ROYAL INSTITUTE
OF TECHNOLOGY

On ISP Friendliness: Introducing an ISP-Friendly Peer-to-Peer Live Streaming System

Niklas Wahlén
nwahlen@kth.se

Master of Science Thesis

Examiner: Dr. Jim Dowling, KTH/SICS

Stockholm, July 20, 2012

TRITA-ICT-EX-2012:170

Abstract

The various capabilities provided by the Internet have attracted a large amount of Internet-based applications over the last decades. Many services that previously only used other means of communication are now also deployed on the Internet. As the content in the communication becomes richer, the bandwidth required to communicate it increases. In the case of delivering audiovisual content over the Internet, a significant amount of bandwidth is required to send the content to a single recipient, and increases rapidly for each additional recipient. To be able to provide scalable, Internet-based systems for video content delivery, researchers and companies have begun to focus on peer-to-peer-based approaches, meaning participants collaborating and contributing their bandwidth to assure content delivery to all others.

This thesis proposes a design for a peer-to-peer system for delivery of live video, and provides simulation results for an implementation of the design. The design targets some of the current issues of peer-to-peer systems – mainly that of providing friendliness towards Internet Service Providers (ISPs). Peer-to-peer systems generate considerable amounts of traffic, which is often sent between peers located in different ISPs, even when data is available at a peer in the same ISP as the recipient. This creates problems for ISPs as they often have to pay other ISPs for data sent over cross-ISP connections, and because congestion can occur in the ISPs gateways to the rest of the Internet – the problems increasing with the number of ISPs that the traffic has to go through. This has forced some ISPs to limit or block peer-to-peer traffic completely.

The system designed in this thesis uses a gossip-based peer-to-peer protocol for content dissemination, and to minimize cross-ISP traffic, the thesis proposes that peers should choose peers closer in the network topology to connect to. This can be achieved by creating a database composed of ISPs and the distance between them, which is consulted every time a new connection is to be created. The database is small enough to be stored locally at each peer. As long as a peer is able to deliver a clear stream it will only connect to close peers, however should the close peers not be able to provide data at a sufficient rate, the peer will request random peers in the system to also provide it with data.

Evaluation of the system in various simulation scenarios shows that it operates well in a constrained environment as well as during peer failures. The evaluation also shows that it is possible to have high clustering of peers and still deliver a clear stream to all of them, as long as a few random connections are allowed to be created when close neighbors can't provide a sufficient download rate. Comparing the use of biased neighbor selection to random selection, traffic between ISPs is efficiently reduced in the overall system, the portion of traffic exchanged between peers in the same ISP experience a ten times increase or more in most scenarios, and in larger ISPs that contain many peers, 75% of all traffic is exchanged between peers within the ISP. Thus the design presented in this thesis can be recommended to developers and content providers that are looking to increase ISP friendliness in their existing or future peer-to-peer applications.

Sammanfattning

De många möjligheter som Internet medför har gett upphov till ett stort antal Internet-baserade tjänster de senaste decennierna. Flertalet tjänster som tidigare endast fanns tillgängliga via andra kommunikationskanaler är numera också möjliga att nå via Internet. I och med att informationen som skall skickas via olika tjänster ökar så ökar också kraven på bandbredd, och för att kunna leverera video krävs en ansenlig bandbredd. Bandbredden som krävs av den som levererar videon ökar dessutom snabbt i takt med antalet mottagare av den. För att kunna skapa Internet-baserade tjänster för video som tillåter ett större antal användare har forskare och företag börjat använda peer- to-peer-baserade lösningar, där användarna av tjänsten samarbetar och bidrar med deras egen bandbredd för att göra det möjligt för samtliga användare att ta emot videon.

Detta arbete beskriver hur ett peer-to-peer-system för live-video kan utformas, och tillhandahåller resultat från simulationer med en implementation av systemet. Det huvudsakliga målet med systemets design är att minska bördan som peer-to-peer-system vanligen utgör för internetleverantörer. Denna typ av system genererar ofta stora mängder data, som i de flesta fall skickas mellan användare som befinner sig i nätverk tillhörande olika internetleverantörer, trots att samma data ofta finns tillgänglig på närmare håll – det vill säga hos användare som tillhör samma internetleverantör som mottagaren. Detta är ett problem för internetleverantörerna eftersom de ofta behöver betala för trafik som lämnar eller går till deras nätverk. Utöver detta så kan de anslutningar som existerar mellan dessa nätverk inte alltid klara sådana mängder trafik, vilket gör att alla användare av de anslutningarna blir lidande. Detta har lett till att vissa internetleverantörer begränsar eller inte tillåter peer-to-peer-trafik överhuvudtaget.

Systemet som utformats i detta arbete bygger på gossiping (ryktesspridning) för att förse användare med videoströmmen. För att minimera mängden trafik som skickas mellan användare hos olika internetleverantörer så jämför användarna avståndet i nätverkstopologin till andra användare och skapar bara anslutningar till de som befinner sig närmast. Avståndet mellan två internetleverantörer utgörs av antalet anslutningar mellan två internetleverantörers nätverk som måste passeras på vägen. Dessa avstånd är lagrade i en databas som finns lokalt hos varje användare. Så länge en användare kan se en videoström utan störningar så laddas den endast ner från närbelägna användare, men skulle dessa inte kunna tillgodose användaren med tillräcklig datahastighet så kommer andra, slumpmässigt utvalda, också att kontaktas för att bidra med delar av videoströmmen till denna användare.

