and we vary their fraction from 0% to 95% of peer population. We evaluate system-wide metrics to focus at chunk losses, and also latency, as a function of incremental constraint and free riders influence. As more peers become free riders, the significant cost of overlooking unresponsive peers stands out among performance issues. This has led to the proposal of Simple Unanswered Request Eliminator – SURE (described at Section 3.4.3).

It has been observed that system-wide chunk loss highly depends on network conditions and system design. Nevertheless, our findings suggest that P2P live streaming can support uncooperative peers up to a certain point with nearly the same quality as on a resourceful scenario. Finally, we conclude by discussing solutions to avoid chunk losses for each of our findings.

3.2 Parameters and Scenarios

Experiments have been run on TVPP. As mentioned in Chapter 2, TVPP is a mesh-pull system with very flexible parameter changing capability. TVPP default setup is explained in Section 3.2.1. As we introduce bandwidth constrainment to the default setup, TVPP has been modified to allow free riding client behaviors described in Section 3.2.2. Finally, our evaluation relies on experiments deployed on PlanetLab, a realistic testbed for distributed systems. A description about PlanetLab is given in Section 3.2.3.

3.2.1 Default Setup

Mesh construction relies on the *Random* candidate selection policy to select and connect peers, thus creating random topologies in each run. Partnership sizes, $\mathcal{I}_p$ and $\mathcal{O}_p$, in every peer are set to 20, channel source included. Pings between peers and to the bootstrap are exchanged every second. Peers request a new peer list to the bootstrap every 30 seconds and try to fill their input partners set. Partnerships are undone after 3 seconds without contact from a partner. Bootstrap removes a peer from the channel list after 10 seconds without contact.

Regarding the exchange of chunks, scheduling is done by requesting the *earliest-deadline-first* (EDF) chunk to a random valid candidate peer. A valid candidate is a partner that has announced that it possesses the chunk requested through its buffer map messages. Each chunk carries around 1500 bytes, the Ethernet MTU. Peer buffer sizes are set to 1600 chunks. Consequently, buffer map messages, which represent buffers through bit maps, are restrained to around 200 bytes. Peers can have any number of simultaneous active requests. Requests time out after 0.5 seconds. Upon failure, requests can be retried 2 more times.

We have configured video and bootstrap servers in our university's network and have used between 486 and 489 available PlanetLab nodes as peers. The video server streams a $\approx$ 420 kbps variable bitrate video. This translates as about 40 chunks per second and a 40 seconds sized buffer.

PlanetLab nodes may be running multiple simultaneous experiments, so each host has variable CPU and bandwidth constraints at each time. As a default, client-side upload bandwidth restrictions are not used. Restrictions are given by local conditions of the set of nodes used in each run. It is important to note that TVPP does not know or measure node bandwidth.

All peers join the live transmission during an initial period of 60 seconds, with joining times chosen randomly following an uniform distribution. Each experiment lasts 8 minutes, and we discard data from the first and last 90 seconds (i.e., the warm up and cool down periods). All results are based on 10 runs for each experiment. The amount of data obtained with these repetitions is enough to produce statistically solid results, with low coefficients of variation for the reported averages.

We quantify the quality of a P2P live streaming using mostly chunk playback deadline miss rate, or chunk loss. Eventually, chunk latency is also analyzed in order not to be deceived by good chunk loss results that are inconsistent with a "good" latency. We define the playback deadline miss rate as the fraction of chunks that are not received before their playback deadline. Missed chunks cause flickering or interruption, specially when many chunks are lost in sequence. Chunk latency, also called diffusion latency, is the delay between the creation of a chunk – at the video server – and its reception by a peer. High latency causes undesirable conditions for viewers as peers will play outdated content (e.g., a neighbor cheering a goal that you will watch a few seconds from now).

3.2.2 Free riding

Free riders are peers that do not contribute with upload capacity. The replacement of peers for free riders causes restrictions to the initially available bandwidth and to the upload distribution. We have observed how these restrictions affects chunk loss and latency by linearly increasing the fraction of free riders. We couldn't find any studies where real P2P live streaming systems were analyzed for the presence of free riders and how they are commonly presented. The closest paper to point the fraction of free riders on a system discusses the Gnutella case, a file sharing system, where 63% of peers have never answered a file search query [Adar and Huberman, 2000]. Some results presented in the following sections have already been published [Oliveira et al., 2013a].

We define two types of free riders, namely *conscious* and *oblivious*. Conscious free riders inform their partners that they are unwilling or unable to upload data. This behavior may be coded in the software or chosen by users. In our system, conscious free riders request chunks as normal but always advertise empty buffer maps. They have their mode parameter set to free rider. As a consequence, no peer ever wastes time and bandwidth sending requests to conscious free riders.

Oblivious free riders do not inform their partners that they are unwilling or unable to upload data. This behavior may happen if the software does not make provisions for free riders, if the user misconfigures the client, or due to malicious intent. In our system, oblivious free riders request chunks normally and advertise buffer maps with the chunks they have, but never upload data (i.e., never answer requests). They have their upload limit parameter set to zero. Oblivious free riders may receive requests and degrade system performance, as their partners will have to retransmit chunk requests after waiting for answers that never arrive.

We have run experiments varying the behavior and fraction of free riders. For each type of free rider behavior, we vary the fraction of free riders from 0% to 95% in steps of 5%. To maximize the impact of free riding, we do not use any mechanism to choose or abandon partnerships (e.g., reputation systems). In other words, peers do not drop or punish uncooperative partners.

3.2.3 PlanetLab

Planet Lab[1] is a worldwide consortium of research institutions that maintains a global environment for the development and testing of distributed applications. Each institution has one or more nodes in the Internet that operate as virtual machines servers. PlanetLab access accounts are called slices. If an institution maintains at least one node in operation it is granted the right to create slices and every slice has the power to control a set of virtual machines on network nodes.

A user with access to a slice is able to create a virtual machine on each node with an initial minimalist set of applications and restricted space available. These resources must be configured and managed individually. Any activity on the nodes should consider that their network and hardware characteristics are heterogeneous, that a node is not always available, and that it may fail.

However, PlanetLab usage brings many data acquisition advantages. The most obvious one is the possibility of using over one thousand nodes, which helps in recovering more representative and reliable data. Another advantage is that PlanetLab nodes are dispersed both geographically and in diverse networks, thus avoiding that location aspects mask network behavior. Finally, PlanetLab nodes possess real IPs and do not suffer packet filtering. These characteristics prevent the need to address the Network Address Translation (NAT) problem [Bellovin, 2002], where nodes's applications may be difficult to communicate with.

3.3 Analysis of Resourceful Scenario

In this section we evaluate experiments executed under the scenario with no enforced bandwidth restriction. The scenario uses the default parameters setup present in Section 3.2.1 such as overlay size of ≈ 500 peers, duration of 8 minutes, and 10 repetitions. Bandwidth distribution is given by PlanetLab hosts momentary setup, and it can vary between experiments. Still, the environment is considered to be resourceful as results presented through out the section are similar to a scenario in which the average peer upload capacity is limited to ten times the stream bitrate.

We explore chunk loss by examining three different scopes: system, peer and chunk. We start with a broad view through a system-wide analysis until we reach individual chunk failure analysis. We show that, although rare, losses do occur in resourceful scenarios, to many peers and steadily on time. Yet, there is certain underutilized upload capacity while peers do not receive requests. We show neighborhood

[1]http://www.planet-lab.org/

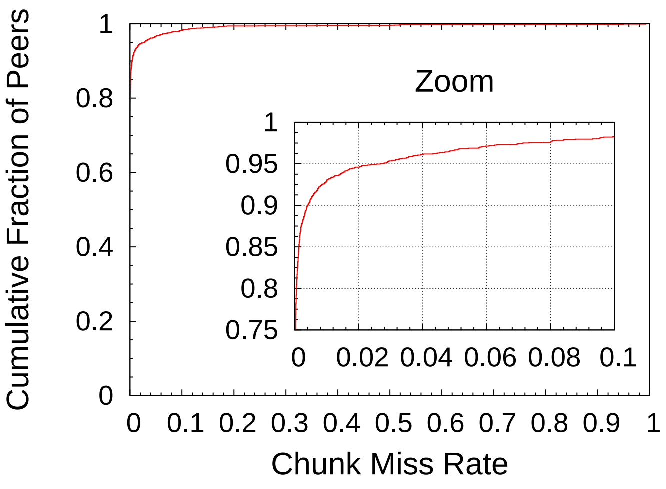


Figure 3.1. Cumulative distribution of chunk loss reported for each peer in a set of runs.

data availability for each attempt of requests that hit and miss and compare it. Finally, we study the loss spread of a specific chunk and the frequency of consecutive losses, which we refer to as chunk loss spatial and temporal locality.

3.3.1 System Behavior Analysis

Even while considering a resourceful scenario, chunk loss can still be observed system-wide. The overall results from this scenario show that:

- 60% of the peers have lost 0% of the chunks;
- 35% of the peers have lost between 0 and 3% of the chunks;
- 5% of the peers have lost more than 3% of the chunks;
- The losses from these 5% last peers represent 86% of the total chunk loss.

Visual disruption caused by chunk loss strongly depends on stream coding algorithm used. Given the highly structured organization of the video streams, the degradation of the received video quality becomes typically noticeable for values of loss higher than 1%, while loss probability of a few percent (3-4%) significantly impair the user quality of experience [Traverso et al., 2014]. Thus, we have chosen to cluster above analysis using the 3% reference (inferior limit to impair the stream).

Figure 3.1 shows the distribution of chunk losses for each peer in all experiments. The smaller figure zooms at the top left part of the curve. We can observe that the knee of the curve occurs at around 92% of the peers with a loss rate of around 0.8%.

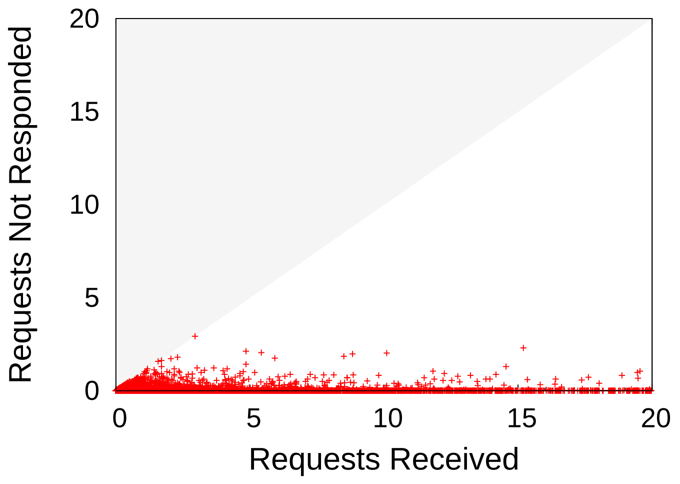


Figure 3.2. Scatter plot between requests received and requests not responded over time normalized by stream rate.

This result can be considered to be very good, but beyond this knee there are many peers that have trouble watching the stream.

If we evaluate the average chunk loss for all peers by time, we can observe that chunk losses have remained stable through time for all experiments. The average chunk loss for all peers for each interval have ranged mostly between 0.5% and 1%.

3.3.2 Peer Behavior Analysis

Do more losses occur at peers that are serving more chunks? Were peers overloaded with chunk requests? Is it a bottleneck issue? In this section, we focus on individual peers and use chunk request metrics as an abstraction to investigate the correlation between chunk losses and peer load or contribution in the network. Figure 3.2 shows a scatter plot between the amount of requests received and the amount of requests not responded for each peer for each second. Since the system has a request retry feature, requests that have not been responded do not necessarily result in a chunk loss, but it is a good approximation of load-related issues. We have expected chunk request received and requests not responded to be related; a peer would fail to respond requests proportionally to the amount of requests received. As Figure 3.2 illustrate, failures could not be associated to peer contribution through a linear dependence. Numerically, a Pearson correlation coefficient of 0.11 indicates that there has been no linear relationship between both variables. From the data, we have extracted that peers have received no chunk requests in 19% of the samples, peers have responded to all the chunk requests in 75% of the samples, and peers have failed to respond to all chunk requests only in 0.4% of the samples.

Thus, we can observe two things: a peer who contributes more does not necessarily fails to respond more, and, although chunk losses occur, there are many moments where peers are not receiving requests, possibly wasting their upload capacity. Unfortunately, with the lack of instant bandwidth measurement we can not establish a relation between peer upload capacity and failures.

3.3.3 Chunk Behavior Analysis

We have analyzed the characteristics of each individual chunk request from all peers. We establish that a chunk *hit* happens whenever a request has been responded in time within three attempts. Conversely, a chunk *miss* (loss) happens whenever a chunk has not been received in time for playback. Chunk losses have represented around only 0.5% of all chunk transfer log, however, this small amount still results in the loss distribution seen in Figure 3.1. We have studied the efficiency of retries, neighborhood data availability for chunk hits and misses, and temporal and spatial locality of misses.

Table 3.1. Distribution of attempts that take for a received chunk to be received.

Attemps	1	2	3
Frequency	98.7%	1.1%	0.2%

Chunks that have been received. Table 3.1 and Figure 3.3 report results for chunk hits. Table 3.1 shows that 98.7% of the hits are from requests that were responded in the first attempt. 1.1% are from the second attempt, and the rest from the third. This suggests that allowing more attempts through the common chunk scheduler will probably result in little improvement over chunk loss.

Figure 3.3 illustrates neighborhood data availability. It presents the number of available candidates whenever a chunk hit happens on the first, second or third attempt. There are few candidates whenever a request is responded within the first attempt. Usually the first attempt is done to the first peer to announce that it has the chunk, a greedy characteristic of TVPP's earliest deadline first (EDF) chunk scheduler policy. Gradually, the chunk is spread to more peers increasing data availability. Thus the subsequent attempts have more candidates to choose from.

Table 3.2 shows how many unique candidates have been used whenever a chunk hit happens on the first, second or third attempt, or when a chunk miss happens. The second and third attempts have been generally made to distinct candidates for chunks that have been received. This result is expected as TVPP selects a random candidate

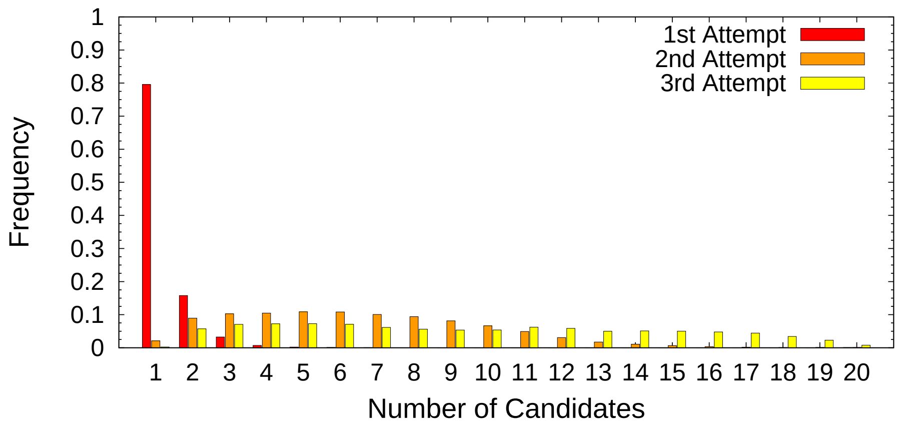


Figure 3.3. Distribution of the number of candidates that could deliver a chunk in each attempt for chunks that have been received.

Table 3.2. Distribution of unique candidates used per request.

Unique candidates	1	2	3
1st Attempt Hit	100.0%		
2nd Attempt Hit	3.4%	96.6%	
3rd Attempt Hit	0.5%	56.9%	42.7%
Miss	25.3%	42.0%	32.7%

to provide the chunk and there are usually more then one candidate to chose from. Results for chunks that have been missed are discussed below.

Chunks that have been missed. For all chunk misses, 15% have happened because there were no candidates for the chunk. For the remainder 85% which have had candidates, we present Figure 3.4. It is important to draw a parallel between this and Figure 3.3. We have observed that missed chunks have had fewer candidates in the subsequent attempts if compared to chunks that hit. This indicates a slower spread progression for these chunks, suggesting eventual issues in the neighborhood such as data unavailability and peer bandwidth saturation.

The smaller number of candidates raises another question. According to Table 3.2, 25% of the requests that miss are being attempted to the same candidate. Why so many? The candidate might be experiencing a transient problem to respond. Candidate distribution whenever the chunk is requested to the same candidate every time is particularly different from previous distributions. Figure 3.5 shows that candidate progression for this scenario is even worse than the general candidate distribution for missed chunks in Figure 3.4. In this case, even after few attempts peers requesting a chunk still have very few candidates to request from. So, the lack of balance in spread

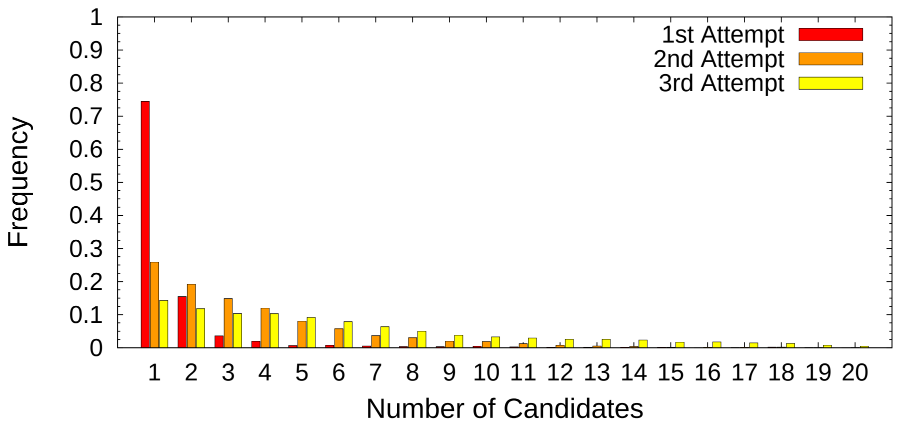


Figure 3.4. Distribution of the number of candidates that could deliver a chunk for each attempt for chunks that have been missed.

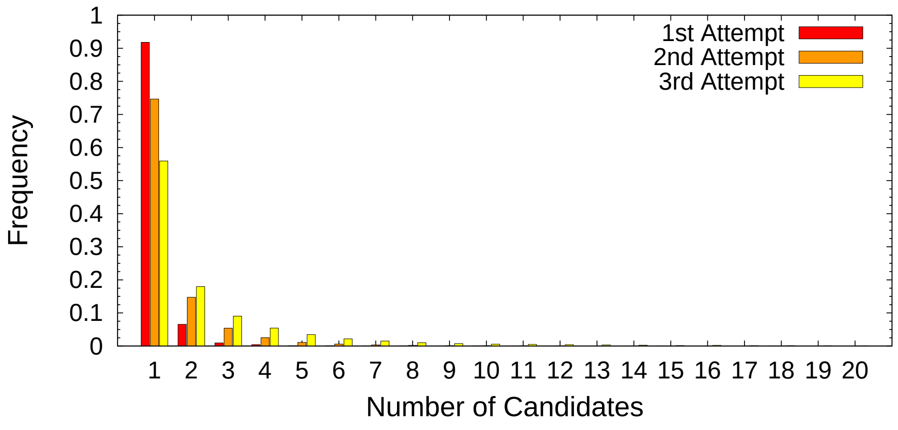


Figure 3.5. Distribution of the number of candidates that could deliver a chunk for each attempt for chunks that have been missed and only requested to the same candidate at all attempts.

may result in this miss scenario.

Temporal locality. We have also investigated bursts of chunk misses, i.e. consecutive chunks miss occurrences. Losing many chunks in a row is undesirable because it accentuates video disruption. Figure 3.6 gathers details about frequency and size of these bursts. The inside figure zooms at bursts with less than 10 misses. The outside figure shows an extended view limited at 100 misses. At the figure, each frequency is weighted by the amount of chunks in the burst. That helps to identify relevant events such as a peer with a huge miss sequence. If we look at the data for frequency alone, instead of frequency multiplied by size, only 1% of the bursts are beyond 10 misses, and just 0.2% beyond 100. These results show that the majority of chunk misses lacks temporal locality, i.e. if we look at a peer, there is a tendency for its chunk misses to oc-

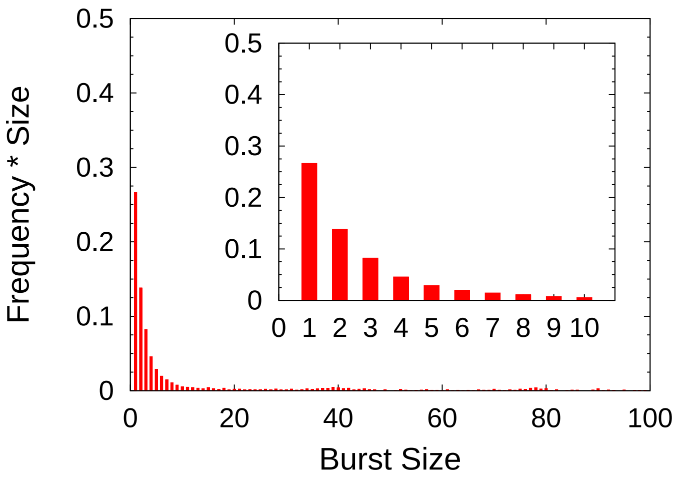


Figure 3.6. Frequency distribution of bursts of chunks missed weighted by their size.

cur individually instead of grouped. Still, few peers may eventually have a continuous issue that impairs their quality of experience (not shown).