I den utvärdering av systemet som gjorts så har det visat sig fungera väl när tillgängliga resurser som bandbredd och tillförlitligheten i det fysiska nätverket är begränsade, samt under svåra förhållanden som när en stor del av användarna lämnar systemet samtidigt. Utvärderingen visar också att användarupplevelsen inte påverkas av förändringen som det innebär att föredra kommunikation med närbelägna användare, så länge några få slumpmässiga anslutningar är tillåtna i de fall då användare i närheten

inte kan tillhandahålla tillräcklig datahastighet. Jämfört med att välja alla kommunikationspartners slumpmässigt så minskar tillvägagångssättet i detta arbete effektivt trafiken mellan olika internetleverantörer både sett till hela systemet och i enskilda nätverk—andelen trafik som utbyts mellan användare tillhörande samma internetleverantör blir i de flesta fall tio gånger större—och i stora nätverk som innehåller många användare så utgör trafik mellan dess användare 75 % av all systemets trafik i detta nätverk. Den design som utformats i detta arbete kan därför rekommenderas till utvecklare och tillhandahållare av både existerande och framtida tjänster som använder peer-to-peer-teknik, och som är intresserade av att minska belastningen som deras system utgör för internetleverantörer.

Keywords: gossip, isp friendly, live streaming, peer-to-peer, p2p, video

Contents

1 Introduction

1.1 Acknowledgements

I would like to thank my supervisor, Dr. Jim Dowling, for the opportunity to work on this thesis, and for his guidance throughout the work.

I am grateful to the students and researchers of the CSL group at SICS for all their help, and also to everyone that has provided feedback on my work.

Most of all, I want to thank my family for their support and for always having faith in me.

1.2 Background

The wide deployment and good extensibility support of the Internet and the TCP/IP model has attracted a large amount of applications and services over the last decades. Media that traditionally belonged to the category of analog telecommunications, such as radio, telephony and TV broadcasts, are becoming digital through deployment over the Internet. This type of media content requires significantly more resources compared to the text-based content that were dominating the Internet in its youth; the resources required to deliver video content—a movie, a seminar, a sport event, or similar—over the Internet increases quickly with the number of clients that are receiving the content, since the supplier has to send an individual copy of the content to each of its clients. The main constraint of such systems is therefor the upload bandwidth capacity of the server. As a result, it is impossible to deliver video content of good quality using a server with an ADSL connection, even to a small number of clients.

A popular solution used to lower the upload bandwidth requirements for a server, in this case a media server, is the peer-to-peer architecture. This approach leverages the upload bandwidth of all participants in the media content distribution, by equipping receivers of the content with the same capabilities as the initial sender. The term peer refers to the participants in the system all having the same role; every participant is in a sense both a client and a server. As soon as a peer has received some content it acts as a server for peers that have not yet received the same content. This solution is most widely used in file sharing applications such as BitTorrent.

Using the peer-to-peer approach other issues arise, as even though the peers in a system are equal in their interface, they are very much heterogeneous in terms of bandwidth, reachability, geographical and network location, and more. To further increase the complexity of designing a peer-to-peer system, peers are often joining and leaving the system at a varying rate, risking system quality to decrease if appropriate action is not taken when such events occur.

1.3 Objective

This thesis focuses on current topics in peer-to-peer-based live video streaming research. The objective is to design a peer-to-peer system for large-scale live

video streaming that

- maximizes throughput and stream quality, provides resistance to churn and message loss, while minimizing overhead
- is friendly towards Internet Service Providers (ISPs)
- is able to traverse most Network Address Translator (NAT) types
- provides easy-access monitoring through common interfaces

and the main contributions of the thesis being the development of a network-biased Peer Sampling Service (PSS), and then, partly by using this new PSS, enhancing an existing protocol for live video streaming with ISP friendliness. NAT traversal is not a contribution of this thesis, but a property of the system developed which is gained through already existing tools in the framework used.

1.4 Limitation

The system designed in this thesis will not

- provide any decentralized protection against freeriders

1.5 Motivation

The main benefit of using a peer-to-peer system for live content delivery, compared to a client-server solution, is the lower requirements on the provider, primarily of the upload bandwidth capacity. These lowered requirements provide a large amount of Internet users with the possibility to stream content to almost any number of receivers, provided a scalable system. Instead of having to pay for gigabit links and advanced server solutions an upload rate of a few hundred kilobytes to a few megabytes, depending on video bitrate and quality, is sufficient. This can heavily reduce the cost for companies interested in sending live video over the Internet, and is enough to allow video streaming even from residential connections.[27, 18]

The motivation for ISP friendliness of peer-to-peer systems is to decrease the cost for ISPs. Peer-to-peer systems generate at least as much traffic as those that use the client-server model to provide the same service, but while the latter often uses dedicated links that are designed and paid for with the single purpose of the system in mind, most peer-to-peer systems use multi-purpose consumer links and connect users without respecting the underlying network design. For example, over 70% of content present in nearby peers showed to be downloaded from distant peers in a study of the popular peer-to-peer file-sharing system BitTorrent[14]. This ignorance of peers' location is costly for ISPs for two types of reasons;

i) infrastructure constraints: long-distance traffic consumes more network capacity and causes congestion at gateways between ISPs[27].

ii) business decisions: the Internet is mainly formed by ISPs having provider-subscriber agreements with eachother, and thus an ISP often has to pay for incoming and outgoing traffic[13, 27].