Spatial locality. Finally, the last result conveys the spread of chunk misses, i.e. we investigate if a given chunk is lost by several peers in the network or losses have less spatial locality. If a given chunk is lost by several peers that means that the distribution tree for that specific chunk have had issues, probably in the early hops of the tree. However, in Figure 3.7 we present the cumulative distribution of chunks by the percentage of peers that miss a chunk. We have observed that no chunk has been lost by more than 3% of the total amount of peers. From the levels in the figure we can extract that 10% of the chunks were received by all peers, 22% were received by all but one, 25% by all but two, 21% by all but three, 10% by all but four and 7.5% by all but five, leaving 5% of the chunks with losses that reach more than five peers. The majority of chunk misses lacks spatial locality, i.e. if we look at a chunk miss, there is a tendency for that to happen in a small amount of peers.

Summary. The main results of this fine grained analysis for a *resourceful scenario* have been that **1)** retrying had low impact in recovery, **2)** chunks that have been missed were less available, peers have had fewer candidates to request from, **3)** chunk misses generally do not happen consecutively, in long bursts, and **4)** any given chunk has been missed only by a few peers throughout the overlay, which help in a better understanding of losses. Chunk misses have been rare and fairly independent of each other, suggesting that occasional availability issues are their causes, not their consequences.

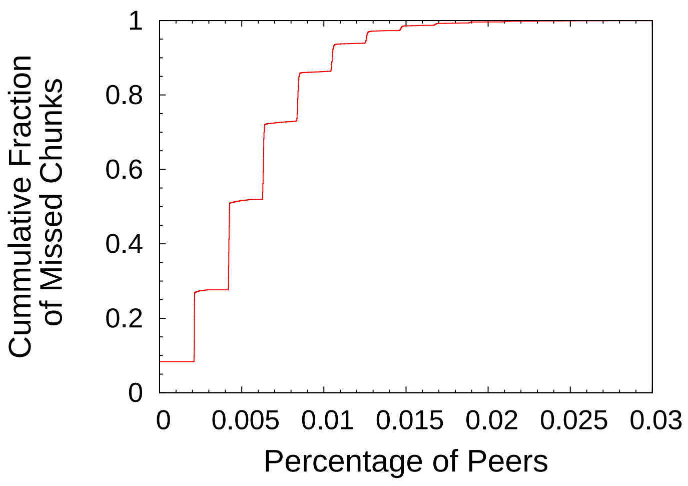


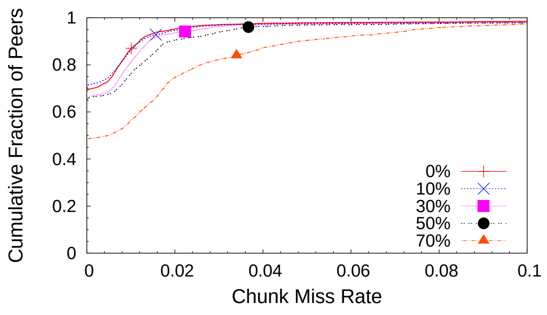
Figure 3.7. Cumulative distribution of chunks that are not received by part of the network.

3.4 Analysis of Bandwidth Constraint Scenario

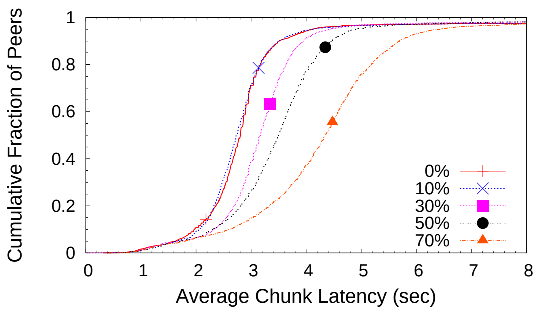
In this section we evaluate the impact of increasing bandwidth constraints – introducing free riders – over chunk loss and latency. As expected, results get worse once we start to reduce available bandwidth. We show that conscious free riders have minor impact on both evaluation metrics, but that oblivious free riders seriously degrade them. We then introduce Simple Unanswered Request Eliminator (SURE) to mitigate the impact of oblivious free riders. Finally, we show that the increased workload caused by free riders is evenly balanced among cooperative peers and is manageable.

3.4.1 Conscious Free Riders

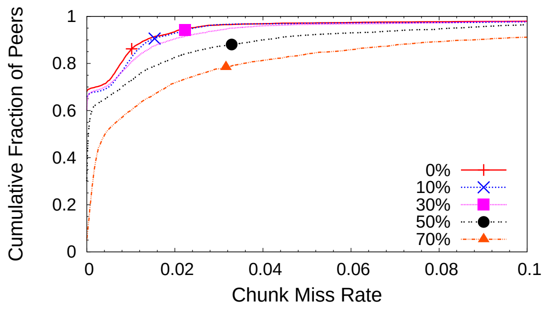
Figure 3.8(a) shows the distribution of chunk playback deadline miss rate (chunk loss) for each peer in a set of runs. We plot various curves varying the fraction of conscious free riders. From what has been observed previously (Figure 3.1), we note that some chunks may miss their playback deadline for factors other than uncooperativeness. For instance, our results include PlanetLab nodes that may be overloaded, lacking enough network bandwidth or CPU to download and process chunks. The chunk miss rate is qualitatively similar for fractions of free riders below 50%. Again, with less than 50% of free riders, around 65% of peers receive all chunks before their playback deadline and 92% of peers experience chunk miss rates lower than 3%. Chunk losses increase significantly with 70% of free riders thus being unnecessary to show higher free riders ratios. Peers have fewer partners that can provide chunks and they may fail to receive a chunk before its playback deadline. If a content provider intends to



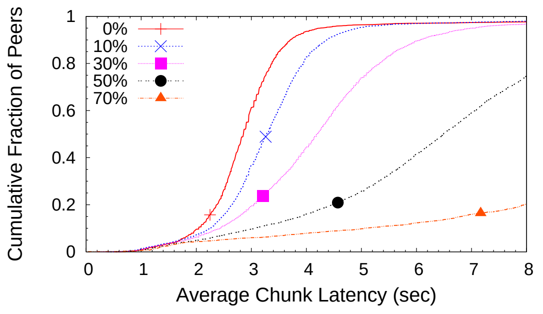
(a) Conscious Free Riders - Chunk Loss



(b) Conscious Free Riders - Latency



(c) Oblivious Free Riders - Chunk Loss



(d) Oblivious Free Riders - Latency

Figure 3.8. Chunk playback deadline miss rate (chunk loss) and latency, for conscious (3.8(a) and 3.8(b)) and oblivious (3.8(c) and 3.8(d)) free riders. Curves are scenarios with indicated free rider ratio.

sustain a system with 70% (or more) of free riders, it may need a hybrid architecture with well-provisioned support peers to cover the missing resources.

Figure 3.8(b) shows the distribution of average chunk latency for each peer in a set of runs. Again, we plot multiple curves varying the fraction of conscious free riders. The average latency is representative of a peer's chunk latencies: the standard deviation of chunk latencies is less than 1 second for 93% of peers. Figure 3.8(b) shows that latency stays stable if the number of conscious free riders is less than 10%. As the fraction of free riders increases, the fraction of partners of a peer willing to contribute decreases. This shortage of contributing partners causes chunk forwarding paths to become longer, meaning a chunk needs to traverse more hops in average to reach all peers. The median chunk forwarding path length is 3.5 for 10% of free riders, but increases to 4.1 for 50% of free riders. Longer forwarding paths increase latency since there are delays associated with buffer map advertisements, chunk requests, and the data transfer itself.

3.4.2 Oblivious Free Riders

When a peer requests a chunk from an oblivious free rider, its request will time out and it will have to be retried later. In particular, the chance that retries are (repeatedly) requested to an oblivious free rider is proportional to the fraction of free riders in the overlay. We compare the number of request retries in scenarios with conscious and oblivious free riders. Retries are rarely needed when free riders are conscious (e.g., when a peer removes a chunk from its local buffer while a request is in transit). Thus, peers usually make a single request per chunk, with a median of 1.007 tries per hop in the chunk forwarding path. When free riders are oblivious, the number of retries to obtain a chunk increases significantly. For instance, with 50% of oblivious free riders, the median number of tries per hop on the chunk forwarding path is 1.4.

Consecutive retries waste bandwidth and time, thus increasing average chunk latency. Moreover, consecutive retries may increase chunk losses if no retry succeeds before chunk playback deadline. Figure 3.8(d) shows the distribution of average chunk latency for each peer in a set of runs. Even 10% of oblivious free riders increase the median of the distribution of average chunk latency to 3.28 seconds, a 15% increase compared to the scenario without free riders. This is significantly worse than having 10% of conscious free riders (Figure 3.8(b)), which has almost no impact on latency. System performance gets even worse when the fraction of oblivious free riders increases. The median average chunk latency is 6.49 seconds if the overlay has 50% of free riders, a 127% increase compared to the scenario without free riders.

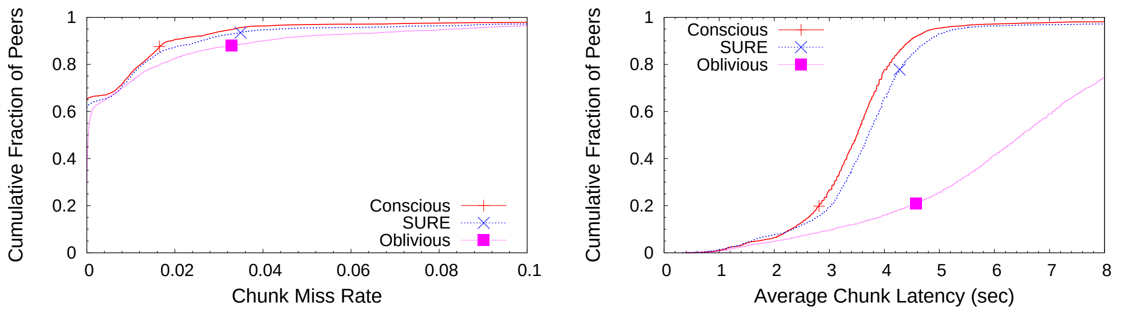
As for chunk losses (Figure 3.8(c)), there are three important things to notice. First, the number of peers that receive 100% of the chunks gets extremely reduced. In this scenario, requesting chunks to a peer that will not respond is a serious problem because it wastes our limited tries. Once the maximum number of tries is reached, peers give up on the requested chunk which results in chunk loss. Second, the visual degradation on the chunk loss figure is not as bad as in latency, because peers eventually find chunks after a few tries but latency delays accumulate between hops. Yet, even small degradation in chunk loss plays a large role on QoS impact. Finally, if we had established a maximum acceptable latency threshold such as 8 seconds, many received chunks with high latencies would be dropped translating into even more chunk losses than what is presented.

Chunk request retries are the major difference between scenarios with conscious and oblivious free riders. If we could avoid chunk requests to peers that are unable to respond – e.g. oblivious free riders – in the first place, we would limit their negative impact on system performance.

3.4.3 Simple Unanswered Request Eliminator

We introduce and evaluate SURE, a modification to our system's request scheduler that avoids excessive retries caused by oblivious free riders. Our goal is to show that even simple solutions can significantly reduce the impact of uncooperativeness on P2P live streaming systems.

SURE maintains a counter of pending requests for each partner. Whenever a peer sends a chunk request to a partner, it increments that partner's pending requests counter. Whenever a peer receives a chunk from a partner, it decrements that partner's counter. The idea is that counters for oblivious free riders will increase rapidly, while counters for contributing peers will remain low. When sending a chunk request, peers choose the partner with the smallest counter among partners with that chunk. If multiple partners have the same amount of pending requests, SURE picks one at random.



(a) Chunk Loss Comparison (b) Latency Comparison

Figure 3.9. Chunk loss and latency comparison over the results of conscious, oblivious and SURE scenarios using 50% free rider ratio.

Figure 3.9(b) shows the distribution of average chunk latency for each peer for 50% of conscious and oblivious free riders (curves "conscious" and "oblivious" in Figure 3.9(b) are the same as "50%" in Figure 3.8(b) and 3.8(d)). We also plot the chunk latency for a scenario with 50% of oblivious free riders when using SURE. With our modification we are able to reduce the number of retries. The chunk latency is almost equivalent to the baseline scenario of conscious free riders; horizontal difference between these curves is less then 0.3 seconds. Results for different fractions of free riders are qualitatively similar (Appendix A).

Similarly, Figure 3.9(a) shows the distribution of chunk playback deadline miss rate for 50% of conscious and oblivious free riders; as well as the miss rate for oblivious free riders when using SURE. Again, this technique reduces chunk losses to levels equivalent to those of the baseline scenario with conscious free riders. In particular, SURE reduces by half the fraction of peers with chunk losses higher than 3%.

SURE works because it identifies uncooperative peers with few interactions. When a peer joins the channel, all its partners have zeroed pending requests counters and it may send requests to uncooperative partners. However, uncooperative partners's pending requests counters increase quickly and they are avoided until the end of the partnership. Another advantage of SURE is that it balances the load among peers. Consider that two partners have many desirable chunks and the same value on their pending requests counters. When SURE issues a request to one of the partners and increments its pending requests counter, it will prefer the other partner for the next request as it will have a smaller counter. Finally, "SURE" depends only on local and individual information, eliminating the need for measurements between the peer and its partners.

SURE shows that even simple solutions allow P2P streaming systems to mitigate most of the impact of uncooperative peers on system performance. We leave the evaluation of recovery mechanisms (e.g., slowly decrementing counters over time, optimistic unchoke) as future work. Recovery mechanisms may improve SURE's performance for long stream sessions or in scenarios where peers change behavior overtime. Finally, we note that other alternatives to eliminate unanswered requests are possible, such as using round trip time instead of pending request as a decision metric.

3.4.4 Workload Distribution Induced by Uncooperative Peers

In previous sections we have discussed how free riding increases chunk loss but is manageable up to a certain point. Yet, a natural side question is what happens to workload distribution when peers do not contribute to the system's aggregate upload capacity; there must be nodes bearing the workload. We categorize peers using their *cooperation level* C, i.e., the ratio of peer average upload rate to the video stream bitrate throughout the experiment. Peers are classified as free riders if $C = 0$, uncooperative if $0 < C \leq 1$, cooperative if $1 < C \leq 5$, and altruistic if $C > 5$. To understand workload distribution, Figure 3.10 presents the fraction of peers in each category for an increasing fraction of free riders. For instance, checking the y axis while looking at 30% induced free riders (in the x axis) we get 41% uncooperative, 25% cooperative and 4% altruistic peers. We extract data from experiments using SURE over oblivious free riders, but results with conscious free riders are quantitatively similar.

Even when there are no free riders in the system, most peers are classified as uncooperative. A large fraction of network load is carried out by a small number of

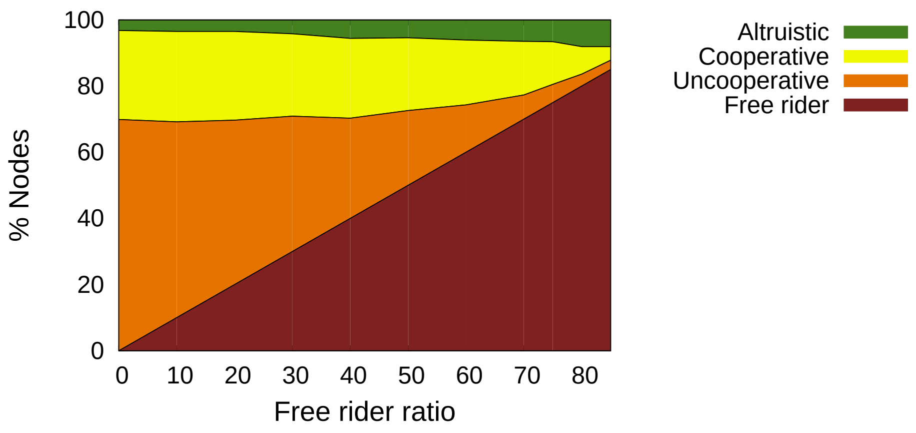


Figure 3.10. Load distribution for several free rider ratios. (free riders, $C = 0$; uncooperative, $0 < C \leq 1$; cooperative, $1 < C \leq 5$; altruistic, $C > 5$)

cooperative and altruistic peers. This unbalanced load distribution happens because **1)** TVPP does not use any strong load balancing scheme and because **2)** peers closer to the server receive chunks earlier and have more time to redistribute chunks than peers far from the server. Uncooperative peers occur due to intrinsic protocol mechanisms, as also observed in SopCast [Oliveira, 2010].

As the fraction of free riders increases, the workload of cooperative peers increases and they shift to the altruistic category. However, the average workload per peer in the cooperative and altruistic category remains stable, close to 2 and 8 stream rates, respectively, if there are less than 70% of free riders. The maximum neighborhood size has been set to 20 peers in our system configuration, which creates an upper bound on each peer's contribution.

Figure 3.10 shows that 70% of peers are uncooperative even when there are no free riders. Peers remain fairly stable in their categories, suffering significant changes beyond 50% free rider ratio. Previously cooperative peers are turned into free riders in scenarios with more than 70% of free riders, resulting in significant workload increase on the remaining cooperative and altruistic peers. After this turning point, system behavior converges quickly to a client-server model, where most peers contribute nothing and a few well-provisioned peers sustain all the workload. Finally, we note that although 70% of free riders is the upper limit before system collapse, performance degradation starts after 50% of free riders, as discussed above through Figure 3.8.

With less then 50% of peers as free riders, the lack of contribution incurred by uncooperativeness is manageable; workload is evenly balanced among contributing peers, and chunk loss and latency impacts are tolerable. But even when there are no free riders, several peers behave uncooperatively although they probably have spare bandwidth while there are chunk losses happening around the overlay. If we had a mechanism beyond the overlay maintenance (such as Emergency Request Service, Section 2.4) to link uncooperative peers with spare bandwidth to peers which are experiencing problems to request specific chunks, we might reduce chunk losses and exploit this spare resource. Finally, we argue that denying service to uncooperative peers may not be the best long-term approach; while content providers would probably benefit from having a higher number of viewers, our findings suggest that P2P live streaming can support uncooperative peers and possibly even explore their resources. Conclusions

Chunk loss is an important issue present in several P2P live streaming scenarios which can reduce the quality of experience for users. The chapter describes a set of experiments run on TVPP mostly aimed at the chunk loss characterization. We describe parameters and scenarios generally used through out experiments. Then we evaluate both a resourceful and a bandwidth constrained scenario.

We have found that for a resourceful scenario most peers do not have chunk loss problems, while a few peers present significant losses. System-wide average chunk loss has been stable in time. While losses do occur, there are many moment when peers had no requests made to them, which suggests that the overlay organization might be improved. Mostly, requests are responded in the first attempt, by the first peer to announce that it has the chunk. Chunks that have been missed have had a worse chunk spread – having less candidates to request from – than chunks that have been received. Finally, chunk loss lacks spatial and temporal locality. For this scenario, results lead us to believe that losses are somewhat erratic, and thus an emergency request mechanism (described in Section 2.4) would be effective to virtually overcome all these losses.

Losses get worse, yet are still sustainable, once we start to reduce bandwidth through the introduction of free riders. Observing conscious free riders we noted that chunk loss has been qualitatively similar for fractions of free rides below 50% while latency has increased by 1s. However, the same fraction of oblivious free riders cause a degradation significantly worse, mostly triggered by the increase in retries. Consecutive retries waste bandwidth and increase average chunk latency. We have introduced SURE, a modification to our system's request scheduler that avoids excessive retries caused by oblivious free riders because it identifies uncooperative peers with few interactions. Using SURE we have obtained similar results for the same amount of conscious and oblivious free riders. While studying chunk loss, we have shown that P2P live streaming systems can sustain 50% of free riders with negligible degradation. We also have found that the workload incurred by free riders is evenly balanced among contributing peers and is manageable. We argue that denying service to uncooperative peers may not be the best long-term approach; our findings suggest that P2P live streaming can support uncooperative peers.

Chapter 4

AERO: Adaptive Emergency Request Optimization

In this chapter we analyse the chunk delivery mechanism present on CDN peer-assisted systems. Live streaming platforms employ advanced mechanisms to guarantee continuous and scalable video playback to large user bases. One such mechanism is CDN-P2P streaming, where servers are hosted on content distribution networks and clients help disseminate content over a peer-to-peer overlay. Hybrid CDN-P2P platforms are slightly different from pure P2P and, therefore, results in this chapter were obtained with SmoothCache [Roverso et al., 2012, 2015] instead of TVPP. In CDN-P2P streaming, all data chunks can be obtained from CDN. Those who are interested in the content form a support P2P overlay, which should be queried first for data. If a chunk cannot be found or a request fail to be responded by its partners, all peers know CDN servers addresses and will request chunks directly to them. We mapped this behavior as an equivalent of TVPP's proposed emergency request service (Section 2.4). Emergency requests allow the retrieval of nearly missed pieces and avoid chunk losses, guaranteeing continuous playback. We show that emergency requests deliver a chunk close to its deadline that has little to no time to be disseminated over the peer-to-peer overlay. We present AERO, a mechanism that dynamically adjusts the rate at which CDN-hosted servers seed content pieces into the peer-to-peer overlay as a function of network conditions. Our evaluation of AERO under diverse conditions shows it reduces emergency requests, guarantees efficient peer-to-peer dissemination, and provides significant server upload bandwidth savings.