In the end this affects all involved parties—connection providers (ISPs), content providers, and content recipients—since due to this increased cost ISPs have begun to block or limit peer-to-peer traffic[27], resulting in decreased quality in affected systems. The service (delivery of content) thus becomes less appealing to users, causing the provider to lose viewers, customers, fans, and advertisement possibilities. Several peer-to-peer video streaming systems that attempt to achieve ISP friendliness already exist, some of them with millions of users, but most of them do so inefficiently.[5, 27]

Monitoring and diagnosing of peer-to-peer systems have become more important as the amount of services that use peer-to-peer based solutions, and the traffic they generate, increases. It is of interest to service providers and consumers to know which system that has the most suitable properties for their needs. System operators must be able to monitor system health and clients' performance in real-time, and also in aggregation over time. Finally, researchers may want to gain further knowledge about systems' behavior by analyzing them in simulations and experiments.[27]

To summarize, this thesis is motivated by current issues and developments in peer-to-peer systems, particularly ISP friendliness and live video streaming. With this in mind, the thesis aims at finding possible solutions and new designs and implementations.

1.6 Outline

This thesis is structured as follows:

Section 2 provides a background to the thesis work, focusing on core concepts of peer-to-peer live streaming and ISP friendliness.

Section 3 presents related work; research on network awareness and ISP friendliness, and evaluations of the ISP friendliness provided by current, widely used peer-to-peer live streaming systems.

Sections 4 and 5 describes the system design and implementation, respectively.

Section 6 provides an evaluation of the system.

Finally, Sections 7 and 8 give a conclusion and outline future work.

2 Core Concepts and Current Issues

2.1 Peer-to-Peer Live Video Streaming

Live streaming in peer-to-peer systems is the task of broadcasting data from a single source to a large number of clients, the data being produced at system runtime and the size of it *unbounded*, meaning that its size is not known until the end of the data dissemination. Most, if not all, live video streaming falls into the category of *high-bandwidth content dissemination*, in which the disseminated data represents a significant amount of the available bandwidth among dissemination participants.[18]

For the clients to be able to experience a smooth playback, it is important that data is received within certain timing constraints. In addition to the above, the notion of live means that there should be little delay between the initial generation of data at the source and the playback at each client. Therefor a client will quickly notice if the download rate drops below the *stream rate* – the rate at which data is produced at the source. Loss of more than 1% of the streamed data can be shown to have a large negative effect on user experience[3].

2.1.1 Comparison to File Sharing

Compared to live video streaming, file sharing is the task of *copying* data that is static and thus *bounded* in size. In peer-to-peer systems, file-sharing protocols like BitTorrent can use the fact that all data is available at the start of dissemination to effectively maximize use of the total upload capacity of the system, using the *rarest first* principle: by having the source send out different parts of the file to different peers, the peers can then exchange those parts with each other, effectively offloading the source in that task and instead sending out parts that were not sent yet. By extension, this means that pieces that are rare in the system will be sent first.[15]

Live video streaming cannot use this principle as content is produced on the go, meaning that the source can only disseminate data in a certain order, and as peers should deliver smooth, near-live playback of the video, they are all interested in roughly the same part of the content at the same time. Therefor peers have to download the stream at an average speed equal to that of the stream rate, with very little variance allowed. The nature of content dissemination in live streaming also makes the successful incentive mechanism *tit-for-tat[23]* used in file-sharing protocols troublesome to implement in a live streaming system. In tit-for-tat a peer is encouraged to upload data to peers which it wants to download from, otherwise risking to be denied the download in favor for other peers which offer more upload. When all peers want the same data and no rarest first policy is applicable, the peers have no leverage on this kind of market.

Table 1 contains a summary of the comparison.

Table 1: Comparison between live streaming systems and file sharing

	Live	*File sharing*
Content	dynamic, unbounded	static, bounded
Download pattern	linear	rarest first
Content access	continuous	at dissemination end
Bandwidth sensitivity	high	none

2.1.2 Comparison to Video on Demand

Video on Demand (VoD) can be described as the task of file sharing, where clients want to start the download at a certain position, sequentially download parts from that position, and start accessing the content as soon as possible. It differs form live video streaming in that considerable buffering may be allowed[1] as there is no requirement on the content being live. As stated, clients in a VoD system may choose to start the download at an arbitrary position in the video, resulting in two characteristic challenges in the development and deployment of distributed VoD systems: a client has to store some parts of the downloaded content to be able to serve other clients which are viewing the same video but not at the same position, and the system has to effectively support clients in the lookup of such content at other clients, preferably in a distributed manner.[27]

A summary of this comparison is available in Table 2.

Table 2: Comparison between live streaming and Video on Demand (VoD)

	Live	*VoD*
Starting position	same	arbitrary
Content	dynamic, unbounded	static, bounded
Download pattern	linear	linear
Content access	continuous	continous
Bandwidth sensitivity	high	low/high

[1] Users still probably want to view the video as soon as possible, but it is not a functional requirement of the system as it is in the case of live video streaming.

2.1.3 Challenges in Live Video Streaming

The following are considerable challenges in live streaming systems formulated by Monod [18]:

- *Minimize the overhead of the protocol*: design the system so that the average upload of the clients is as close to the stream rate as possible
- *Maximize the stream quality*: provide clients with a stream that is as close to the original stream produced by the source as possible
- *Minimize the buffering delay*: let clients start watching the stream as soon as possible after they start receiving data
- *Minimize the stream lag*[2]: provide a stream that is as live as possible to all clients
- *Maximize simplicity*: aim to provide simple protocols to ease implementation and deployment over large-scale systems

It is apparent that the challenges in live video streaming, and one of the reasons why they are difficult to solve, are in conflict with each other. For example, to provide constant high stream quality it may be necessary to use a significant buffer size, which causes high stream lag and long startup delay.[20]

2.2 System View and Overlay Classifications

An important task in getting a peer-to-peer system to scale with the number of participants in the system is keeping all peers connected. A single peer can't keep track of all peers currently in the system, considering most systems having a non-negligible amount of *churn*—nodes joining and leaving for valid (application- or user-imposed), or invalid (crash, failures) reasons—why the task of monitoring them all would become too costly in large systems, both in terms of memory resources and communication overhead. Instead each peer has its own *view* of the system; a subset of the system's peers. Which peers a certain peer includes in its view depends on the *overlay* used.[18]

2.2.1 Structured and Unstructured Overlays

In general, an overlay is a network that is built on top of another network. Peer-to-peer overlays, from here on referred to as just overlays, are built on top of the Internet. These are commonly categorized as being structured, unstructured, or a hybrid of the two. In structured overlays the views of the peers follow certain rules, and connections between nodes follow certain semantics, in other words they have a certain meaning. Because connections are created according to these rules, structured overlays typically only change in case of churn, i.e. they are *static* and *reactive.*

[2]Stream lag is defined as the time difference between the moment at which the stream is sent from the source and the moment at which it is played on the client.

Unstructured overlays can be either static or *dynamic*. The views and connections in unstructured overlays are fairly random and there is no hierarchical relationship between connected peers. Systems that use a static unstructured overlay are referred to as *mesh-based*. Peers in dynamic unstructured overlays update their views periodically, making them *proactive* to churn. Gossip-based overlays falls into this category, and will be discussed further in Section 4 . [18]

Table 3 summarizes the classification of overlay types.

Table 3: Classification of overlay types

	Static, reactive	*Dynamic, proactive*
Structured	DHT-based, Ring, Trees, Multitrees	
Unstructured	Mesh-based	Gossip-based

A visualization of some overlay structures is available in Figure 1, which in addition to the basic structures contains some structures that has interesting properties related to Internet applications; *scale-free* and *small-world*. The distribution of connections in a scale-free overlay follows the power law—the probability that any node is connected to k other nodes is proportional to k^{-n}, where n typically is between 2 and 3—meaning few nodes have many connections, and many nodes have few connections. The small-world structure depicts an overlay where nodes mainly are connected in clusters, each node with a number of connections significantly smaller than the number of nodes in the system, but can reach any other node with just a few hops, thanks to a small number of random, longer links. Figure 1 visualizes scale-free and small-world topologies with the purpose of showing their characteristics, however when the number of connections each node has can be chosen freely, systems often incorporate the properties of both of them. Examples of such systems are the hyperlinks composing the World Wide Web, router connections in the Internet, and the collaborations between scientists in research papers, all of them exhibiting clustering but with some random links, and the connections in them following power-law distributions.[30]