4.1 Introduction

Peer-to-peer (P2P) content distribution improves the scalability of live streaming platforms and reduces costs. One disadvantage, however, is that P2P distribution is best-effort and cannot guarantee quality of service (QoS). This is specially impactful in real-time live streaming [Hei et al., 2007b; Lu et al., 2009]. Modern streaming platforms complement P2P distribution with reliable high-capacity servers hosted on content delivery networks (CDNs) [Roverso et al., 2015, 2012; Lu et al., 2012; Zhao et al., 2013; Yin et al., 2009; Mansy and Ammar, 2011]. To guarantee QoS, platforms allow peers to issue 'emergency' requests to servers whenever the P2P overlay fails to distribute a video chunk before its playback deadline [Roverso et al., 2012, 2015; Payberah et al., 2012b; Kreitz and Niemela, 2010; Zhang, 2005; Jin et al., 2013]. CDNs are well suited for such a solution as they are built to handle variable load and can guarantee QoS up to the contracted capacity.

Streaming cost in hybrid CDN-P2P streaming platforms is proportional to bandwidth utilization at the CDN-hosted servers. Server bandwidth utilization is composed of two elements: **1)** the number of peers of the P2P overlay which are being seeded by servers, and **2)** the rate of emergency requests. Increasing the number of seeded peers decreases both the dependence on P2P distribution and the number of emergency requests.

The challenge is to balance the number of seeded peers and emergency requests to minimize cost. Current platforms compute the number of seeded peers as a function of peer upload bandwidth [Roverso et al., 2012, 2015; Simoni et al., 2014; Yin et al., 2009] or peer arrival and departure rates [Tewari and Menon, 2009; Yin et al., 2009; Mansy and Ammar, 2011]. Unfortunately, we show that such allocations may become suboptimal depending on network conditions (Section 4.4). We also show that emergency requests may undermine P2P distribution efficiency. In these cases, peers receive responses for emergency requests close to playback deadlines and do not have time to widely redistribute these chunks through the P2P overlay (Section 4.4).

In this chapter we present Adaptive Emergency Request Optimization, AERO, a mechanism to minimize streaming cost (Section 4.6). AERO dynamically computes the number of peers of the P2P overlay which are being seeded by servers to reduce total bandwidth consumption on servers. When the P2P overlay is distributing video chunks efficiently and peers have unused upload bandwidth, AERO decreases the number of seeded peers. The rate of emergency requests does not increase and total bandwidth utilization at servers decreases as peers start using their spare bandwidth to take over distribution driven by the peer that is no longer seeded by CDN servers. When the

P2P overlay is not distributing video chunks efficiently but peers have unused upload bandwidth, AERO increases the number of seeded peers. The bandwidth consumption from the additional seeded peers is compensated by a greater reduction in the rate of emergency requests, as seeded peers using spare upload bandwidth possibly start new chunk redistribution chains in the P2P overlay.

Our evaluation shows that the rate of emergency requests is a good estimator for P2P distribution efficiency (Section 4.7). AERO compares observed and expected rate of requests done to servers to adapt the number of seeded peers to variable overlay conditions. We evaluate AERO across different scenarios varying peer upload bandwidth, fraction of free-riding peers, peer churn (up to flash crowds), overlay sizes, and overlay maintenance policies. Our results show that AERO increases server upload bandwidth savings by up to 30% compared to static configuration of the number of seeded peers.

AERO is flexible and can be used in combination with previous mechanisms for CDN-P2P live streaming. AERO is complementary to P2P overlay organization mechanisms [Castro et al., 2003; Zhang et al., 2005; Li et al., 2008; Tang et al., 2009; Traverso et al., 2014]; in particular, AERO does not impose any structure on the P2P overlay and can be applied to both tree-like and mesh overlays. AERO is also complementary to algorithms of peer sampling [Roverso et al., 2013] and chunk scheduling [Zhao et al., 2009].

As increasing broadband penetration drive up media consumption, content providers need efficient distribution mechanisms. We believe AERO is useful to large-scale content providers with global overlays that have variable and unpredictable performance, as well as to small-scale content providers who have less resources and are proportionately more heavily impacted by operational costs.

4.2 CDN-P2P Live Streaming

CDN peer-assisted (CDN-P2P) live streaming systems [Roverso et al., 2015, 2012; Lu et al., 2012; Zhao et al., 2013; Yin et al., 2009; Mansy and Ammar, 2011; Payberah et al., 2012b] share many mechanics with pure P2P systems (such as TVPP, Chapter 2). CDN clients form a peer-to-peer overlay between themselves and, whenever content cannot be found over P2P, they resort to CDN servers. In this section we present a broad view of a CDN-P2P live streaming system considering bootstraping, chunk delivery from CDN to peers, chunk scheduling, overlay maintenance, and others. Some of these features are similar to those presented in Chapter 2 while others are particular to the studied platform.

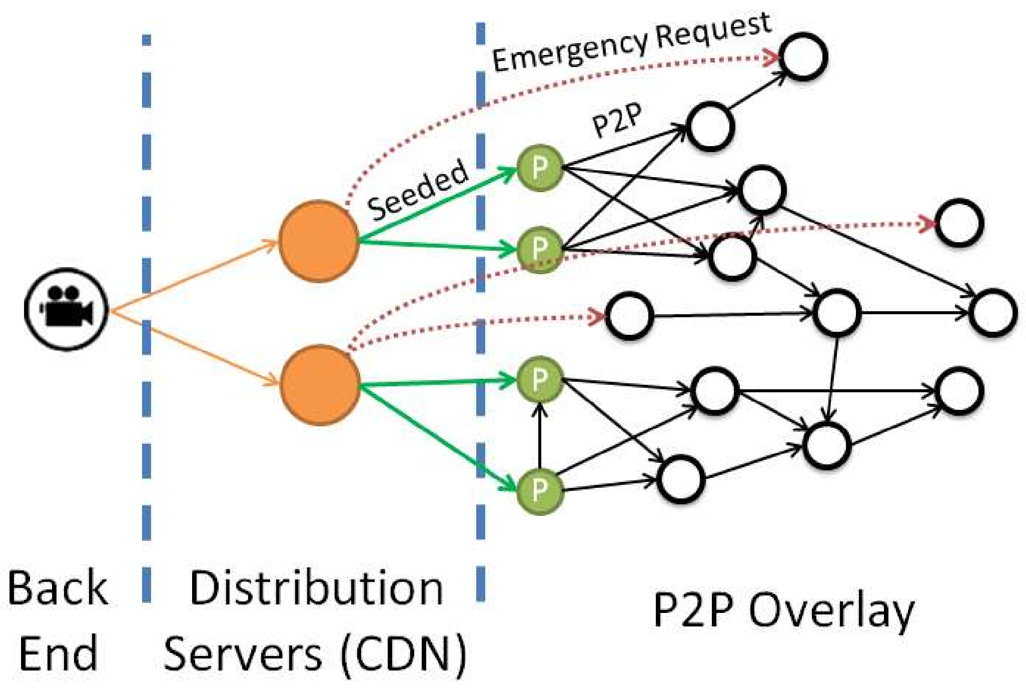


Figure 4.1. CDN-P2P chunk flow overview

Live streaming systems have a back-end that captures, encodes, and splits a video stream into *chunks*. Chunks have fixed size but may contain multiple video frames under variable bit rate encoding. In hybrid CDN-P2P systems, the back-end distributes video chunks to a set of well-provisioned *servers* hosted on CDNs or on cloud providers. CDN clients become *peers*, organizing themselves in a peer-to-peer overlay (Figure 4.1). Servers *seed*, i.e., immediately redistribute, video chunks received from the back-end to a subset of these peers. Peers that are being seeded by servers are tagged as *prefetchers*. Peers collaborate with each other to receive and redistribute video chunks in a P2P overlay. Peers have a local buffer to store video chunks downloaded before their playback, to synchronize playback at a constant delay relative to the video's capture, and to allow P2P distribution.

Servers also answer *emergency requests* that peers issue when the P2P overlay does not deliver a video chunk before its playback deadline. Unless underlying connectivity issues prevent timely data transmission (e.g., congestion), emergency requests ensure all chunks are delivered and guarantee continuous playback. In CDN-P2P streaming, the total cost to the content provider is dominated by the aggregate upload bandwidth across all servers.

New peers enter the CDN-P2P system via a *bootstrap* that maintains membership information for each streaming channel, i.e., which peers are receiving the stream and participating in its P2P overlay. Peers register with the bootstrap and query membership information to discover potential neighbors and join the P2P overlay. Moreover, peers periodically report neighborhood information and performance metrics to the bootstrap. The bootstrap aggregates reports to build snapshots of the P2P overlay.

Bootstrap also stores metadata about the latest chunk available at the servers. Prefetchers receive the latest chunk available as soon as servers have it, however, bootstrap delays the metadata update, pushing prefetchers slightly ahead of others. Peers joining the overlay will be tricked into considering an older chunk as the latest one, thus giving a time advantage for prefetchers to spread chunks through P2P [Roverso et al., 2012]. For example, at time t, the back-end produces chunk c_t, pushes to the CDN servers that immediately let prefetchers request it. Prefetchers announce c_t and its distribution start, prior to peers need. Bootstrap 'latest chunk available' metadata, however, is updated to announce chunk c_{t-1} and peers will believe that they need to look for c_{t-1}. Since c_{t-1} spread started in $t-1$ they might already have it in their buffer. Time interval between $t-1$ and t is used to distribute c_{t-1}.

Each peer p joins the streaming channel in *initialization mode*, where p requests chunks directly to CDN servers. This initialization reduces the time to fill peer p's buffer and ensures data availability. Initialization mode ends when peer p fills its buffer and establishes neighborhood partnerships with a subset of the peers it received from the bootstrap. Each peer p maintains two sets of neighbors. A peer p receives chunks from a set of *input neighbors*, denoted $\mathcal{I}_p$, and sends chunks to a set of *output neighbors*, denoted $\mathcal{O}_p$. We call the number of input and output neighbors, i.e., $|\mathcal{I}_p|$ and $|\mathcal{O}_p|$, a peer's *in-* and *out-degree*, respectively. Table 4.1 summarizes notation.

If a peer p is not receiving all the chunks it needs from its input neighbors (i.e., it is issuing emergency requests) it will remove from $\mathcal{I}_p$ the neighbor with highest number of unanswered chunk requests. Previous research on overlay construction algorithms show that keeping peers with more upload bandwidth closer to the server improves distribution efficiency [Felber and Biersack, 2005; Payberah et al., 2012a; Traverso et al., 2014]. A peer p will try to establish input partnerships with known peers that have equivalent or more upload bandwidth than itself. A peer p limits its out-degree, $|\mathcal{O}_p|$, to the number of streams it can upload (i.e., the ratio of its upload bandwidth, B_p, to the video streaming rate, R) and denies partnership requests if $\mathcal{O}_p$ is full. A peer p periodically refreshes its list of known peers using two peer sampling (2.2.4) methods: by contacting the bootstrap directly, and by exchanging membership information with

Table 4.1. Definitions and notation.

VAR.	DEFINITION
$\mathcal{P}$	Set of all peers.
$\mathcal{I}_p$	Set of input neighbors of peer p.
$\lvert\mathcal{I}_p\rvert$	In-degree of peer p.
$\mathcal{O}_p$	Set of output neighbors of peer p.
$\lvert\mathcal{O}_p\rvert$	Out-degree of peer p.
S	Server's aggregated seeding ratio.
$\mathcal{O}_S$	Set of prefetchers.
$\lvert\mathcal{O}_S\rvert$	Number of prefetchers.
B_p	Upload bandwidth of peer p.
R	Video streaming rate.
U_p	Peer p's upload rate (to neighbors in $\mathcal{O}_p$).

other known peers through gossiping [Roverso et al., 2013].

Peers periodically send buffer maps to output neighbors in $\mathcal{O}_p$ announcing chunks stored in their buffer. Peers generate chunk requests sequentially, i.e., earliest deadline first, and serve requests in order of arrival. When multiple neighbors can provide video chunks, a peer requests a number of chunks proportional to each neighbor's upload bandwidth.

The benefit in P2P overlay distribution obtained from seeding the stream to a peer is proportional to that peers's upload bandwidth [Felber and Biersack, 2005; Payberah et al., 2012a; Traverso et al., 2014]. As a result, servers choose prefetchers as the first among those with higher upload bandwidth and compute the number of prefetchers as a function of the obtained benefit. Servers seed streams to a set of peers denoted $\mathcal{O}_S$. The number of prefetchers, $|\mathcal{O}_S|$, is defined as a function of the *seeding ratio* S, the ratio of prefetchers's aggregate upload bandwidth to all peers's aggregate upload bandwidth. More formally, if $\mathcal{P}$ is the set of all peers, $\mathcal{F}$ is a candidate set of prefetchers, B_p is the upload bandwidth of peer p, then the number of prefetchers is the size of the smallest set of peers whose summed bandwidth is greater than or equal to a fraction (S) of all peer's summed bandwidth, or:

$$|\mathcal{O}_S| = \min\Big\{|\mathcal{F}| \;\Big|\; \mathcal{F} \subseteq \mathcal{P} \wedge \sum_{p\in\mathcal{F}} B_p \geq S \sum_{q\in\mathcal{P}} B_q\Big\}. \tag{4.1}$$

4.3 Simulation Setup

In this section we describe the simulation environment and experiment setup we use to evaluate distribution efficiency in Sections 4.4 and 4.7. We evaluate CDN-P2P streaming costs using a simulator which shares most of its code with SmoothCache, a

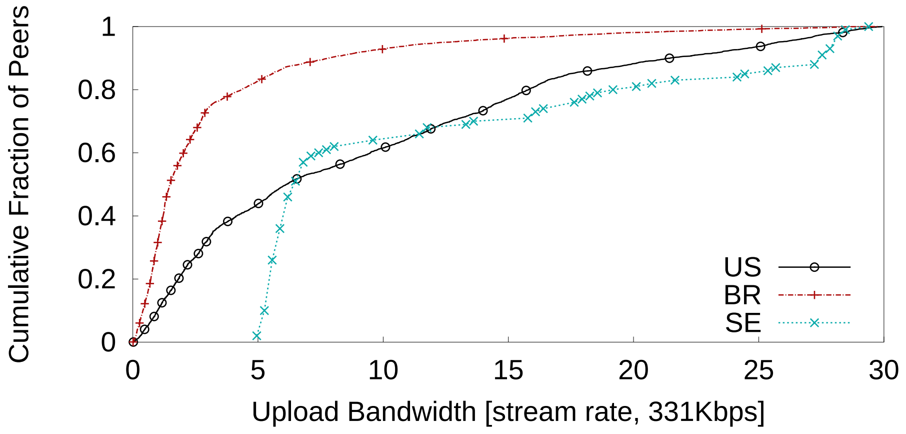


Figure 4.2. Peer upload bandwidth distributions.

real implementation of the CDN-P2P system described in Sec. 4.2 [Roverso et al., 2012, 2015]. The simulator substitutes the system's networking code for a network emulator that allows control of a number of characteristics such as link bandwidths, transmission delays, queueing, and NATs. We repeat each simulation five times and report averages. We simulate for 20 or 60 minutes depending on the scenario. Numerical results ignore data from the first three minutes of all simulations to avoid the overlay's warm-up period.

Underlay configuration

We configure the network emulator without any underlay bandwidth constraints (but we do limit peer upload bandwidth below) and set the transmission error rate to zero. We choose the one-way latency between peers uniformly distributed between 10ms and 50ms. Although the underlay never drops video chunks, we note the P2P overlay may still fail to disseminate a video chunk due to lack of peer upload bandwidth and inefficient neighborhoods leading to long, high-latency dissemination paths.

Peer upload bandwidth distributions

We evaluate CDN-P2P streaming costs using the three different peer upload bandwidth distributions shown in Figure 4.2. Upload bandwidth distribution for North American ('US' line) and Brazilian ('BR' line) hosts were collected from TestMy.net, a website where users can test their Internet speed. TestMy.net offers reports of the last 10000 upload and download bandwidth measurements performed on its servers from each country. We filtered upload bandwidths greater than 10Mbps to avoid distortions caused by potentially non-domestic connections. The last distribution ('SE' line) was collected from upload bandwidths observed in an operational test deployment with thousands of peers over a Swedish corporate network [Roverso et al., 2012]. Unless stated otherwise, we use the bandwidth distribution of North American hosts as a baseline and refer to it as BASE (most results for other bandwidth distributions can be found at Appendix B).

Restrictions to bandwidth distributions

From each distribution, we generate four resource-constrained scenarios. We consider two scenarios where we divide peer bandwidths by two and four, denoted DIV2 and DIV4, respectively. This is equivalent to multiplying the streaming rate by two and four, e.g., as needed for streaming HQ videos. We also consider scenarios where 50% and 75% of peers are free-riders, i.e, do not upload video chunks, denoted 50F and 75F, respectively. To better illustrate the difference between these, one can imagine a hypothetical distribution in which all peers have the same bandwidth x. In DIV2 distribution for the hypothetical distribution, all peers have the same bandwidth equals to $x/2$. While in 50F distribution, half of the peers have bandwidth equal to zero and the other half equal to x. These distributions make the system perform differently despite having the same average bandwidth. The same idea applies to DIV4 and 75F.

Overlay sizes

We evaluate channel populations varying from 100 to 2000 simultaneous peers. Peers join the overlay at a constant rate over the first 100 seconds of the simulation. Unless stated otherwise, we run simulations with 500 peers.

Overlay configuration

We assume servers have bandwidth to upload the number of streams required to achieve the configured seeding ratio S, to support peers in initialization mode, and answer all emergency requests. We set the seeding ratio $S = 2.5\%$. This seeding ratio has been observed to yield positive results in previous work [Roverso et al., 2012] and is small enough to allow cost savings. We vary peer in-degrees $|\mathcal{I}_p|$ between 2 and 5, and minimum peer out-degrees $|\mathcal{O}_p|_{min}$ between 0 and 10. Unless stated otherwise, we set $|\mathcal{I}_p| = 3$, $|\mathcal{O}_p|_{min} = 0$ and $|\mathcal{O}_p|_{max} = \min(\lfloor B_p/R \rfloor, 10)$, where B_p is peer p's upload bandwidth and R is the streaming rate. The out-degree is bounded by the number of streams a peer can upload to avoid receiving more requests than can be served and by a fixed threshold to limit abuse of peer upload capacity.

Peer behavior

To further stress the P2P overlay with dynamic peer behaviors, we emulate both peer churn and flash crowd scenarios. For the peer churn scenario, we have considered a severe case where we remove and insert 3% of peers in the overlay at random every ten seconds; severe because this churn rate is at least three times higher than previously found in other studies [Simoni et al., 2014; Payberah et al., 2012b]. For the flash crowd scenario, we have considered an overlay initially with 100 peers alternating between adding and removing 1000 peers every ten minutes.

4.4 P2P Distribution Efficiency

Our goal is to evaluate P2P distribution efficiency for different network configurations. We show that distribution efficiency can degrade significantly for bandwidth-constrained scenarios even though peers have spare upload bandwidth.

Video chunks are either distributed by the server via seeding, peer initialization, and emergency requests, or disseminated by the P2P overlay. We note that peers that do not receive a video chunk from the P2P overlay issue an emergency request for that chunk. There are no "missed" chunks; servers cover all costs and distribute all chunks that the P2P overlay did not.

We measure P2P distribution efficiency as the fraction of chunks that servers did not have to distribute due to the P2P overlay, which we refer to as *savings*. More precisely, savings is the ratio between the number of chunks distributed by the P2P overlay and the total number of distributed chunks.

Figure 4.3 shows savings over time for four different peer upload bandwidth scenarios. Results for DIV2 (omitted) are quantitatively similar to results for 50F. Savings for BASE and 50F stabilize at 96% and 95% approximately three minutes after the start of the experiment. In these scenarios, server workload (seeding, peer initialization, and emergency requests) accounts for 4–5% of all distributed chunks. Savings increases sharply in the beginning as peers optimize the overlay self-organizing away from servers by decreasing upload bandwidth. Figure 4.3 shows that savings for the 75F and DIV4 scenarios stabilize substantially lower at 77% and 50%, respectively. This means that servers are responsible for distributing 23% and 50% of all chunks, workloads that are 5 and 10 times higher than in the BASE scenario. One may think

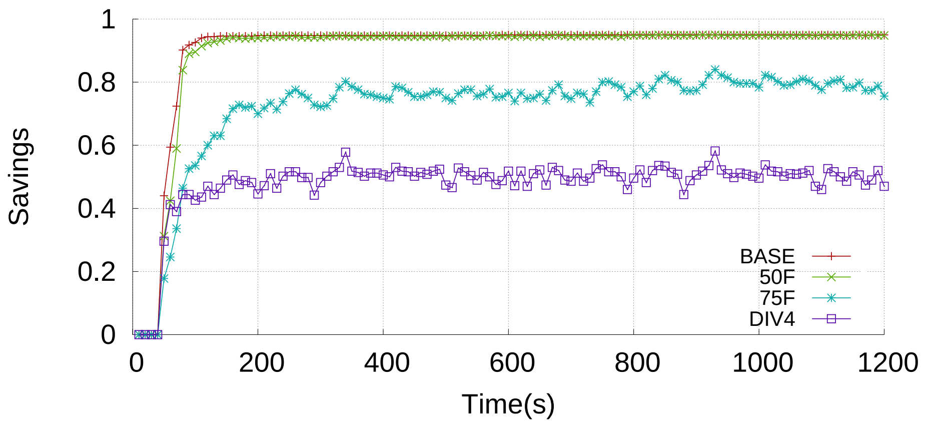


Figure 4.3. Savings for different upload bandwidth distributions.