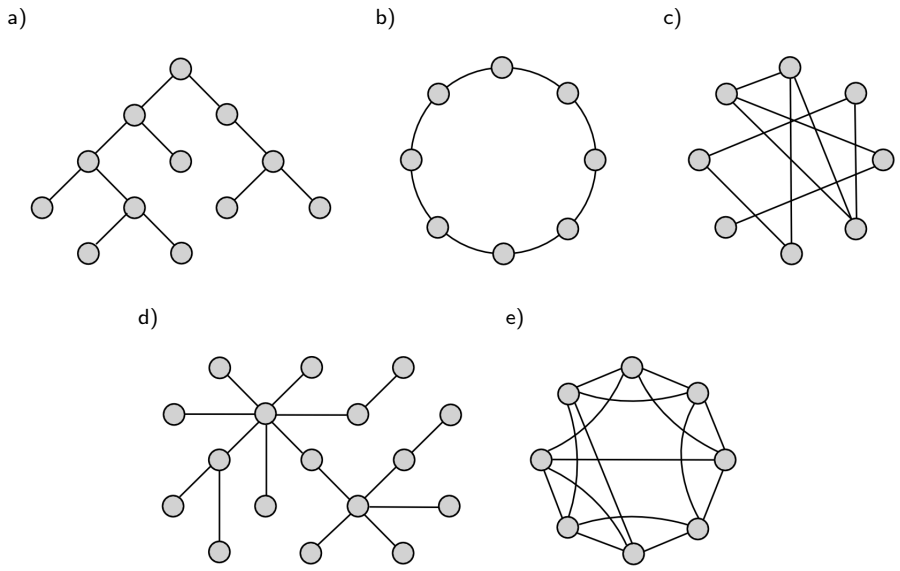

Figure 1: *Visualization of overlay structures. a) shows a tree structure where most nodes have one parent and two children, b) a ring structure where all nodes have exactly two neighbors, c) a random structure (unstructured overlay), d) a scale-free structure with minimum of one connection, and e) a small-world structure with four connections per node.*

2.2.2 Overlays and Content Dissemination

There are mainly two types of content dissemination used in peer-to-peer live streaming systems; either data is sent directly to other peers using a *push model*, or sent on request from other peers in a *pull model*. The former is commonly used when peers are connected in a *structured tree overlay*, the basic design being a peer having a single parent and a fixed number of children. The source is at the top of the tree and is the only node that doesn't have a parent. The video stream is propagated down the tree by having parents replicating data to their children. In this setting there is little overhead after connection establishment between child and parent, and the stream lag is strictly bounded by the path between source and most distant child in the overlay. Disadvantages are the high complexity of maintaining a stable tree in presence of churn and the large portion of nodes without children that are not contributing any upload bandwidth.

When peers are unstructured it is common to do pull-based dissemination of media. Peers send information about which data they have, and other peers can then request that data. Content dissemination using unstructured overlays is in comparison to structured overlays more resilient to churn, thanks to the proactiveness and elimination of hierarchical roles; from a design perspective no node is more important than another to the system, in contrast to tree struc-

tures where parents higher up in the tree have more children depending on them to provide data. In addition, thanks to the random nature and lack of rules for neighbor connections, the maintenance complexity of unstructured overlays is low, however for the same reasons content dissemination paths become sub-optimal and additional communication is required to coordinate what content should be sent between peers. Moreover, when relying on randomness there is no bound on the delay of the dissemination, as the path for delivery to a certain peer can be arbitrarily long.[27]

2.3 Need for ISP Friendliness

The Internet is composed of several interconnected domains, called Autonomous Systems (ASes), which are owned by Internet Service Providers (ISPs). The ASes are connected through gateways, and the topology in which they are connected is hierarchical, and not for technical or performance reasons such as the tree structure for content dissemination, but for business reasons – the hierarchy is formed from commercial contractual relationships between ISPs[12]. Thus an AS and its connections can be thought of as the technical instance of an ISPs commercial contracts.

There are mainly three types of ISPs; those that provide connectivity to other ISPs, those that provide connectivity to home users and smaller companies, and those who do both. In the AS infrastructure, the most typical relationship between two ASes is the *customer-to-provider* relationship, wherein the provided service is connectivity to the rest of the Internet, and the customer pays its provider for any traffic sent between the two. Other relationships are the *peer-to-peer* relationship (which has nothing to do with peer-to-peer systems), where two ASes agree to exchange traffic between their customers free of charge for each other, and the *sibling-to-sibling* relationship, which depicts two ASes being under the same administration.[8] An AS that provides connectivity for other ASes, either their provider or their sibling, is said to *transit* their traffic. An AS that is only connected to its providers, i.e. does not transit any traffic, is called a *stub AS*. Figure 2 shows an example of a possible AS topology.

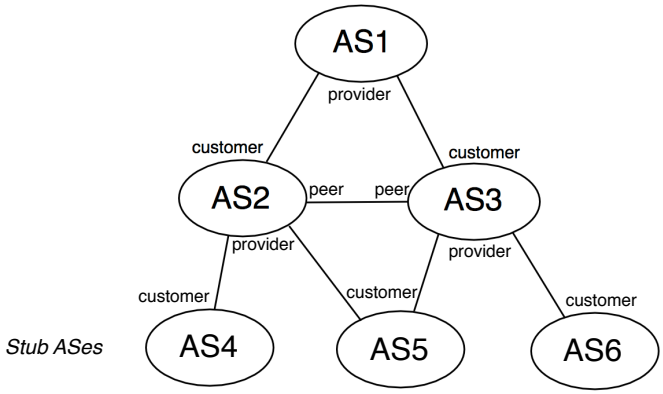

Figure 2: *Example of an AS topology and the relationships between ASes.*

Returning to peer-to-peer applications, they are known to generate large amounts of traffic—according to recent studies at least 50% of all traffic on the Internet[25, 22]—and most of the time they are unaware of the underlying network topology; despite data being available in topologically close peers it is downloaded from peers far away. A study on this issue, monitoring peer behavior in BitTorrent, shows that its peers does so for 70% of closely available data[14], and perhaps this shouldn't come as a surprise since BitTorrent organizes peers in an unstructured overlay. Studies on peer-to-peer live streaming systems using unstructured overlays show that the same ratio applies in those[5]. Still, the problem is present in systems using structured overlays too; many of these systems are not structured with respect to the network topology.[27, 22, 5] Traffic that passes between at least two ASes is referred to as *inter-AS traffic* or *cross-ISP traffic*.

This results in heavily increased costs for customer ASes since they have to pay their providers for all this long-distance traffic, most of it unnecessary. Another problem that applies to all types of AS connections is the risk of congestion and delays at the ASes gateways, which may harm the overall Internet experience for everyone using those gateways. The adverse effects of inter-AS traffic on finances and operability of ISPs have caused some of them to limit, shape, or completely block peer-to-peer traffic.[13, 27]

For a non-hostile networking environment, peer-to-peer applications have to respect the desires among all parties involved in the exchange of data: users, content providers, and service providers. This adds a sixth challenge for peer-to-peer live streaming systems in addition to the ones mentioned in Section 2.1.3:

- *Minimize inter-AS traffic*: design the system so that communication be-

tween peers as close as possible in the network topology is preferred.

Finally, it is worth mentioning that ISPs without any providers, *tier 1* ISPs, with globally spanning networks, prefer more inter-AS traffic as their customer ASes have to pay them for the transit traffic, while the tier 1 ISP itself does not pay any provider.[21].

2.4 The NAT Problem

The existence of peers behind firewalls and NATs—*private* or *guarded* peers—and the issues which arise in their presence have been frequently discussed in peer-to-peer system research[18, 20, 7]. Firewalls and NATs are usually present between a single or a group of end-hosts and the Internet. Firewalls are used to filter unwanted traffic, possibly in both directions, while NATs are used to allow a group of end-hosts with private addresses to share a single public address, and thereby allowing communication with other hosts on the Internet using various translation techniques, thereby the name *Network Address Translator*. Due to increased security requirements and the shortage of public IPv4 addresses the use of these devices is rapidly increasing. [7]

Neither of these technologies is any hindrance to peer-to-peer systems by design; a private peer behind a correctly implemented and configured firewall or NAT will have the same capabilities as a public peer. The issues arise because i) most users do not have the knowledge to do the necessary configurations, and ii) in the many NATs that are implementing more cumbersome policies, each connection has to be configured separately by the user, an impossible task in peer-to-peer systems that often create new connections frequently. Therefor these peers may be hard to reach or not be reachable at all, limiting or preventing their collaboration with other peers in a system. [7]

To tackle these issues, the Internet Engineering Task Force (IETF) promotes a set of requirements and rules that these devices should follow in order to ease the communication with private peers in peer-to-peer systems. As the classifications of NAT types regarding policy and filtering for UDP traffic shows (Table 4), one can see that there is a lack of standardization, i.e. agreement among vendors, for NAT operation. NATs have to implement three different kinds of policies: *port mapping*, that describes whether to create a new private peer to port mapping or to use and old, *port assignment*, describing which port to use when a new mapping has to be created, and *port filtering*, which describes how to filter incoming packets depending on their source[20].

Table 4: NAT policies

Policy category	*Policy*	*Description*
Port mapping *rules for when to reuse port mappings*	Endpoint-independent	The same port on the NAT is reused for all outgoing messages from the same private peer to any destination.
	Host-dependent	The same port on the NAT is reused for all outgoing messages from the same private peer to any port on a certain host.
	Port-dependent	The same port on the NAT is reused for all outgoing messages from the same private peer to a certain IP address-port pair.
Port assignment *rules for how to create new mappings*	Port-preservation	The same port used by the private peer is used on the public interface of the NAT.
	Port-contiguity	The NAT maps ports according to an internal value, which is incremented for each new mapping.
	Random	The NAT uses a random port on the public interface for each new mapping.
Port filtering *rules for which incoming packets to forward to private nodes*	Endpoint-independent	All incoming packets on a certain port is forwarded to the private peer's port mapped to that public port, i.e. the peer has previously sent a message, using this mapping, to any host.
	Host-dependent	A packet is only forwarded if the mapped private peer has previously sent a packet with destination address being the same as the source address for the incoming packet.
	Port-dependent	A packet is only forwarded if the mapped private peer has previously sent a packet with destination address *and* port being the same as the source address and port for the incoming packet.

The combination of these policies present in a NAT tells how difficult it will be to connect to a peer behind it, i.e. *traverse* the NAT. For some combinations simple heuristics suffice, for others assistance from a server is required, and some are not possible to traverse under any circumstances.[24] To greatly simplify the process of NAT traversal, the Internet Gateway Device (IGD) protocol[11] was devised to allow an application running on a private host to automatically create port mappings in their NAT. As IGP is implemented via Universal Plug and Play (UPnP), devices supporting this mechanism is commonly referred to as *UPnP devices*, and peers behind them *UPnP peers*. With applications utilizing UPnP capabilities when present, UPnP peers essentially become public peers.

Measurements have shown that only about 20-30% of participants in peer-to-peer systems belong to the categories public or UPnP[16, 32]. As a small comfort, and fortunately for the well-being of many currently deployed systems, these peers make up a significantly larger portion of the total upload bandwidth in the system[32]. Many peer-to-peer systems fail to account for the presence of these devices[7], and for these and future systems to gain wider deployment and provide better user experience, given today's status of the global networking environment, they will have to start implementing NAT traversing capabilities.