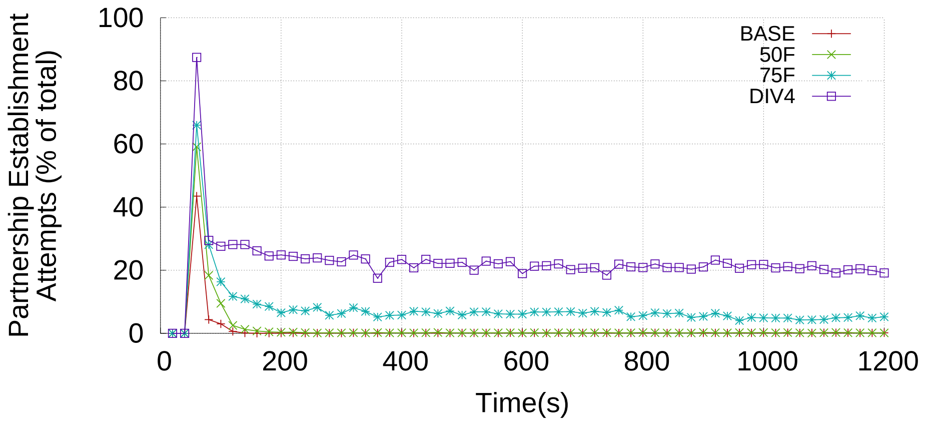


Figure 4.4. Partnerships establishment attempts over time for different upload bandwidth distributions.

that savings is low because peer upload bandwidth is fully utilized and insufficient to distribute chunks, but we find this is not the case. We identify several reasons for inefficient P2P distribution.

Peers cannot establish input partnerships. Recall from Section 4.2 that a peer p limits its out-degree, $|\mathcal{O}_p|$, to the number of streams it can upload, and that the system employs a bandwidth-aware overlay construction policy. Low upload bandwidths lead to low out-degrees. Low amount of out-degrees on the overlay lead to competition for output partnerships. Moreover, as each peer tries to establish input partnerships with peers that have equivalent or more upload bandwidth than itself, high-bandwidth peers have less input peer options. For example, a free-rider can establish input partnerships with any peer but a high-bandwidth peer can establish input partnerships with a few peers only.

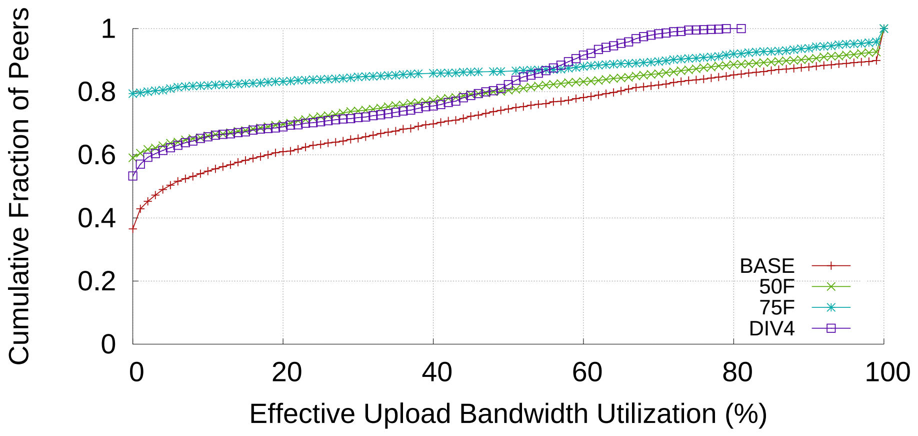


Figure 4.5. Distribution of peer upload bandwidth utilization.

Figure 4.4 shows ratio between the number of partnership establishment attempts and the number of possible partnerships, i.e., $\min(\sum_p |\mathcal{I}_p|, \sum_p |\mathcal{O}_p|)$, over time. The spike at the beginning is due to new peers establishing their initial partnerships and changing partnerships to optimize the P2P overlay. We observe that BASE and 50F build an efficient overlay and partnership establishment attempts stabilize close to zero. Under more severe bandwidth constraints such as in 75F and DIV4, peers keep attempting to establish new input partnerships, either because their previous attempts failed or because they are not obtaining sufficient data from the overlay.

The issue could be eased by topology adjustments (e.g., different $|\mathcal{I}_p|$ and $|\mathcal{O}_p|$ limits, or other overlay construction policies). However, these configurations are scenario dependant, and more constrained scenarios need, proportionally, more fine-tuned configurations.

Peers underutilize upload bandwidth. Figure 4.5 shows the distribution of effective upload bandwidth utilization over time for different upload bandwidth distributions. Effective upload bandwidth utilization is the rate at which a peer p uploaded chunks to its output neighbors, U_p, divided by the maximum rate at which that peer could upload chunks, $\min(B_p, |\mathcal{O}_p|R)$. From the data, the fraction of peers that are free-riders or have negligible bandwidth and thus zero output neighbors (i.e., $|\mathcal{O}_p| = 0$) is 10%, 55%, 77%, and 39% in BASE, 50F, 75F, and DIV4, respectively. These values are visually correlated to the y-values where effective upload bandwidth equals zero. Peers have average utilizations of 17% and 32% in the BASE and 50F scenarios, respectively, which is enough to redistribute most chunks and achieve high savings (Figure 4.3). Peers in the 75F and DIV4 scenarios also have spare upload bandwidth capacity; savings, however, remains low.

Figures 4.4 and 4.5 show that peers have spare upload bandwidth but cannot

Table 4.2. Origin of video chunks received.

CHUNK ORIGIN	BASE	50F	75F	DIV4
Seeded by servers	1.4%	0.8%	0.3%	0.7%
Emergency request	3.0%	4.9%	25.9%	49.5%
P2P overlay	95.6%	94.3%	73.8%	49.8%

Table 4.3. Average number of traversed hops for a video chunk that enters the P2P overlay clustered by origin.

CHUNK ORIGIN	BASE	50F	75F	DIV4
Seeded by servers	3.65	4.07	4.59	1.84
Emergency request	1.01	1.30	1.47	1.43

establish input partnerships, relying on emergency requests. Manual inspection of logs from peers that issue many emergency requests shows that (i) these peers may only know peers with less upload bandwidth than themselves; or that (ii) they may know peers with more upload bandwidth but fail to establish input partnerships (i.e., the peers they know cannot establish additional output partnerships); or that (iii) they may have input neighbors that seldom transmit video chunks. When case (iii) happens, it is usually the case that upstream peers in the partnership chain are in cases (i) or (ii), also issuing many emergency requests.

Emergency requests are not P2P-friendly. Chunks obtained through emergency requests have short lifespans in the P2P overlay and cannot be widely distributed. Conversely, seeded requests are transmitted early, stay longer in the P2P overlay, and have more opportunities to be distributed. Table 4.3 shows, for different upload bandwidth distributions, the average number of traversed hops for chunks originated as seeded by CDN servers or as answers to emergency requests. We note that chunks obtained from seeded requests traverse more hops than chunks from emergency requests - a direct consequence of previous claims. Table 4.2 shows, for different upload bandwidth distributions, the average fraction of chunks seeded by servers, transmitted as answers to emergency requests, and redistributed by the P2P overlay. We observe that in bandwidth-constrained scenarios low savings result from significantly higher fractions of emergency requests than BASE. Peers that issue emergency requests may fail to redistribute chunks, leading to their output neighbors also issuing emergency requests, degrading overall P2P distribution efficiency. This gets more evident as resources get more constrained.

Summary. These findings show that static configuration of the number of prefetchers or seeding ratios may perform poorly depending on P2P overlay characteristics. System performance, however, may be improved with specialized or more elaborate overlay maintenance mechanisms [Simoni et al., 2014; Payberah et al., 2012a; Fortuna et al., 2010; Felber and Biersack, 2005; Wichtlhuber et al., 2014; Tang et al., 2009; Traverso et al., 2014]. We next discuss those possible improvements and then propose a practical and complementary approach that adapts to variable overlay conditions.

4.5 Current Efficiency Improvements

A large body of work has been dedicated to building and evaluating algorithms and mechanisms for efficient P2P live streaming. Contributions cover techniques to optimize P2P overlay topologies [Simoni et al., 2014; Payberah et al., 2012a; Fortuna et al., 2010; Felber and Biersack, 2005], schedule chunk requests and transmissions [Fortuna et al., 2010; Carlsson and Eager, 2007; Zhao et al., 2009], adapt topologies according to overlay and network conditions [Wichtlhuber et al., 2014], increase peer cooperation and avoid free-riding [Guerraoui et al., 2010; Piatek et al., 2010; Gonçalves et al., 2014], mitigate collusion attacks [Piatek et al., 2010], handle peer churn and flash crowds [Kumar et al., 2007; Liu et al., 2012]. We note our CDN-P2P streaming system incorporates findings from previous work, e.g., it organizes peers away from servers by decreasing upload bandwidth [Felber and Biersack, 2005; Payberah et al., 2012a; Traverso et al., 2014]. These works improve a P2P overlay and may be applicable to CDN-P2P systems. However, they do not consider emergency requests, a central point in our proposed design (Section 4.6).

Regarding hybrid CDN-P2P live streaming, researchers have studied how to use system parameters and performance metrics—such as user churn rate, stream rate, average user upload bandwidth, maximum tolerable chunk distribution delay—to build resource allocation models. These allocation models can be used, e.g., to provision upload bandwidth at streaming servers [Tewari and Menon, 2009], deciding where to connect new peers joining the stream (e.g., CDN servers or the P2P overlay) [Yin et al., 2009], or to switch between CDN and P2P operation [Mansy and Ammar, 2011]. Another similar resource allocation solution comes from CLive [Payberah et al., 2012b], a hybrid cloud-P2P solution which dynamically adds/removes cloud-hosted servers as seeding supernodes in the P2P overlay. CLive considers overlay size, estimated peer upload bandwidth distribution, estimated chunk diffusion trees, and maximum acceptable chunk distribution delay to adjust cloud servers demand. These solutions, again, do not consider emergency requests. Yet they share a similar concept with

Table 4.4. Definitions and notation used by AERO.

VAR.	DEFINITION
R	Video streaming rate.
S	Server's aggregated seeding ratio.
$\|\mathcal{O}_S\|_t$	Number of prefetchers at round t.
C_t	Server upload bandwidth consumption at round t.
E_t	Seeding ratio error at round t ($C_t - R\|\mathcal{O}_S\|_{t-1}$).
δ	AERO's seeding ratio scaling factor.
Δ	AERO's seeding ratio scaling factor upper bound.

our proposed design - as adapting the number of prefetchers is a form of resource allocation. Our design does not require detailed information about system parameters, performance metrics or network conditions. Compared to other solutions, ours requires less monitoring overhead and can be more easily deployed as it makes decisions based only on server bandwidth consumption, which is trivial to obtain. It is a simpler alternative that can efficiently allocate resources on demand in diverse scenarios.

4.6 Adaptive Emergency Request Optimization

We propose AERO, the Adaptive Emergency Request Optimization mechanism. AERO's goal is to minimize servers bandwidth consumption by increasing P2P distribution efficiency. Our main idea is to configure the seeding ratio S dynamically – and, consequently, the number of prefetchers $|\mathcal{O}_S|$. AERO increases the seeding ratio when P2P distribution efficiency is low, seeding chunks to peers that can then use their upload bandwidth more effectively and possibly start new chunk redistribution chains in the P2P overlay. This improvement over P2P distribution efficiency and its costs are beneficial as they reduce emergency requests. AERO decreases the seeding ratio when P2P distribution efficiency stabilizes in an attempt to maximize savings. AERO tracks a practical seeding ratio as network conditions and P2P overlays change. Table 4.4 summarizes notation used by AERO.

AERO operates in rounds. Each round, AERO decides whether to increase or decrease the seeding ratio S depending on historical seeding ratio allocations and bandwidth consumption observations, as shown in Alg. 1.

AERO keeps a history of bandwidth consumption at CDN servers, C_t, to compute whether bandwidth consumption is stable or not. AERO considers that bandwidth consumption is increasing or decreasing if current bandwidth consumption is higher or lower than that observed in the last three rounds. If the current bandwidth consumption is similar to those observed in previous rounds or if a trend is unclear,

Algorithm 1: AERO's algorithm to update the seeding ratio S at each round

```
input: history of bandwidth consumption, seeding ratio
input: seeding ratio scaling factor $\delta$ (upper bound $\Delta$)
input: seeding ratio error $E$ at rounds $t$ and $t-1$
if bandwidth consumption is stable then $S \leftarrow 0.95S$;
else
    if bandwidth consumption jumps then $\delta \leftarrow \Delta$;
    else
        if $\mathrm{sign}(E_t) = \mathrm{sign}(E_{t-1})$ then $\delta \leftarrow \min(\delta/0.75, \Delta)$;
        else $\delta \leftarrow \delta \times 0.75$;
    end
    $S \leftarrow S + \mathrm{sign}(E_t)\delta$
end
```

AERO considers that bandwidth consumption is stable. More precisely, the trend of bandwidth consumption is given by the following, where $I(\cdot)$ in an indicator function that returns zero if trend is stable and 1 otherwise.

$$\text{trend} = \begin{cases} \text{increasing} & \text{if } \sum_{i=1}^{3} I(0.95C_t > C_{t-i}) \geq 2, \\ \text{decreasing} & \text{if } \sum_{i=1}^{3} I(1.05C_t < C_{t-i}) \geq 2, \\ \text{stable} & \text{otherwise.} \end{cases} \tag{4.2}$$

AERO configures the seeding ratio S as a function of CDN servers upload bandwidth consumption due to seeding and emergency requests. We define the seeding ratio error at round t, E_t, as the difference between the total servers upload bandwidth consumption, C_t, and the expected bandwidth consumption, $R|\mathcal{O}_S|_{t-1}$ (streaming rate multiplied by the number of prefetchers in the last round). The seeding ratio error is positive when the seeding ratio is low and servers receive a large number of emergency requests (underprovisioning). Conversely, the seeding ratio error is negative when the seeding ratio is high and prefetchers are exchanging chunks between themselves (overprovision). AERO does not consider the upload bandwidth used to support peers in the initialization mode because this bandwidth demand is fixed per peer and because peer arrival times are outside the control of servers.

AERO gradually modifies the seeding ratio S using a variable defined as scaling factor, δ. The scaling factor works as a fine tuner determining how much S should be modified in each round. Its value can decrease or increase as S gets, respectively, near or far from an optimum seeding ratio. We set a maximum seeding ratio modifier, Δ, to limit δ.

Whenever bandwidth consumption is not stable, AERO updates the seeding ratio S according to the seeding ratio error E_t and E_{t-1} of the last two rounds. If $\text{sign}(E_t) = \text{sign}(E_{t-1})$, the last two rounds overestimated or underestimated the optimal seeding ratio. AERO concludes it is not close to the optimum seeding ratio and increases the scaling factor by a constant factor (shown as 0.75 in Alg. 1) to improve convergence speed. If $\text{sign}(E_t) \neq \text{sign}(E_{t-1})$, then one round underestimated and the other overestimated the optimal seeding ratio. AERO decreases the scaling factor by a constant factor (shown as 0.75 in Alg. 1) to converge closer to the optimal seeding ratio. The scaling factor is restored to the upper bound whenever a jump in bandwidth consumption is detected, i.e., whenever bandwidth consumption doubles or halves between rounds. At each round, AERO increases the seeding ratio by the scaling factor when the system is underprovisioned ($E_t > 0$) and decreases the seeding ratio when the system is overprovisioned ($E_t < 0$).

The process above iterates until the scaling factor decreases enough that servers bandwidth consumption stabilizes. To avoid getting stuck at a local minimum, AERO systematically decreases the seeding ratio by a constant factor once bandwidth consumption stabilizes (shown as 0.95 in Alg. 1). This systematic reduction of the seeding ratio will either reduce servers bandwidth consumption or cause fluctuations due to underprovisioning. When bandwidth fluctuates, AERO will reconverge to an optimum seeding ratio (possibly the same).

Stability periods may end due to varying network and P2P overlay conditions, e.g., peer churn or flash crowds. AERO's round duration must be long enough to allow AERO to observe the impact of seeding ratio adaptation decisions on bandwidth consumption. This property is necessary to compute the seeding ratio errors E_t, and may negatively impact convergence times on systems with very large buffers. AERO makes no assumptions about overlay properties or peer behavior, and can be deployed in conjunction with existing overlay maintenance or chunk scheduling mechanisms.

4.7 Evaluation

In this section we evaluate AERO for different deployment scenarios varying peer upload bandwidth distributions, overlay construction policies, overlay sizes, and peer behavior. As in Section 4.4, we consider four out of five different BASE peer upload bandwidth distribution derivatives: BASE, 50F, 75F, and DIV4. We show that AERO increases P2P savings across all scenarios, trading bandwidth consumption spent on emergency requests for seeding peers, improving P2P distribution efficiency.

Unless stated otherwise, we use the default parameters defined in Section 4.3

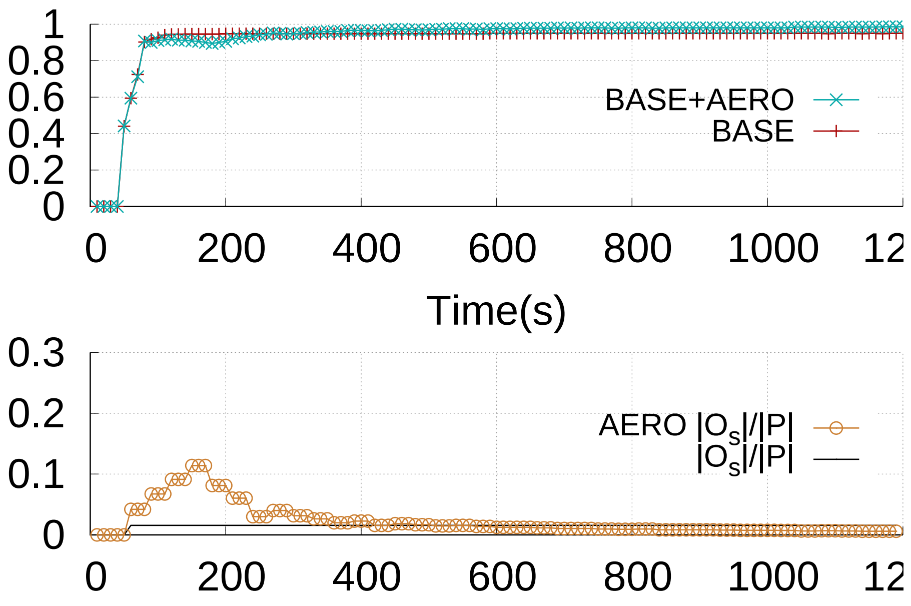


Figure 4.6. Comparison of overall savings for BASE scenario with and without AERO.

and configure AERO with 30-second rounds, initial seeding ratio $S = 2.5\%$, maximum seeding ratio modifier $\Delta = 5\%$. This initial seeding ratio has been observed to yield positive results in previous work [Roverso et al., 2012].

4.7.1 Peer upload bandwidth

Figures 4.6, 4.7, 4.8, and 4.9 compare server bandwidth savings for different peer upload bandwidth distributions with and without AERO on streaming channels with 500 peers (left y-axes). The lines labeled BASE, 50F, 75F, and DIV4 in each figure were taken from Figure 4.3. Figures 4.6 and 4.7 show that AERO adapts to high-resource scenarios and even increases savings by a little.

Savings shows P2P distribution efficiency and it provides a good comparison for gains between hybrid CDN-P2P and CDN-only models. Since AERO brings an optimization over the current CDN-P2P model, we also measure server consumption relative difference between using and not using AERO. For example, on the BASE scenario, although absolute difference is small, AERO reduces *average server upload bandwidth consumption* from 5.1% to 2.9%, an 43% economy to servers.

Figures 4.8 and 4.9 show that AERO achieves even higher savings in resource-constrained scenarios (and up to 60% reduction of average server upload bandwidth consumption). We note that there are no “missed” chunks; both solutions (with and

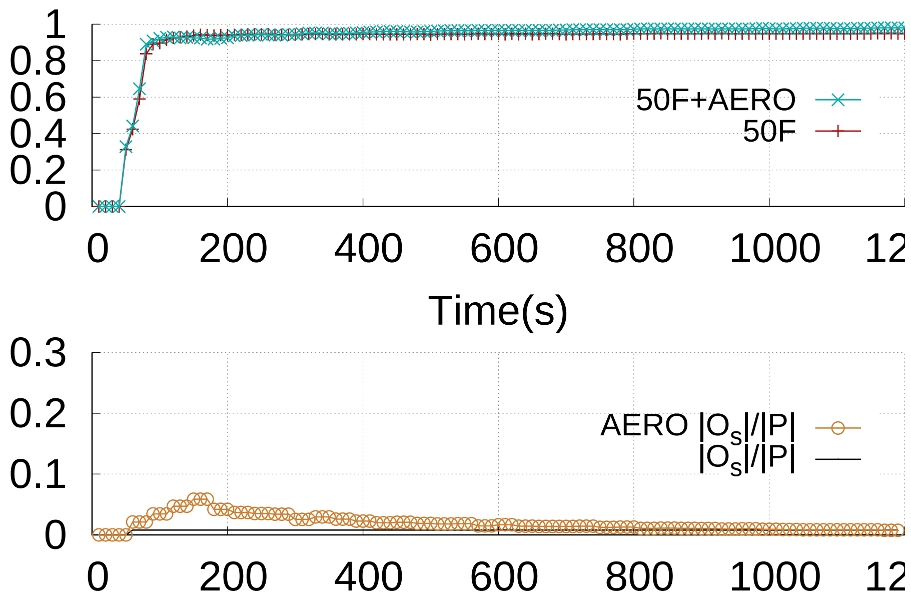


Figure 4.7. Comparison of overall savings for 50F scenario with and without AERO.

without AERO) distribute the same number of chunks. AERO's savings come entirely from higher P2P distribution efficiency.

To explain this behavior, Figures 4.6, 4.7, 4.8, and 4.9 also show the fraction of prefetchers configured by AERO and the fraction of prefetchers statically configured when not using AERO (right y-axes, with range up to 30%). In high-resource scenarios (Figures 4.6 and 4.7), AERO reduces the fraction of prefetchers while maintaining P2P distribution efficiency. AERO achieves higher bandwidth savings when its fraction of prefetchers gets lower than in the static configuration. In resource-constrained scenarios (Figures 4.8 and 4.9), we observe AERO increases the number of prefetchers compared to the static configuration. This allows peers to distribute more chunks through the P2P overlay and helps further reduce the number of emergency requests. Finally, we note that AERO seeds to a lower fraction of peers in scenario 75F compared to DIV4. On average, high-resource peers in 75F have four times more bandwidth than high-resource peers in DIV4, and less seeding is necessary to achieve efficient P2P distribution.

Table 4.5 shows the fraction of chunks that were distributed via seeding, emergency requests, and the P2P overlay, throughout scenarios using AERO. We have copied the data from Table 4.2 to aid comparison. It shows that AERO significantly improves P2P distribution efficiency. We note, however, that the amount of emergency requests on constrained scenarios is still high. AERO aims at reducing emergency

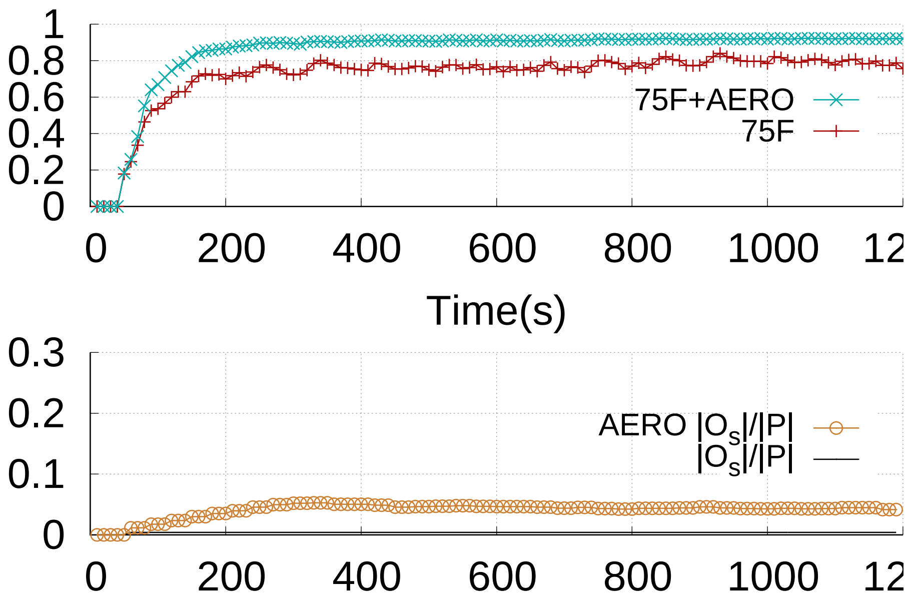


Figure 4.8. Comparison of overall savings for 75F scenario with and without AERO.

Table 4.5. Origin of received video chunks when using AERO (data copied from Table 4.2 below to aid comparison).

CHUNK ORIGIN	BASE AERO	50F AERO	75F AERO	DIV4 AERO
Seeded by servers	1.6%	1.6%	3.2%	6.6%
Emergency request	0.9%	1.6%	5.9%	13.4%
P2P overlay	97.5%	96.8%	90.9%	80.0%
	BASE	50F	75F	DIV4
Seeded by servers	1.4%	0.8%	0.3%	0.7%
Emergency request	3.0%	4.9%	25.9%	49.5%
P2P overlay	95.6%	94.3%	73.8%	49.8%

requests to virtually none, but currently does it by analysing observed and expected server bandwidth consumption. When the seeding ratio increases, the probability that prefetchers exchange chunks between themselves increases as well. Prefetchers trading chunks between themselves do less server requests than expected thus leaving space for AERO to stabilize with emergency requests consuming the bandwidth expected to be used by prefetchers.

Figure 4.10 shows savings over time when peer upload bandwidths are sampled from the BR and SE distributions (Fig. 4.2). Results for the SE upload bandwidth distribution are qualitatively similar to the high-bandwidth scenarios shown in Figures

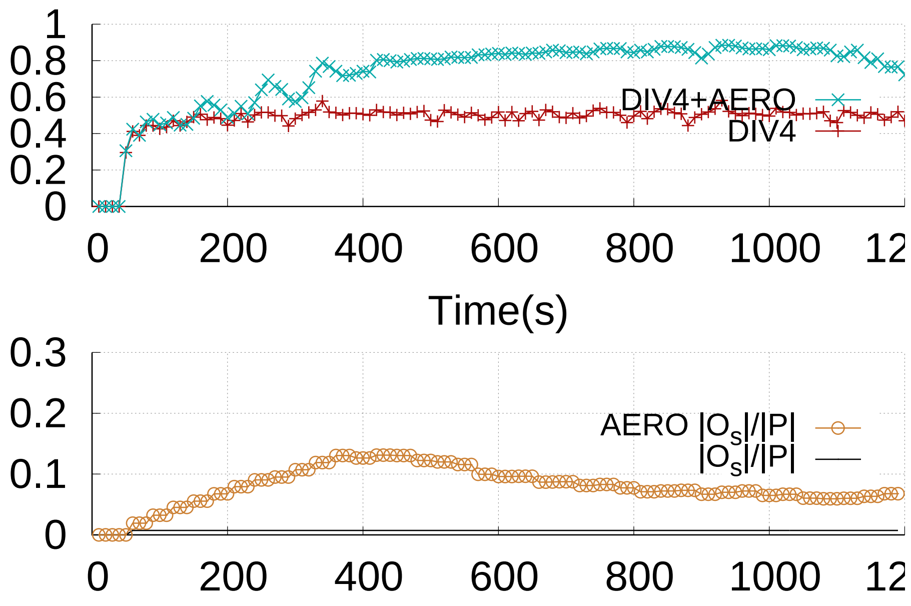


Figure 4.9. Comparison of overall savings for DIV4 scenario with and without AERO. $|\mathcal{O}_S|/|\mathcal{P}|$ removed from key for clarity.

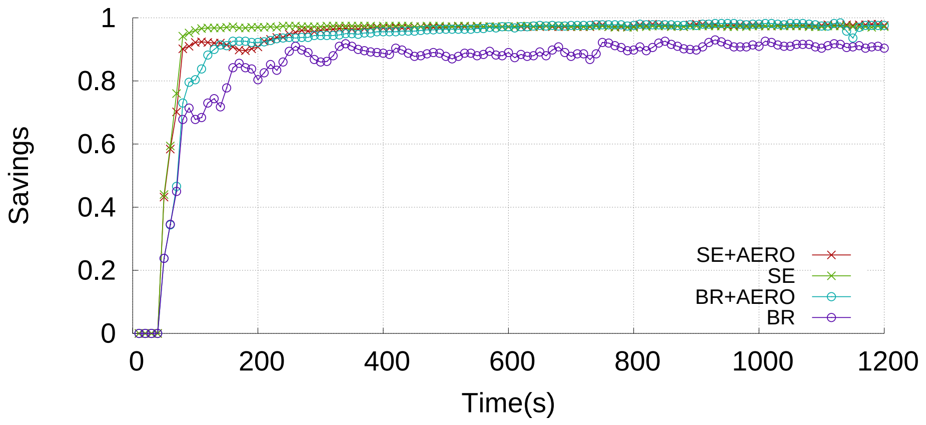


Figure 4.10. Savings for BR and SE peer upload bandwidth distributions.

4.6 and 4.7. Brazilian hosts have lower upload bandwidths (BR distribution), which results in a more resource-constrained scenario. In this scenario AERO has been able to achieve very high savings and reduce average server upload bandwidth consumption by 68%.

These two bandwidth distributions has been investigated in the same way as North American one has and the results can be found on Appendix B. We point out that the brazilian distribution is already way constrained; all restrictions (50F, DIV2, 75F, DIV4) made peers average upload bandwidth drop below one. This means

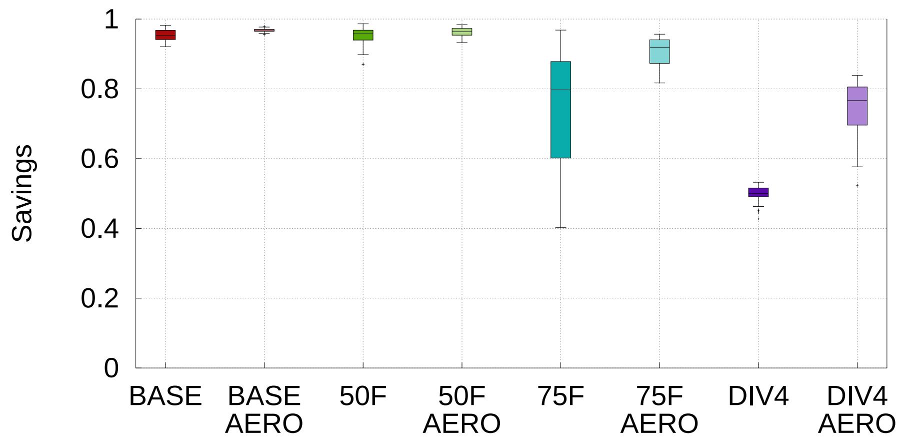


Figure 4.11. Savings of different overlay construction configurations grouped by peer upload bandwidth distribution.

Table 4.6. Savings for different overlay construction limits

SCENARIO		I_2O_0	I_3O_0	I_4O_0	I_5O_0	I_5O_5	I_5O_{10}	$\overline{Savings} \pm \sigma$
	BASE	0.978	**0.949**	0.968	0.940	0.948	0.942	0.954 ± 0.015
	50F	0.980	**0.946**	0.964	0.908	0.958	0.952	0.951 ± 0.024
	75F	0.821	**0.773**	0.741	0.575	0.752	0.829	0.748 ± 0.092
	DIV4	0.493	**0.504**	0.515	0.491	0.522	0.447	0.496 ± 0.027
A	BASE	0.973	**0.971**	0.968	0.965	0.967	0.969	0.969 ± 0.003
E	50F	0.980	**0.965**	0.970	0.965	0.945	0.951	0.963 ± 0.013
R	75F	0.931	**0.911**	0.927	0.929	0.875	0.867	0.907 ± 0.029
O	DIV4	0.797	**0.696**	0.694	0.726	0.819	0.738	0.745 ± 0.052

that peers do not have enough capacity to fully distribute the stream to every overlay participant. In harsh scenarios such as BR-75F there is little that AERO can do to aid the system, yet it does not harm it either.

4.7.2 Overlay properties and overlay size

P2P distribution efficiency is highly dependent on overlay maintenance mechanisms and overlay properties. We choose six configurations varying peer in-degrees $|\mathcal{I}_p| \in \{2,3,4,5\}$ and minimum out-degrees $|\mathcal{O}_p| \in \{0,5,10\}$ to evaluate AERO under different overlay construction constraints. Figure 4.11 shows savings for different configurations grouped by peer upload bandwidth distribution. Each bar shows the minimum, quartiles, and maximum upload bandwidth savings from 30 experiments (5 for each in- and out-degree configuration). Some peer upload bandwidth scenarios are very dependent on overlay configurations (e.g., 75F), illustrating that P2P efficiency depends

Table 4.7. Savings for different neighbor selection policies.

SCENARIO	RANDOM	BW-AWARE	BW-RELAX
75F	64.0%	77.3%	85.9%
DIV4	52.8%	50.4%	50.3%
75F AERO	88.9%	91.1%	92.7%
DIV4 AERO	78.3%	79.6%	72.8%

Table 4.8. Savings for different overlay sizes.

	OVERLAY SIZE (peers)			
SCENARIO	100	500	1000	2000
BASE	96.4%	94.9%	97.2%	97.2%
50F	96.9%	94.6%	96.9%	97.2%
75F	91.3%	77.3%	78.0%	75.9%
DIV4	50.3%	50.4%	50.6%	48.6%
BASE AERO	96.3%	97.1%	97.2%	97.4%
50F AERO	96.3%	96.5%	98.1%	97.8%
75F AERO	92.5%	91.1%	93.5%	92.9%
DIV4 AERO	64.1%	79.6%	74.6%	78.1%

on overlay properties. AERO improves savings for all peer upload bandwidth distributions. Table 4.6 shows the average savings for each different overlay construction limit. We have highlighted the default configuration. For any fixed in- and out-degree configuration, average savings when using AERO is always higher or equivalent (i.e., lower within 1%) than average savings when not using AERO (static seeding ratio). We draw attention to the most constrained scenarios, the worse average savings with AERO is better than the best without it, e.g., worse 75F AERO=0.867, best 75F=0.829 and worse DIV4 AERO=0.694, best DIV4=0.522. These results indicate that AERO could be used as an alternative or complement to P2P overlay maintenance mechanisms (e.g., [Simoni et al., 2014; Payberah et al., 2012a; Fortuna et al., 2010; Felber and Biersack, 2005; Wichtlhuber et al., 2014; Tang et al., 2009; Traverso et al., 2014]).

We have also evaluated savings under three different neighbor selection policies where each peer tries to (i) establish partnerships with random known peers ('random'), (ii) establish partnerships with known peers that have equivalent or higher upload bandwidth (the default, 'BW-aware'), and (iii) establish partnerships with known peers that have at least 80% of its own upload bandwidth ('BW-relax'). Table 4.7 shows average upload bandwidth savings for bandwidth-constrained scenarios. We observe that AERO not only improves savings, but that adding AERO results in higher impact than changing the neighbor selection policy. Results for high-bandwidth scenarios show quantitatively similar savings across all neighbor selection policies (omitted).

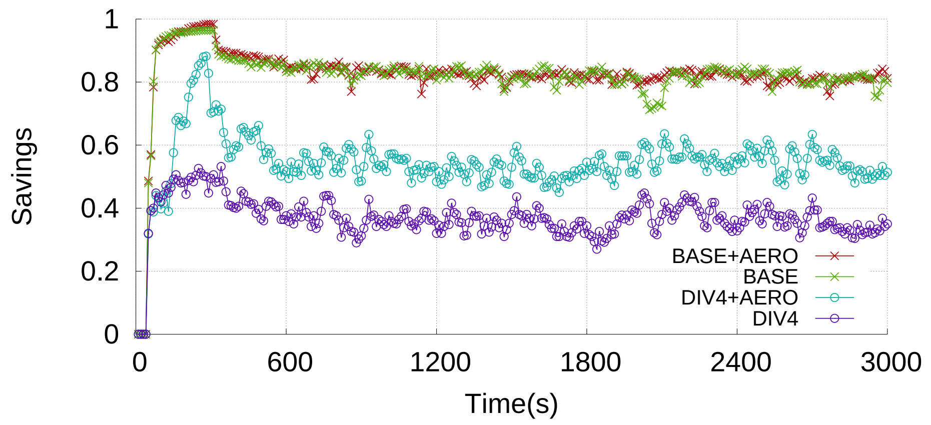


Figure 4.12. Savings under peer churn.

Table 4.8 shows average savings for different overlay sizes and bandwidth-constrained scenarios. We find that varying the overlay size does not significantly impact savings (lines have similar values), except in a few cases of small overlays with highly variable performance (100 peers, 75F and DIV4). AERO provides equivalent or higher savings (up to 30%) and substantially reduces server upload bandwidth consumption (up to 70%) across all overlay sizes.

4.7.3 Peer churn and flash crowds

We also evaluate AERO under adverse peer behavior. Figure 4.12 shows savings for the BASE and DIV4 scenarios when we turn on peer churn at second 300. Every 10 seconds 3% of peers are replaced (Sec 4.3). Each new peer spend approximately 60 seconds requesting chunks to CDN servers at initialization. Thus the configured churn rate and peer initialization time *implies* a reduction of 18% in savings. Moreover, churn replaces peers that have complete neighborhoods for new peers that have empty buffers and zero partnerships, disrupting the P2P overlay and further degrading savings. Figure 4.12 shows that AERO's dynamic seeding ratio configuration adapts to very dynamic overlays and provides equivalent or higher bandwidth savings compared to static seeding ratios (results for 75F are omitted but qualitatively similar to DIV4).

Churn can be especially harmful if it removes a significant fraction of prefetchers from the overlay at once. During our experiments we have verified the occurrence of one such case in one simulation run for the BASE scenario at around 2000 seconds. In Figure 4.12, the disruption on this specific run can be seen even after averaging all five simulation runs.

Figure 4.13 shows savings over time for the BASE and DIV4 scenarios when the

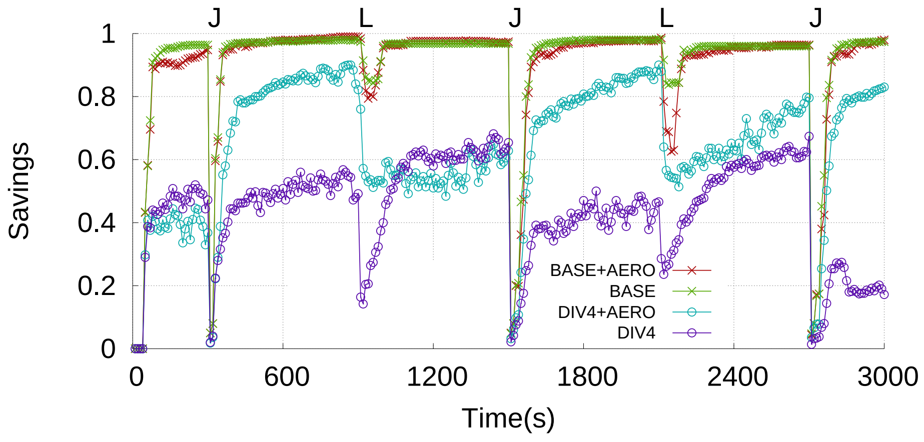


Figure 4.13. Savings over time for periodic flash crowd events.

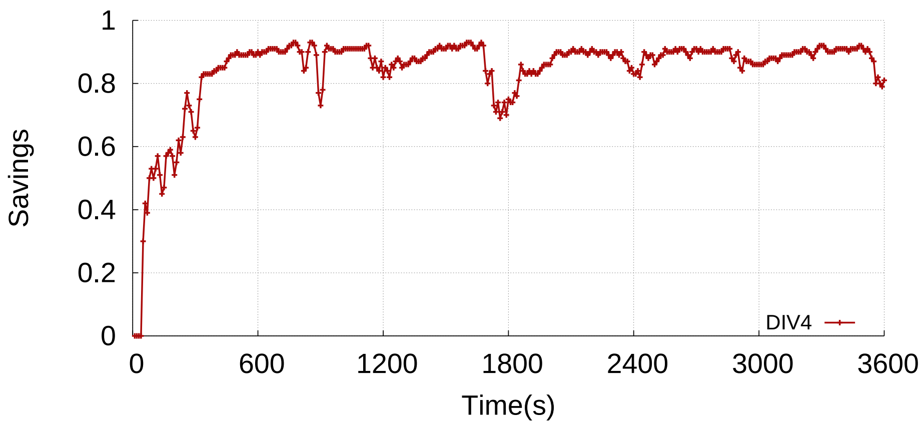


Figure 4.14. Savings for DIV4 scenario with AERO over longer experiments.

streaming channel experiences periodic flash crowd events. We start with an overlay containing 100 peers, and alternate between adding and removing 1000 peers from the overlay periodically every 10 minutes. Peers join the overlay at seconds 300, 1500, and 2700, denoted ‘J’, in initialization mode and significantly degrade savings, as shown by the simultaneous valleys. AERO then quickly adjusts the seeding ratio and achieves high savings. Peers leave at seconds 900 and 2100, denoted ‘L’, severing the majority of partnerships and leaving a (possibly disconnected) P2P overlay with 100 peers. As in Tab. 4.8, we observe higher variation in savings for these small overlays. After both arrival and departure events, AERO achieves equivalent or higher savings compared to static configuration of seeding ratios.

4.7.4 Local minima

Figure 4.9 shows a drop in savings for DIV4 scenario, in the last few minutes (near 1200 seconds). AERO systematically decreases the seeding ratio to avoid getting stuck at local minima. Figure 4.14 presents the curve of one out of five runs of a longer DIV4 scenario; four other curves have been similar and have not been shown to increase clarity. The drop observed at Figure 4.9 becomes the expected upside down ripples on Figure 4.14. It results from a burst of emergency requests due to underprovisioning $|O_s|$, leading to a re-convergence process. Re-convergence happens more often in scenarios where global minimum is high, compared with scenarios such as BASE, because of the multiplicative decay. The ripple can be eliminated by removing this multiplicative decay, or reduced by adjusting it to a lower value. We have tested with and without a 5% decay. However, the loss in savings caused by the ripple is compensated by better convergence outside of local minima.

4.7.5 Summary

AERO dynamically computes the number of prefetchers. We have evaluated AERO for different peer upload bandwidths, under different overlay properties and sizes, and for adverse peer behavior. We have shown that AERO reduces emergency requests and consistently achieves higher P2P distribution efficiency and higher savings.

4.8 AERO at TVPP's Emergency Request Service

We have discussed AERO implementation at SmoothCache, a hybrid CDN-P2P system. AERO benefits stand out in hybrid CDN-P2P by decreasing CDN bandwidth utilization while it could be increasing the number of seeded peers. Benefits are easy to comprehend since CDN servers respond for both seeded and emergency requests. However, emergency requesting for pure P2P systems might be implemented differently making AERO implementation unclear.

Compared to the proposed Emergency Request Service for pure P2P systems, the CDN acts both as server and emergency request handler. There are a few ways to implement emergency requesting on pure P2P systems and, thus, a few AERO approaches. Remember that AERO is an effort to adjust chunk seeding costs to a minimum value that would provide all chunks to all peers by finding an optimal balance between seeding and answering emergency requests. Below we discuss one complete approach.