3 Related Work

In this section important research that discusses or attempts to solve the current issues of peer-to-peer live streaming mentioned in this thesis is presented. Focus lies on research that has influenced the system design. First there is a summary of some existing peer-to-peer live streaming systems that are widely deployed and their achievements to be ISP friendly, then recent research targeted at providing tools or solutions for increased ISP friendliness in peer-to-peer applications is presented.

3.1 Studies on Peer-to-Peer Live Streaming Systems

Measurement and monitoring studies on peer-to-peer live streaming systems are mainly focused on those that have gained a large user base, some of them with millions of users, considering the effects on the network increases with the number of clients running the system.

Ali et al. provide measurements on SopCast and PPLive from 2005. Their conclusions are that the two applications incorporate none or little network awareness. The study also analyses how the two protocols behaves in the presence of private peers (behind NAT) and see that these peers can receive data but do not contribute any data back to the system.[1]

A study by Ciullo et al. with measurements from 2008 examines network awareness in the popular live streaming systems PPLive, SopCast and TVAnts. The study also measures peers' transmission rates, number of contacted peers, and contribution rates among contacted peers. All three applications were released in the period 2004–2005 and have since gained much popularity. They are all proprietary and closed, meaning that they are not offering any insights or monitoring of their behavior. Therefor the study deploys about 40 peers in different locations and monitors the communication between them while they are watching the same stream. The study concludes that these systems are not particularly network aware. When content is available in the same AS, it is still mainly downloaded from outside the AS; this is the case for 68% of such data in TVAnts, 87% in PPLive and 96% for SopCast. Some of these results can be explained by looking at the neighbors of each peer, and which of those neighbors that were chosen to download data from. As the results hint, SopCast shows no preference for closer peers, and TVAnts and PPlive show some. The measurement also shows that none of the systems exhibit any subnet or router hop count awareness other than that inferred by the AS awareness. Finally, it is pointed out that there is no sign of any of the systems implementing an incentive mechanism.[5]

One study made by Wu et al. explores topological properties of UUSee, using traces from 2006 provided by UUSee Inc., focusing on ISPs in China. According to their analysis UUSee does not take ISP membership into consideration when doing peer selection (this is both the case when retrieving peers from a tracker server used in UUSee, and when exchanging peers with neighbors). As other systems do, it does achieve some clustering of nodes with respect to ISPs since

connections with high bandwidth and low delay are preferred, which is more likely if the connection is kept within one AS.[31]

3.2 ISP Friendliness

There have been many different suggestions on how to achieve ISP friendliness, and the various research on the topic is mostly divided into two categories: what data that is most suitable to use for overlay construction—i.e. which model of the network to use—and how should this data be used. These topics will therefor be discussed in two separate sections.

3.2.1 Modeling the Network

A system that honors the network underlay and infrastructure can either use an AS-based model[13, 22], respecting business decisions and policies, or a metric-based, the metric being delay or network coordinates, which estimates the network design and thus the ISPs' infrastructure, but not necessarily their policies.

Hsu and Hefeeda [13] proposes in detail a way of minimizing inter-ISP traffic by using AS hop count, and how to construct a storage of hop counts between ASes that is small enough to store in each peer, thereby giving a solution that eliminates the need for any central online storage or ISP collaboration. The storage is constructed by using public BGP data, as that provided by Route Views[3]. The article further suggests two algorithms to minimize the cost on ISPs: `ISPF` and `ISPF-Lite`. The two algorithms differ in how they break ties when multiple AS pairs have the same hop count, which is often the case. In addition to AS hop count, `ISPF` uses geolocation to determine the distance between peers, while `ISPF-Lite` compares the length of the shared IP prefixes of the peers. The results show that `ISPF` and `ISPF-Lite` are great improvements over random peer selection such as in BitTorrent, and that doing peer selection by only using shared IP prefix provides significant reduction of inter-ISP traffic at very little cost. Another important part of the results is that they show that an algorithm sorting peers on ISP distance is much better than just separating same-AS peers and outside-AS peers, i.e. to really minimize inter-ISP traffic it is important to know the distance to neighbor peers, not just whether they are inside our outside the own AS.[13]

A metric-based model typically uses measurements in some way: round trip times (RTT), network coordinates, or DNS redirects.