TVPP's Emergency Request Service proposal separates the role of source and emergency request handlers. Handlers should be peers which have high and spare upload bandwidth to deal with emergency requests. In this approach we argue to employ a bandwidth aware overlay maintenance policy on TVPP and limited number of output partners. Handlers would then be elected from within the overlay – using spare bandwidth and upload capacity measurements – and placed near the source. Inclusion on the Emergency Request Service handlers set would bring the benefit of being closer to the stream source, increasing chunk delivery odds and reducing latency. This would mimic hybrid CDN-P2P, considering source's direct partners as the CDN layer from Figure 4.1. The major differences are that these peers capacity is not dedicated to streaming, will not vary to adapt to the load, nor they can be trusted to stay in the system indefinitely.

The Emergency Request Service can trivially account for emergency requests and adjust handlers's out-degree. Since handlers are not dedicated, the Emergency Request Service has to identify if the handler set has enough spare resources to serve the amount of emergency requests issued and, otherwise, more handlers should be added. Such mechanism should like-wise reduce emergency requesting thereby mitigating handlers bandwidth utilization.

Other ideas derive from this approach. For example, the system could use the source as the handler but that would demand the source to have a enough capacity to scale its out-degree up if needed. Further, handlers could be instantiated instead of elected, thus a content provider would have more control about delivery adding/removing dedicated handlers considering the Emergency Request Service demand.

4.9 Conclusion

CDN-P2P live streaming systems allow clients to issue emergency requests and guarantees QoS. In this chapter, we have shown that emergency requests deliver chunks close to their playback deadlines, which leaves little to no time for redistribution in the P2P overlay. This leads to decreased P2P distribution efficiency, which causes even more emergency requests, which ultimately results in higher bandwidth costs for content providers.

We have presented AERO, a mechanism to minimize bandwidth consumption in CDN-P2P live streaming systems. AERO first increases the number of peers into the P2P overlay which are being seeded by servers to guarantee that the overlay is effectively using peer upload bandwidth to redistribute chunks, reducing emergency requests. AERO then attempts to reduce bandwidth consumption at servers reducing the amount of seeded peers without increasing the rate of emergency requests. Our evaluation of AERO under diverse conditions shows that it achieves up to 30% higher server upload bandwidth savings compared to static allocation of the number of seeded peers.

AERO requires readily available information: the number of peers into the P2P overlay which are being seeded by servers and servers bandwidth consumption. AERO runs entirely on CDN servers and does not require modification of client software. AERO also does not impose any restriction on and is compatible with existing P2P overlay construction and chunk scheduling mechanisms. We believe AERO can be easily integrated into existing systems that use emergency requests or similar mechanisms (such as into Section 2.4) to guarantee quality of service, helping improve scalability and reducing operational costs.

As future work, we plan to investigate how to improve AERO's convergence delay by computing constant scaling factors dynamically and using historical information, and to better classify requests done to servers so AERO would have the emergency request rate as input instead of estimating it through servers bandwidth consumption.

Chapter 5

Final Remarks

Live streaming systems are becoming more popular each day. These systems attract a growing number of users, and some important TV channels already broadcast their live content on the Internet. Peer-to-peer live streaming systems may provide a low cost solution to the problem of transmitting live content to a large number of viewers.

This technology relies on splitting the stream into chunks and spreading them to the P2P overlay participants. Peers distribute each chunk among their partners, so that all peers receive all chunks. Whenever a chunk misses its playback deadline either the player will have to wait until it arrives, delaying the peer in relation to others, or it will skip it. For the viewer, both scenarios are undesirable and indicatives of the system's bad quality. Yet today most commercial P2P systems are unreliable, as quality assurance mechanisms are still in a developing stage, or even completely ignored, and several scenarios can cause degradation of system performance experienced by peers.

This work have studied how to ensure quality of service through minimization of chunk loss, understanding the reason behind losses, and proposing new ways to respond to each reason. We have developed a P2P live streaming system, characterized chunk losses using it, studied the effect of free riders, developed a non-punitive algorithm to reduce peer uncooperativeness impact, proposed an emergency request mechanism that aims to eliminate chunk losses, and developed an algorithm to reduce costs associated with emergency requesting.

First, we have presented TVPP, an academic research-oriented P2P live streaming system designed to provide a similar service to popular proprietary commercial systems, such as SopCast. We have exposed several details of TVPP design and architecture in the light of overlay construction and chunk scheduling theory. We have also described the design of an emergency request feature. We have shown what TVPP is currently able to log, and some ways on how it can be expanded for other researches. We have introduced the current parameter set that can be fixed at TVPP runtime. Finally, we have highlighted benefits of TVPP usage through an experimental comparison with SopCast about traffic and network related metrics.

Second, we have characterized chunk loss causes using TVPP. We have evaluated both a resourceful and a bandwidth constrained scenario. We have found that for a resourceful scenario most peers do not have chunk loss problems, while a few peers present significant losses. System-wide average chunk loss has been stable in time. While losses do occur, there are many moments where peers had no requests made to them. Mostly, requests are responded in the first attempt, by the first peer to announce that it has the chunk. Chunks that have been missed have had a worse chunk spread – having less candidates to request from – than chunks that have been received. Finally, chunk loss lacks spatial and temporal locality. These results suggest that two solutions would be effective to overcome these losses: an improved overlay organization policy, and an emergency request mechanism (described in Section 2.4) to request "about to be lost" chunks outside of a peer's partner set.

Losses get worse, yet sustainable, once we reduced bandwidth through the introduction of free riders. We present the concept of conscious and oblivious free riders. Observing conscious free riders we have noted that chunk loss has been qualitatively similar for fractions of free rides below 50% while latency has increased by 1s. However, the same fraction of oblivious free riders have caused a degradation significantly worse, mostly triggered by the increase in retries. Consecutive retries waste bandwidth and increase average chunk latency.

We have introduced Simple Unanswered Request Eliminator (SURE), a modification to our system's request scheduler that avoids excessive retries caused by oblivious free riders because it identifies uncooperative peers with few interactions. Using SURE we have obtained similar results for the same amount of conscious and oblivious free riders. While studying chunk loss, we have shown that P2P live streaming systems can sustain 50% of free riders with negligible degradation.

Third, we have studied an alternative to reduce chunk losses that uses CDN architecture. We have worked with Hive Streaming, a commercial modern live streaming solution. Hive uses SmoothCache, a hybrid CDN-P2P live streaming system, as its distribution platform. SmoothCache includes a concept that ensures chunk delivery – typical of CDN peer-assisted systems. The concept states that all data is acquirable from CDN; those who are interested in the content form a support P2P overlay, which should be queried first for data. SmoothCache peers know CDN servers addresses and, ultimately, will request chunks to the CDN if a chunk cannot be found or a request fail to be responded by its partners. We argue that this design can be mapped to the emergency request mechanism proposed for P2P systems in Section 2.4 and results obtained on SmoothCache would be equivalent.

We have found that, although emergency requests guarantee quality of service, they deliver a chunk close to its playback deadline, which leaves little or no time for redistribution in the P2P overlay. The benefit caused by recovering a chunk through emergency requests is hardly propagated to a peer's partners, which can cause a chain reaction. This avoids chunk losses but is inefficient to solve P2P data availability problems that might be causing them.

We have developed Adaptive Emergency Request Optimization (AERO), a mechanism that dynamically minimizes the number of video streams seeded to the P2P overlay while guaranteeing efficient dissemination and avoiding emergency requests. We have shown that AERO significantly reduces bandwidth consumption at the streaming infrastructure (up to 30%) compared to a static configuration of the number of seeded streams. We have evaluated AERO under different scenarios varying peer upload bandwidth, fraction of free-riding peers, peer churn (up to flash crowds), overlay size, overlay construction and organization mechanism, and underlying physical network conditions, with positive results in all cases. AERO does not impose any restriction on and is compatible with any peer-to-peer overlay construction and chunk scheduling mechanisms. Although developed using SmoothCache, which is a CDN peer-assisted system, AERO can be applied to "pure" P2P overlays. We believe AERO can be easily integrated into systems that use emergency requests or similar mechanisms to guarantee quality of service, helping to improve scalability and reducing operational costs by by adjusting emergency requests handlers's out-degree.

In summary we have found out that most losses happen for reasons too specific to be pointed out and have it treated. One important reason that has been identified, however, was requesting chunks to uncooperative peers. We have proposed a simple method to track this behavior and avoid it. For other reasons, we have proposed a service where chunks that would be lost could have a last chance to be retrieved outside of the peer's neighborhood. This solution has potential to guarantee close to 100% delivery depending on how it is implemented. Nevertheless, such method to neutralize losses might incur in additional infrastructure costs as an external entity will have to handle that chunk delivery. While this may be unavoidable, we have proposed a method to minimize these costs. Finally, in this work we have obtained a better understanding of why chunks are lost in the transmission of live P2P streams, and how to minimize these losses. A transmission system using these techniques will be significantly more efficient to transmit streams without increasing costs as much as possible.

Future work includes the implementation of the emergency request service in TVPP, as well as its multiple conceptual approaches, resulting in tests, analyses and comparisons to determine the potentials of each approach. Some of these ideas were discussed at Section 2.4 through questions such as **1)** "how to select emergency request handlers" or **2)** "to which handler forward a emergency request if there is more than one handler".

Another activity would be to further explore tweaks over AERO algorithm, determining if there is a better configuration for its parameters. AERO results from exploratory research; seventeen versions of the algorithm were created with slightly different ideas such as **1)** setting the seeding ratio straight to the consumption level that CDN servers were observing, **2)** identifying requests done to the servers as prefetched and emergency ones, and use this to reduce emergency requests to zero, **3)** using thresholds to identify if the system is stable or should adapt, e.g., if emergency requests are more than 10% of all server upload or if prefetched requests are less than 90% of the expected server upload (which considers that all prefetchers will request all chunks to servers), or **4)** using thresholds to perform more frequent or bigger seeding ratio adjustments. AERO has initially achieved good results but one could better organize and structure the ideas from its versions.

Finally, many possibilities would arise from implementing AERO in TVPP, especially since handlers chosen among peers are not dedicated to just streaming and they can leave the system at anytime. Ideas presented at Section 4.8 are future work themselves as they depend on the implementation of TVPP's emergency request service.

Bibliography

Adar, E. and Huberman, B. (2000). Free Riding on Gnutella. *First Monday*, 5(10-2).

Ali, S., Mathur, A., and Zhang, H. (2006). Measurement of commercial peer-to-peer live video streaming. In *Proc. of Workshop in Recent Advances in Peer-to-Peer Streaming.*

Amazon (2014). Amazon will stream in ultra-high def 4k by january. `http://time.com/3581899/amazon-4k-streaming-prime-instant-video/`.

Asaduzzaman, S., Qiao, Y., and Bochmann, G. (2008). Cliquestream: an efficient and fault-resilient live streaming network on a clustered peer-to-peer overlay. In *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*, pages 269--278. IEEE.

Banerjee, S., Bhattacharjee, B., and Kommareddy, C. (2002). Scalable application layer multicast. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, page 217. ACM.

Bellovin, S. M. (2002). A technique for counting natted hosts. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 267--272, New York, NY, USA. ACM.

Birke, R., Kiraly, C., Leonardi, E., Mellia, M., Meo, M., and Traverso, S. (2011). Hose rate control for p2p-tv streaming systems. In *P2P, 2011 IEEE International Conference on*, pages 202--205. IEEE.

Carlsson, N. and Eager, D. L. (2007). Peer-assisted on-demand streaming of stored media using bittorrent-like protocols. In *NETWORKING. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 570--581. Springer.

Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., and Singh, A. (2003). Splitstream: high-bandwidth multicast in cooperative environments. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 298--313. ACM.

Cisco (2015). Cisco visual networking index: Forecast and methodology, 2014-2019. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white-paper-listing.html.

CNET (2010). The next big thing supersession: I want my iptv. Filme-video.

CNN (2015). Periscope: Four ways it's shaking up media. http://edition.cnn.com/2015/05/26/tech/periscope-android-media/.

Cohen, B. (2003). Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, volume 6. Berkeley, CA, USA.

Deshpande, H., Bawa, M., and Garcia-Molina, H. (2002). Streaming Live Media over a Peer-to-Peer Network. *Stanford database group technical report (2002).*

Diestel, R. (1997). *Graph Theory.* New York: Springer-Verlag, 3rd edition.

Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B. (2002). Globally distributed content delivery. *Internet Computing, IEEE*, 6(5):50–58. ISSN 1089-7801.

Dowling, J. and Payberah, A. H. (2012). Shuffling with a croupier: Nat-aware peer-sampling. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 102--111. IEEE.

Felber, P. and Biersack, E. W. (2005). Cooperative content distribution: Scalability through self-organization. In *Self-star Properties in Complex Information Systems*, pages 343--357. Springer.

Fodor, V. and Dan, G. (2007). Resilience in live peer-to-peer streaming [peer-to-peer multimedia streaming]. *Communications Magazine, IEEE*, 45(6):116--123. ISSN 0163-6804.

Fortuna, R., Leonardi, E., Mellia, M., Meo, M., and Traverso, S. (2010). Qoe in pull based p2p-tv systems: overlay topology design tradeoffs. In *Peer-to-Peer Computing (P2P), 10th International Conference on*, pages 1--10. IEEE.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software.* Pearson Education.

Gonçalves, G., Cunha, Í., Vieira, A., and Almeida, J. (2014). Predicting the level of cooperation in a peer-to-peer live streaming application. *Multimedia Systems*, pages 1–20. ISSN 0942-4962.

Guerraoui, R., Huguenin, K., Kermarrec, A., Monod, M., and Prusty, S. (2010). Lifting: lightweight freerider-tracking in gossip. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, pages 313--333. Springer-Verlag.

Guha, S., Daswani, N., and Jain, R. (2006). An experimental study of the skype peer-to-peer voip system. In *Proc. of IPTPS*, volume 6. Citeseer.

Hefeeda, M., Habib, A., Botev, B., Xu, D., and Bhargava, B. (2003). PROMISE: peer-to-peer media streaming using CollectCast. *Proceedings of the eleventh ACM international conference on Multimedia*, pages 45--54.

Hei, X., Liang, C., Liang, J., Liu, Y., and Ross, K. (2007a). A Measurement Study of a Large-Scale P2P IPTV System. *Multimedia, IEEE Transactions on*, 9(8):1672--1687.

Hei, X., Liang, C., Liang, J., Liu, Y., and Ross, K. (2007b). A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, 9(8):1672--1687.

Hei, X., Liu, Y., and Ross, K. (2008). IPTV over P2P streaming networks: the mesh-pull approach. *Communications Magazine, IEEE*, 46(2):86--92. ISSN 0163-6804.

Hitbox (2015). Hitbox. `http://www.hitbox.tv/`.

Hive (2015). Hive streaming. `https://www.hivestreaming.com/`.

Horvath, A., Telek, M., Rossi, D., Veglia, P., Ciullo, D., Garcia, M., Leonardi, E., and Mellia, M. (2008). Dissecting PPLive, SopCast, TVAnts. *submitted to ACM Conext.*

Huang, Y., Fu, T. Z., Chiu, D.-M., Lui, J. C., and Huang, C. (2008). Challenges, design and analysis of a large-scale p2p-vod system. *SIGCOMM Comput. Commun. Rev.*, 38(4):375--388. ISSN 0146-4833.

ITU-T (2015). H.264: Advanced video coding for generic audiovisual services. `http://www.itu.int/rec/T-REC-H.264`.

Jannotti, J., Gifford, D., Johnson, K., Kaashoek, M., and O'Toole Jr, J. (2000). Overcast: reliable multicasting with on overlay network. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*, pages 14--14. USENIX Association Berkeley, CA, USA.

Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., and Van Steen, M. (2007). Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)*, 25(3):8.

Jian, F. (2009). A research on scheduling strategy in peer-to-peer streaming media. In *Circuits, Communications and Systems, 2009. PACCS'09. Pacific-Asia Conference on*, pages 439--442. IEEE.

Jin, Y., Yi, Y., Kesidis, G., Kocak, F., and Shin, J. (2013). Hybrid Client-Server and Peer-to-Peer Caching Systems with Selfish Peers. In *Proc. IEEE INFOCOM*.

Kostić, D., Rodriguez, A., Albrecht, J., and Vahdat, A. (2003). Bullet: high bandwidth data dissemination using an overlay mesh. *ACM SIGOPS Operating Systems Review*, 37(5):282--297.

Kreitz, G. and Niemela, F. (2010). Spotify – large scale, low latency, p2p music-on-demand streaming. In *Peer-to-Peer Computing (P2P), IEEE Tenth International Conference on*, pages 1–10.

Kulbak, Y. and Bickson, D. (2005). The eMule Protocol Specification. *eMule project, http://sourceforge.net.*

Kumar, R., Liu, Y., and Ross, K. (2007). Stochastic fluid theory for p2p streaming systems. In *Computer Communications (INFOCOM), IEEE 26th International Conference on*, pages 919–927. ISSN 0743-166X.

Li, B., Xie, S., Qu, Y., Keung, G., Lin, C., Liu, J., and Zhang, X. (2008). Inside the new coolstreaming: Principles, measurements and performance implications. In *Proc. INFOCOM*.

Liu, F., Li, B., Zhong, L., Li, B., Jin, H., and Liao, X. (2012). Flash crowd in P2P live streaming systems: fundamental characteristics and design implications. *Parallel and Distributed Systems, IEEE Transactions on*, 23(7):1227–1239. ISSN 1045-9219.

Lu, Y., Fallica, B., Kuipers, F., Kooij, R., and Van Mieghem, P. (2009). Assessing the Quality of Experience of SopCast. *IJIPT*, 4(1):11--23.

Lu, Z., Wang, Y., and Yang, Y. R. (2012). An analysis and comparison of cdn-p2p-hybrid content delivery system and model. *Journal of Communications*, 7(3):232--245.

Magharei, N. and Rejaie, R. (2006). Understanding mesh-based peer-to-peer streaming. In *NOSSDAV '06: Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, pages 1--6, New York, NY, USA. ACM.

Magharei, N., Rejaie, R., and Guo, Y. (2007). Mesh or multiple-tree: A comparative study of live p2p streaming approaches. In *IEEE INFOCOM 2007. 26th IEEE International Conference on Computer Communications*, pages 1424--1432.

Mansy, A. and Ammar, M. (2011). Analysis of adaptive streaming for hybrid cdn/p2p live video systems. In *Network Protocols (ICNP), 19th International Conference on*, pages 276--285. IEEE.

Mol, J.-D., Epema, D. H., and Sips, H. J. (2007). The orchard algorithm: Building multicast trees for p2p video multicasting without free-riding. *Multimedia, IEEE Transactions on*, 9(8):1593--1604.

Moltchanov, D. (2011). Service quality in p2p streaming systems. *Computer Science Review*, 5(4):319 – 340. ISSN 1574-0137.

Moreira, L. (2015). Fifa 2014 world cup live stream architecture. `http://leandromoreira.com.br/2015/04/26/fifa-2014-world-cup-live-stream-architecture/`.

Oliveira, J., Miguel, E., Ítalo Cunha, Vieira, A. B., Rocha, M., and Campos, S. (2013a). Can p2p live streaming systems coexist with free riders? In *P2P*. IEEE.

Oliveira, J., Viana, R., Vieira, A. B., Rocha, M., and Campos, S. (2013b). Tvpp: A research oriented p2p live streaming system. In *SBRC 2013 - Salão de Ferramentas*, Brasília-DF.

Oliveira, J. F. A. (2010). Super nós em sistemas p2p de distribuição de mídia ao vivo. Master's thesis, UFMG.

Pai, V., Kumar, K., Tamilmani, K., Sambamurthy, V., and Mohr, A. E. (2005). Chainsaw: Eliminating trees from overlay multicast. In *Peer-to-peer systems IV*, pages 127--140. Springer.

Park, K., Pack, S., and Kwon, T. (2008). Climber: an incentive-based resilient peer-to-peer system for live streaming services. In *IPTPS*, page 10.

Payberah, A. H., Dowling, J., Rahimian, F., and Haridi, S. (2012a). Distributed optimization of p2p live streaming overlays. *Springer Computing, Special Issue on Extreme Distributed Systems: From Large Scale to Complexity*, 94(8):621--647.

Payberah, A. H., Kavalionak, H., Kumaresan, V., Montresor, A., and Haridi, S. (2012b). Clive: Cloud-assisted p2p live streaming. In *Peer-to-Peer Computing (P2P), 12th International Conference on*, pages 79--90. IEEE.

Pianese, F., Perino, D., Keller, J., and Biersack, E. W. (2007). Pulse: an adaptive, incentive-based, unstructured p2p live streaming system. *Multimedia, IEEE Transactions on*, 9(8):1645--1660.

Piatek, M., Krishnamurthy, A., Venkataramani, A., Yang, R., Zhang, D., and Jaffe, A. (2010). Contracts: Practical Contribution Incentives For P2P Live Streaming. In *USENIX NSDI*.

Picconi, F. and Massoulié, L. (2008). Is there a future for mesh-based live video streaming? *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, pages 289–298. ISSN .

Planet Lab (2010). Planetlab. `http://www.planet-lab.org/`.

Ripeanu, M. (2001). Peer-to-Peer Architecture Case Study: Gnutella Network. In *Proceedings of International Conference on Peer-to-peer Computing*, volume 101. Sweden: IEEE Computer Press.

Roverso, R., Dowling, J., and Jelasity, M. (2013). Through the wormhole: Low cost, fresh peer sampling for the internet. In *Peer-to-Peer Computing (P2P), 13th International Conference on*, pages 1--10. IEEE.

Roverso, R., El-Ansary, S., and Haridi, S. (2012). Smoothcache: Http-live streaming goes peer-to-peer. In *IFIP NETWORKING 2012*, pages 29--43. Springer.

Roverso, R., Reale, R., El-Ansary, S., and Haridi, S. (2015). Smoothcache 2.0: Cdn-quality adaptive http live streaming on peer-to-peer overlays. In *Proceedings of the 6th ACM Multimedia Systems Conference*, MMSys '15, pages 61--72, New York, NY, USA. ACM.

Sentinelli, A., Marfia, G., Gerla, M., Kleinrock, L., and Tewari, S. (2007). Will IPTV ride the peer-to-peer stream?[Peer-to-Peer Multimedia Streaming]. *Communications Magazine, IEEE*, 45(6):86--92. ISSN 0163-6804.

Shirky, C. (2001). Listening to Napster. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, pages 19--28.

Silva, T. H. (2009). Transmissões de vídeo ao vivo geradas por usuários: Caracterização e análise. Master's thesis, Universidade Federal de Minas Gerais.

Silverston, T., Fourmaux, O., Botta, A., Dainotti, A., Pescapé, A., Ventre, G., and Salamatian, K. (2009). Traffic analysis of peer-to-peer IPTV communities. *Computer Networks*, 53(4):470--484. ISSN 1389-1286.

Simoni, G., Roverso, R., and Montresor, A. (2014). Rankslicing: A decentralized protocol for supernode selection. In *Peer-to-Peer Computing (P2P), 14th International Conference on*, pages 1--10. IEEE.

Skiena, S. (1991). *Implementing discrete mathematics: combinatorics and graph theory with Mathematica*. Addison-Wesley Longman Publishing Co., Inc. ISBN 0201509431.

Tang, S., Lu, Y., Hernández, J., Kuipers, F., and Van Mieghem, P. (2009). Topology dynamics in a P2PTV network. *Proc. NETWORKING*.

Tewari, S. and Menon, S. (2009). On resource provisioning in hybrid peer-to-peer live streaming systems. In *Broadband Multimedia Systems and Broadcasting (BMSB), International Symposium on*, pages 1–6. IEEE.

Tran, D., Hua, K., and Do, T. (2004). A peer-to-peer architecture for media streaming. *Selected Areas in Communications, IEEE Journal on*, 22(1):121--133.

Tran, D. A., Hua, K. A., and Do, T. (2003). Zigzag: An efficient peer-to-peer scheme for media streaming. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1283--1292. IEEE.

Traverso, S., Abeni, L., Birke, R., Kiraly, C., Leonardi, E., Lo Cigno, R., and Mellia, M. (2014). Neighborhood filtering strategies for overlay construction in p2p-tv systems: Design and experimental comparison. *Networking, IEEE/ACM Transactions on*, PP(99):1–1. ISSN 1063-6692.

Twitch TV (2015). Twitch tv. `http://www.twitch.tv/`.

Vieira, A., Gomes, P., Rocha, M., Almeida, J., and Campos, S. (2009). A behaviour model of the SopCast users. In *Proc. WEBMEDIA*.

Vlavianos, A., Iliofotou, M., and Faloutsos, M. (2006). Bitos: Enhancing bittorrent for supporting streaming applications. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1--6. IEEE.

Voulgaris, S., Gavidia, D., and Van Steen, M. (2005). Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197--217.

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

Wichtlhuber, M., Richerzhagen, B., Ruckert, J., and Hausheer, D. (2014). TRANSIT: Supporting Transitions in Peer-to-Peer Live Video Streaming. In *Proc. IFIP Networking*.

Wu, C., Li, B., and Zhao, S. (2007). Characterizing Peer-to-Peer Streaming Flows. *Selected Areas in Communications, IEEE Journal on*, 25(9):1612--1626.

Yin, H., Liu, X., Zhan, T., Sekar, V., Qiu, F., Lin, C., Zhang, H., and Li, B. (2009). Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 25--34. ACM.

Zhang, L. (2005). *Efficient video streaming in peer-to-peer networks*. PhD thesis, The Hong Kong Polytechnic University.

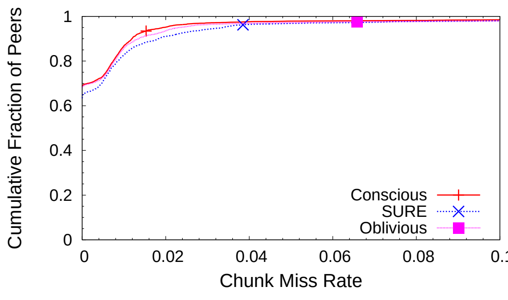
Zhang, X., Liu, J., Li, B., and Yum, T. (2005). CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In *Proc. IEEE Infocom*.

Zhao, B. Q., Lui, J. C.-S., and Chiu, D.-M. (2009). Exploring the optimal chunk selection policy for data-driven p2p streaming systems. In *Peer-to-Peer Computing (P2P), IEEE Ninth International Conference on*, pages 271--280. IEEE.

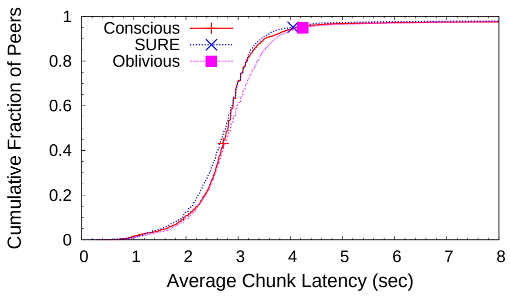
Zhao, M., Aditya, P., Chen, A., Lin, Y., Haeberlen, A., Druschel, P., Maggs, B., Wishon, B., and Ponec, M. (2013). Peer-assisted Content Distribution in Akamai NetSession. In *Proc. IMC*.

Appendix A

SURE Comparison Figures

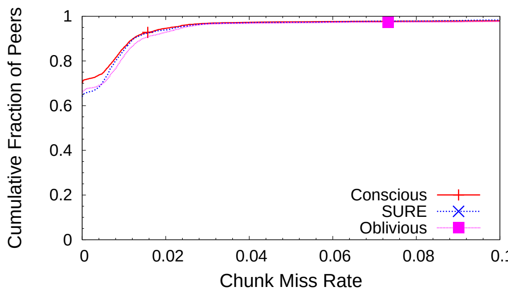


(a) 00% - Chunk Loss Comparison

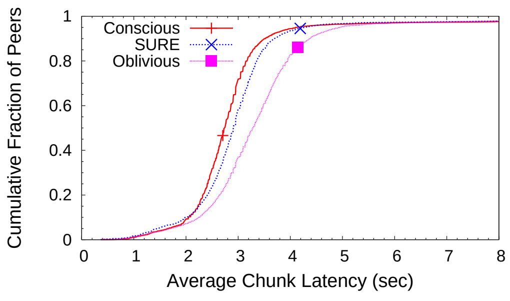


(b) 00% - Latency Comparison

Figure A.1. Chunk loss and latency comparison over the results of conscious, oblivious and SURE scenarios using 00% free rider ratio.

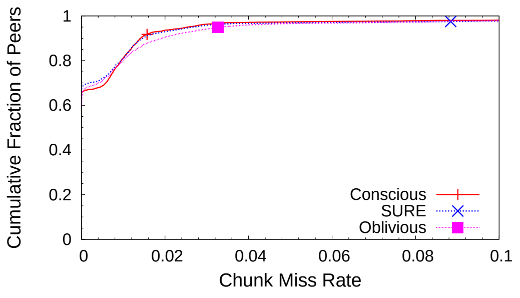


(a) 10% - Chunk Loss Comparison

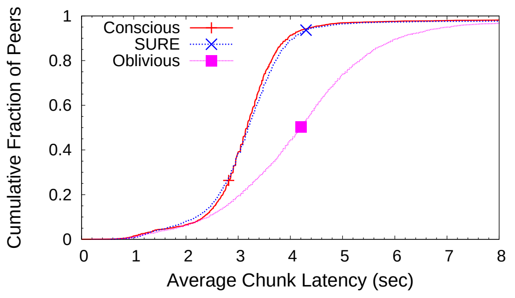


(b) 10% - Latency Comparison

Figure A.2. Chunk loss and latency comparison over the results of conscious, oblivious and SURE scenarios using 10% free rider ratio.

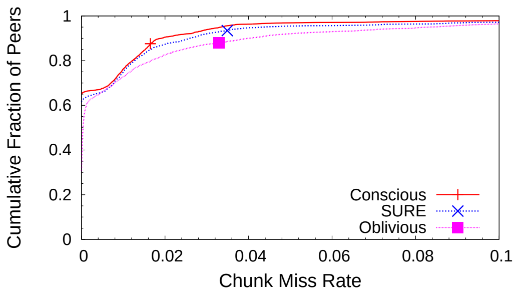


(a) 30% - Chunk Loss Comparison

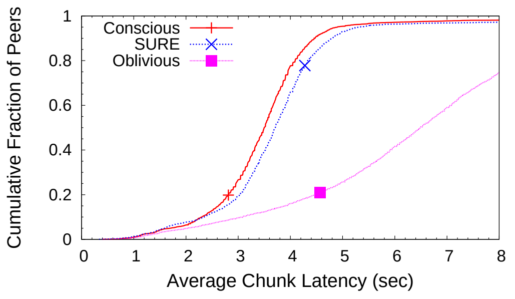


(b) 30% - Latency Comparison

Figure A.3. Chunk loss and latency comparison over the results of conscious, oblivious and SURE scenarios using 30% free rider ratio.

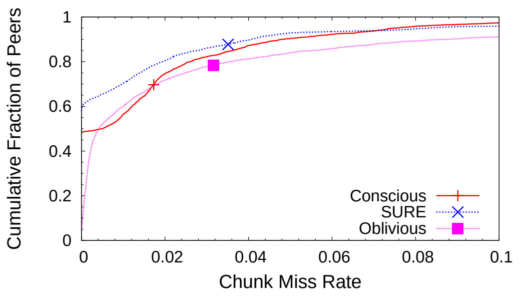


(a) 50% - Chunk Loss Comparison

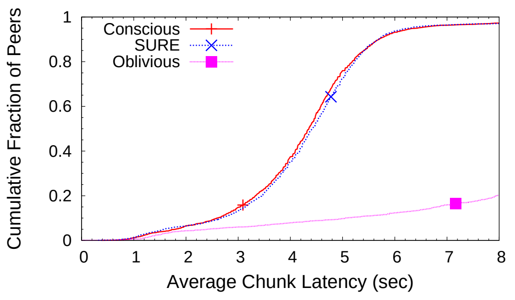


(b) 50% - Latency Comparison

Figure A.4. Chunk loss and latency comparison over the results of conscious, oblivious and SURE scenarios using 50% free rider ratio.



(a) 70% - Chunk Loss Comparison



(b) 70% - Latency Comparison

Figure A.5. Chunk loss and latency comparison over the results of conscious, oblivious and SURE scenarios using 70% free rider ratio.

Appendix B

AERO Evaluation Figures

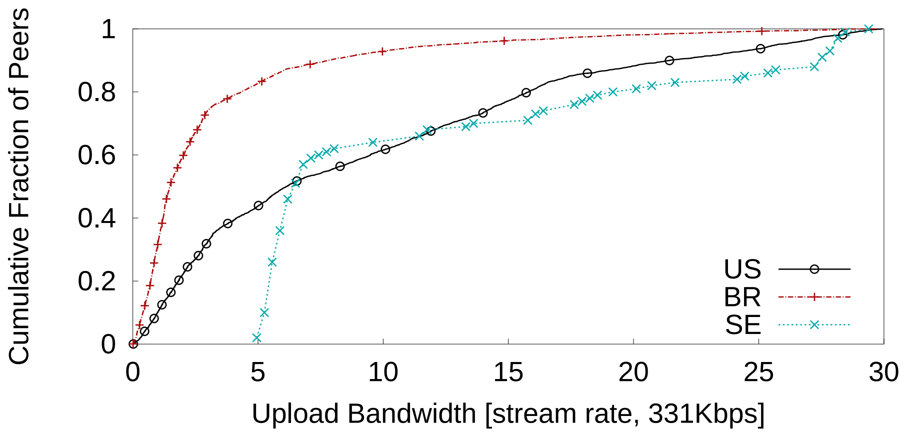


Figure B.1. Peer upload bandwidth distributions.

In each of the following sections, the curve pointed as BASEis given by distributions in Figure B.1 (same as Figure 4.2). 50F, 75F and DIV4 curves are restrictions over BASE, as explained in Section 4.3.

B.1 Brazilian Bandwidth Distribution

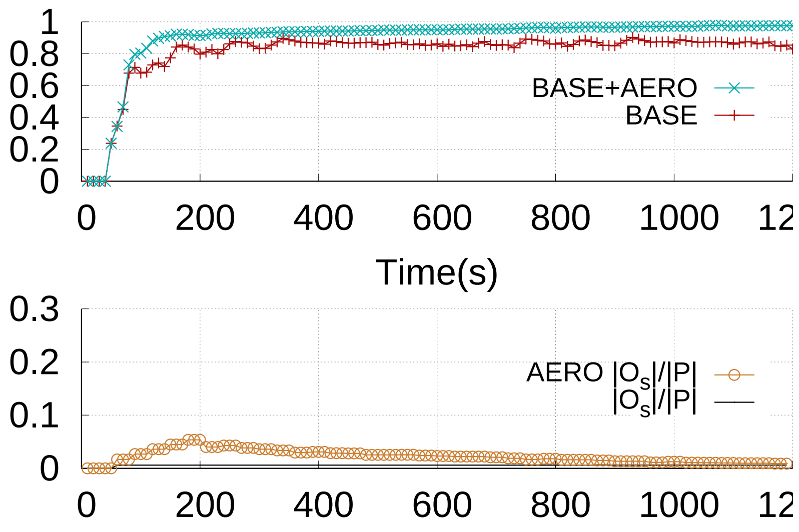


Figure B.2. (BR)Comparison of overall savings for BASE scenario with and without AERO.

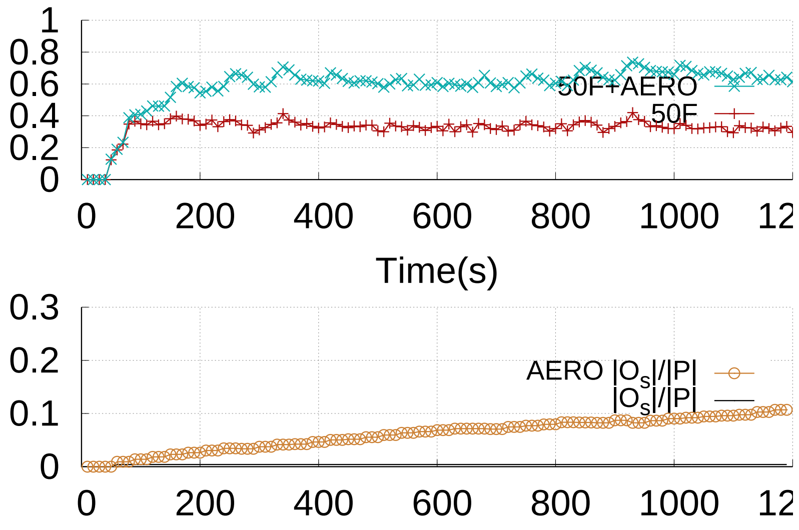


Figure B.3. (BR)Comparison of overall savings for 50F scenario with and without AERO.

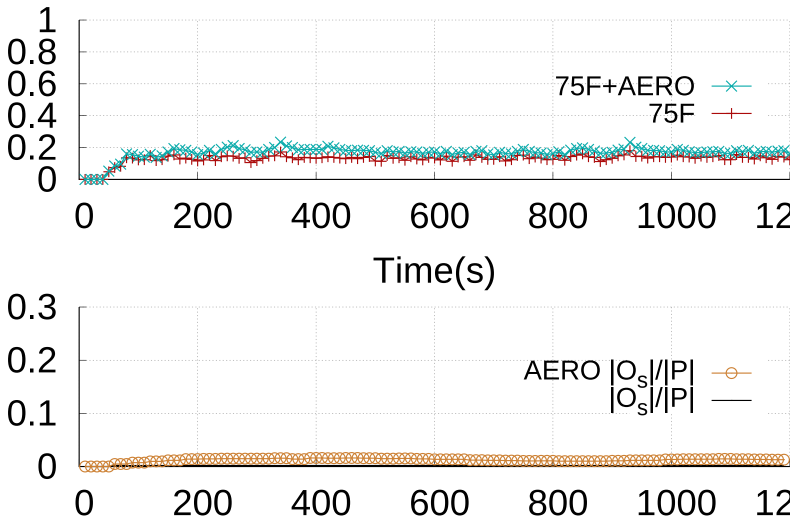


Figure B.4. (BR)Comparison of overall savings for 75F scenario with and without AERO.

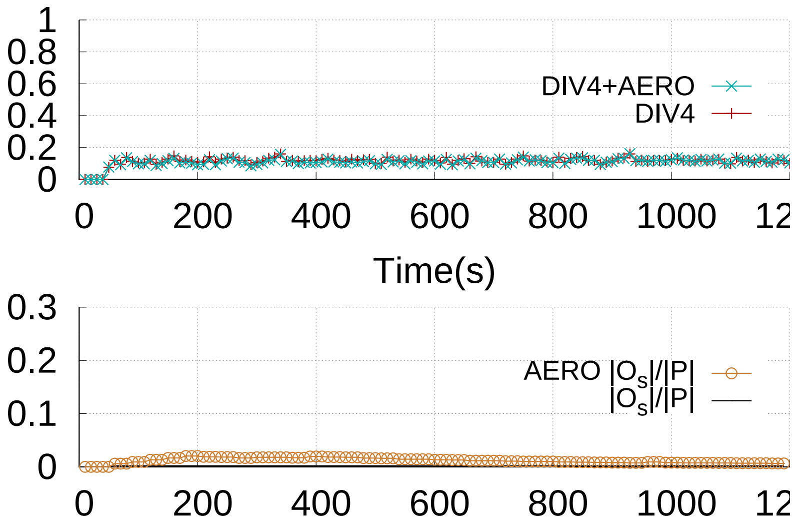


Figure B.5. (BR)Comparison of overall savings for DIV4 scenario with and without AERO. $|\mathcal{O}_S|/|\mathcal{P}|$ removed from key for clarity.

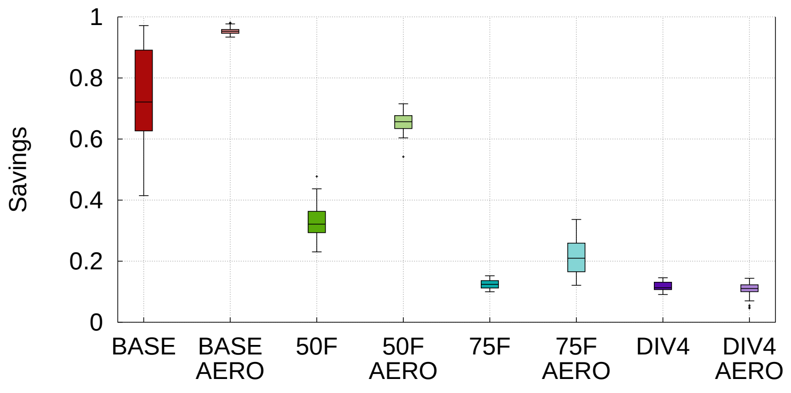


Figure B.6. (BR)Savings of different overlay construction configurations grouped by peer upload bandwidth distribution.

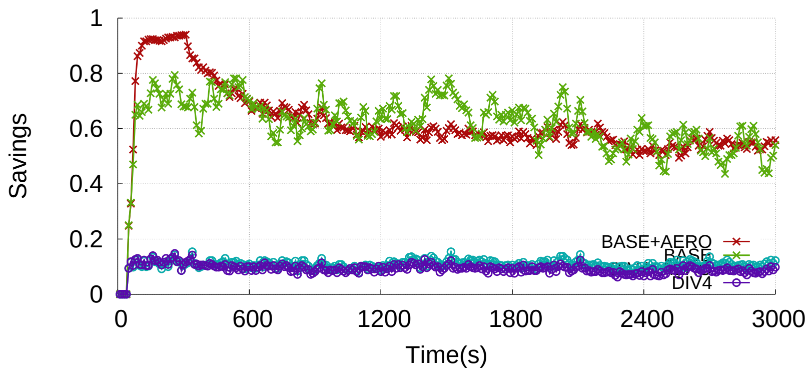


Figure B.7. (BR)Savings under peer churn.

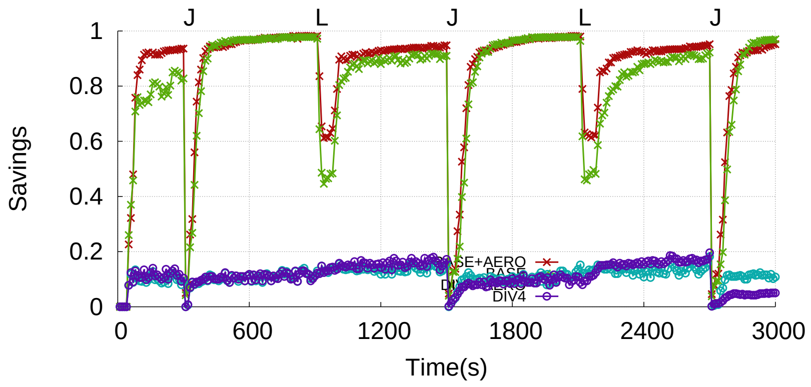


Figure B.8. (BR)Savings over time for periodic flash crowd events.

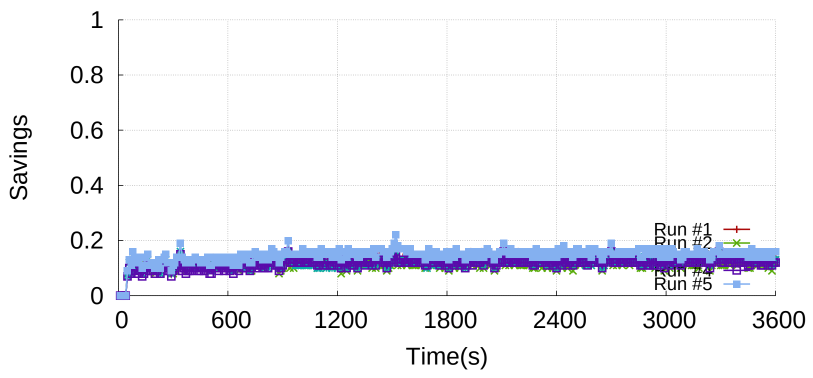


Figure B.9. (BR)Savings for DIV4 scenario with AERO over longer experiments.

B.2 North American Bandwidth Distribution

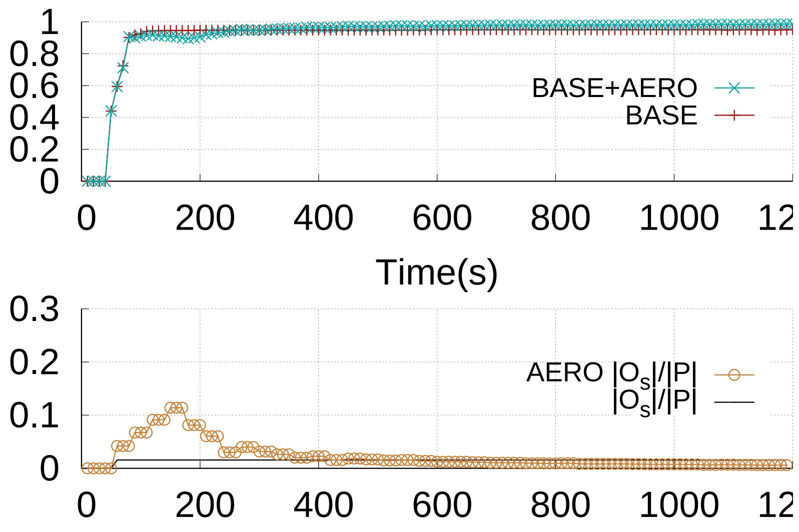


Figure B.10. (US)Comparison of overall savings for BASE scenario with and without AERO.

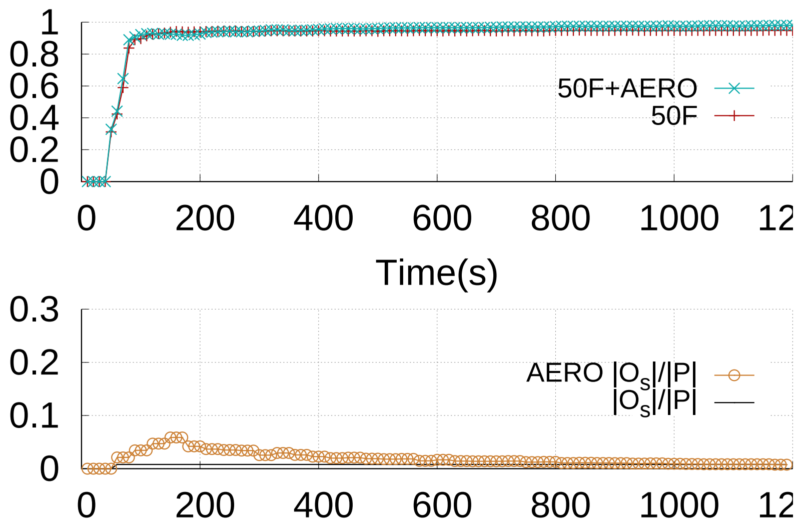


Figure B.11. (US)Comparison of overall savings for 50F scenario with and without AERO.

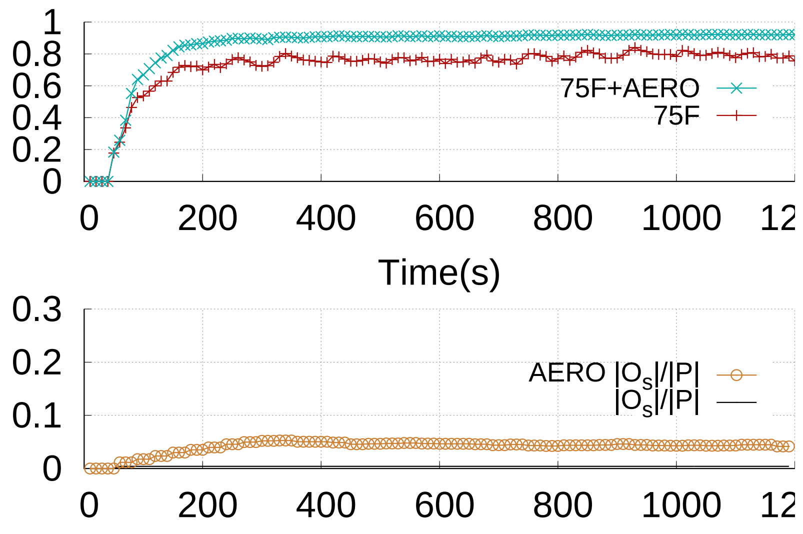


Figure B.12. (US)Comparison of overall savings for 75F scenario with and without AERO.

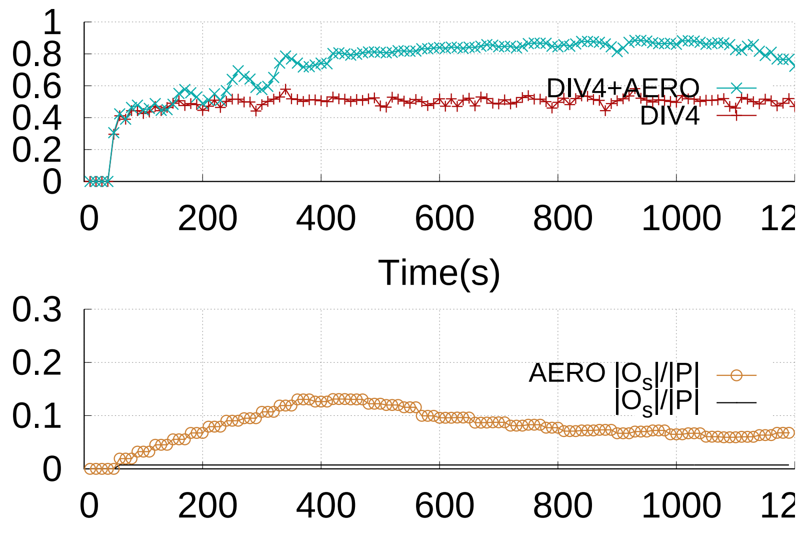


Figure B.13. (US)Comparison of overall savings for DIV4 scenario with and without AERO. $|\mathcal{O}_S|/|\mathcal{P}|$ removed from key for clarity.

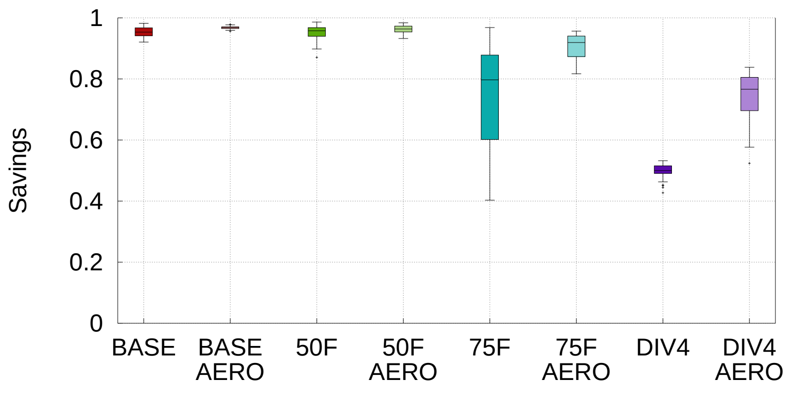


Figure B.14. (US)Savings of different overlay construction configurations grouped by peer upload bandwidth distribution.

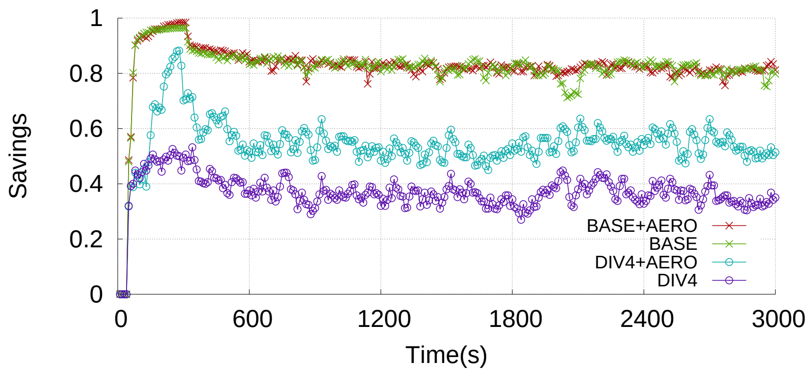


Figure B.15. (US)Savings under peer churn.

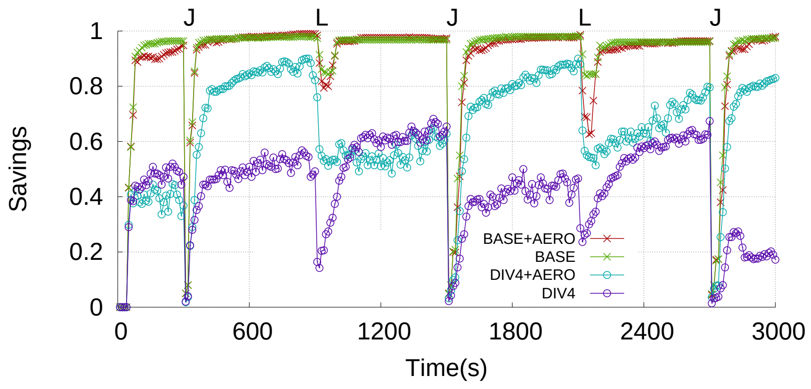


Figure B.16. (US)Savings over time for periodic flash crowd events.

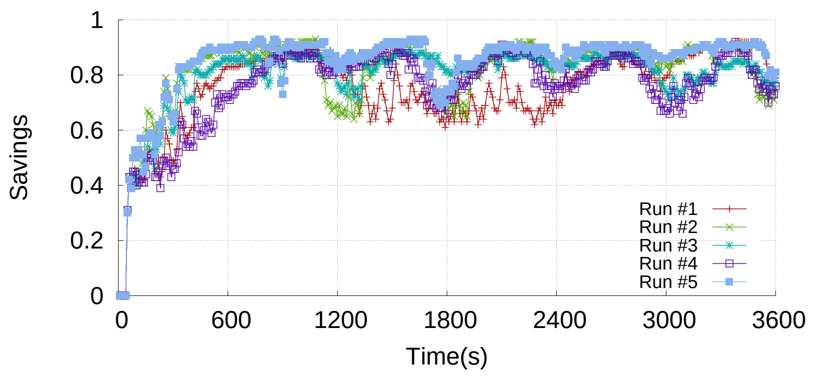


Figure B.17. (US)Savings for DIV4 scenario with AERO over longer experiments.

B.3 Swedish Bandwidth Distribution

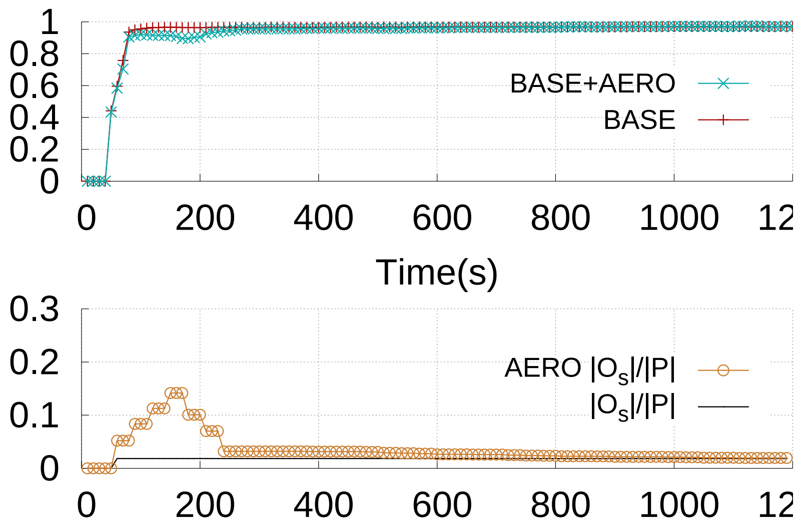


Figure B.18. (SE)Comparison of overall savings for BASE scenario with and without AERO.

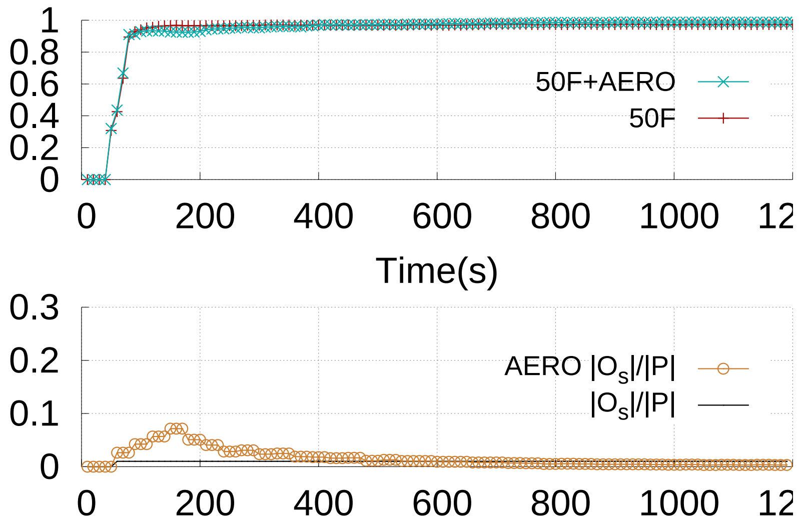


Figure B.19. (SE)Comparison of overall savings for 50F scenario with and without AERO.

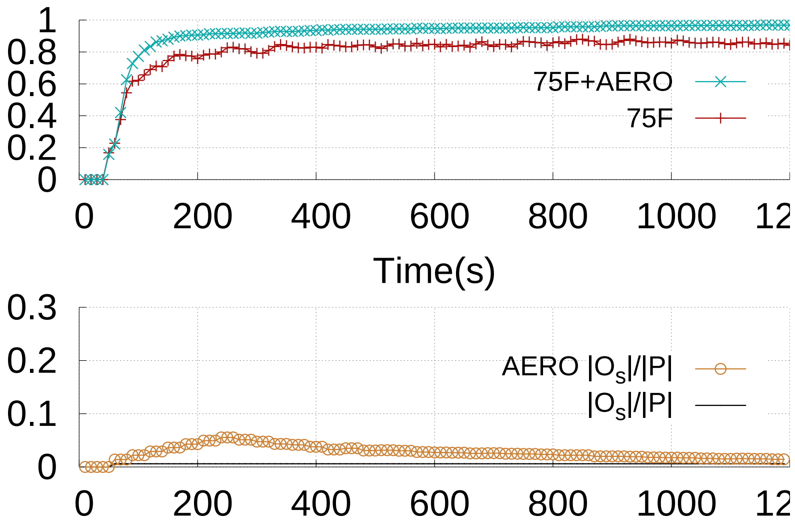


Figure B.20. (SE)Comparison of overall savings for 75F scenario with and without AERO.

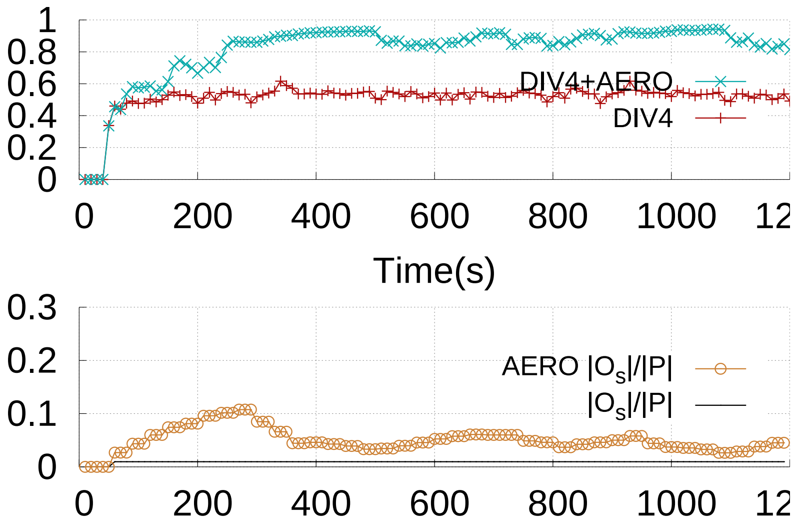


Figure B.21. (SE)Comparison of overall savings for DIV4 scenario with and without AERO. $|\mathcal{O}_S|/|\mathcal{P}|$ removed from key for clarity.

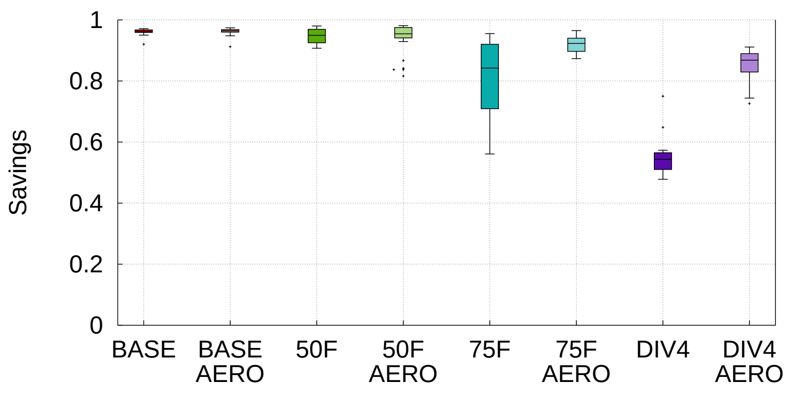


Figure B.22. (SE)Savings of different overlay construction configurations grouped by peer upload bandwidth distribution.

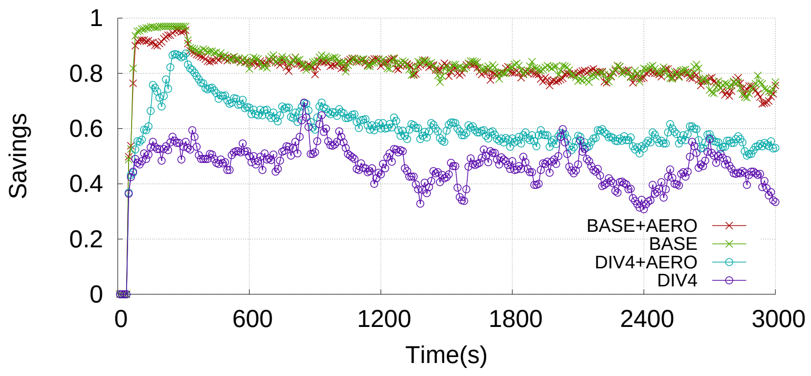


Figure B.23. (SE)Savings under peer churn.

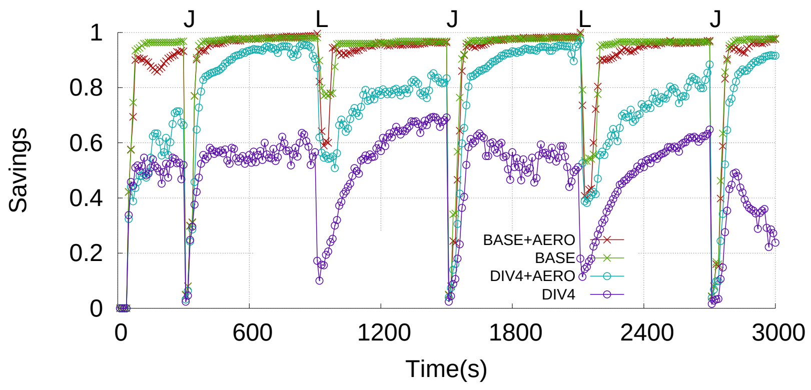


Figure B.24. (SE)Savings over time for periodic flash crowd events.

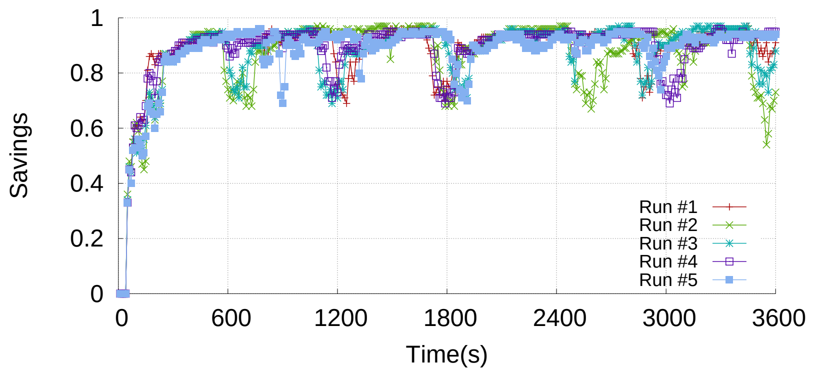


Figure B.25. (SE)Savings for DIV4 scenario with AERO over longer experiments.