Choffnes et al. provide a thorough empirical study of the effectiveness of metric-based network models in peer-to-peer systems[4]. Approaches studied are network positioning systems using Vivaldi, direct measurement using Meridian (a lookup framework for close nodes using a ring-structure), and relative positioning using the infrastructures of Content Delivery Networks (CDNs). The network positioning approach usually measures round trip times (RTT), either to a set of neighbors or some designated nodes called *landmarks*, and uses that data to map itself to some geometric coordinates. Vivaldi is the most popular

[3]University of Oregon Route Views Project . http://www.routeviews.org/

network positioning system, and is often called a network *coordinate* system. Any node that has mapped itself to these coordinates can then find out its distance to other mapped nodes by comparing their coordinates to get an approximation, without the need for further RTT measurements (doing end-to-end delay measurements between nodes on demand has been proven to be too costly, too time-consuming and does not scale with system size[9]). The CDN-based approach, CDN-based Relative Positioning (CRP), leverages the existing CDN infrastructures which uses DNS to redirect clients to their closest (low-latency) server; peers with similar DNS redirection in a CDN infrastructure are assumed to be close to each other. This approach may also reflect network design choices of ISPs. Their conclusion based on the results is that existing network positioning systems not only exhibit large errors in predictions, but those errors significantly impact application performance in large-scale peer-to-peer environments. Meridian and CRP achieve relatively good performance; on average the systems locate close nodes most of the time, with CRP being the most accurate.[4]

Another evaluation of the network coordinate system Vivaldi by Steiner and Biersack [28], that also compares both version 1 and 2 of Vivaldi, comes to the same conclusion: Vivaldi coordinates are not suitable for selecting close-by peers, neither with respect to geographical location, nor in the network topology. However, in the defense of Vivaldi's potential, it is pointed out that Vivaldi coordinates can be very useful for estimating RTTs.

On Inferring AS Relationships It should be mentioned that most AS-aware models use several algorithms and heuristics to infer AS relationships into the model. In peer-to-peer systems research, the reasoning behind this is that peer-to-peer and sibling-to-sibling links are cost-free should be preferred to customer-to-provider links. However, recent research[8] has shown that most peer-to-peer relationships (60%) are not known to any other ASes than those part of the relationship (and neither should they be from a routing perspective). Peer-to-peer relationships are also the hardest to guess, 80% are found using recently provided heuristics. Sibling-to-sibling relationships make up a small part of all relationships – about 1%. It may not be bad to use relationships in the model, however it will increase the storage cost to include them, for little gain in ISP friendliness. As implementation of ISP friendliness can heavily affect the traffic in the network, the data for doing so should be as reliable as possible.

3.2.2 Leveraging the Network Model to Achieve ISP Friendliness

Two types of approaches have been proposed to minimize cross-ISP traffic in a peer-to-peer live streaming system given known distances to other peers. One is focusing on *chunk* or *piece scheduling*, which means changing the dissemination protocol, and the other on *neighbor selection*, meaning structuring the overlay so that it reflects the underlay network.

Picconi and Massoulie suggest a model belonging to the former category, in which peers keep two sets of neighbors; one set containing close peers and

the other containing random peers. During normal operation, when the peer receiving a good stream rate, only close peers are used as sources. If chunks are not received according to a certain rate an *early starvation signal* (ESS) is generated, triggering an increase in requests to the random peers. Evaluations show that this approach dramatically reduces inter-ISP traffic and provides good reactivity to churn.[22]

Another model for chunk scheduling by Magharei et al. called OLIVES[17] uses a two-tier overlay-aware block scheduling scheme. In each ISP, there exist external peers and internal peers – external peers, or edge peers, establish connections to external peers in other ASes. The external peers are then responsible for disseminating the stream to the internal peers. A local tracker within each ISP decides which peers that should be external peers and elects a new one when any of them leaves, and a session level tracker helps external peers in discovering each other. OLIVES uses shortest-path scheduling for scheduling chunks both between ISPs and in them. The stream is divided into substreams, and each block contains information about how many peers it has passed (a hop count). According to shortest-path scheduling each peer then pulls a certain substream from the peer that advertises it with the lowest hop count, effectively creating *content delivery trees* with minimum depth.

Focusing on overlay topology construction, Shen and Zimmermann [26] propose a network biased, adaptive peer selection algorithm where peers exchange information about their neighbors and learn about new ones through gossiping. They specify the main problems that can affect a peer's streaming rate negatively when using pure biased peer selection, i.e. selection only based on the underlying network: (1) local clustering peers may suffer from congestion due to the heavy local traffic; (2) local neighbors of a peer may provide a lower uplink bandwidth while farther participants could potentially provide a higher uplink bandwidth; (3) a peer and its local neighbors may have similar data availability so that the peer can obtain little streaming data that does not already exist in its data buffer. To avoid these problems the algorithm uses some *probability distribution function* (PDF) with a tunable parameter α (for example the geometric PDF $(1-\alpha)^{k-1}\alpha$) that affects which peers are chosen as neighbors. Before each gossip round, a peer orders its neighbor set of size n according to topological closeness by using an *Oracle*, a service provided by the ISP, and uses the PDF to choose a gossip partner. Through tuning of α, either close or distant peers can have a higher probability of being chosen. After the peers have exchanged their neighbor sets (or a subset of it) the PDF is used a second time to filter which n neighbors to retain in the set. An adaptive algorithm is achieved by storing neighbors' streaming contributions between two gossip rounds, and before choosing gossip partners the subset of close neighbors is compared to the subset of distant peers with respect to their contribution to decide if there can be an increase in biasing, or if a decrease is necessary. How large each of these subsets is is decided by the PDF each round.

4 System Design

The following section explains the design choices made to achieve the thesis' objective. First an overview of the system is presented, then the video dissemination protocol *Gossip++* is introduced, and finally this thesis' suggestion on how to achieve ISP friendliness is described.

4.1 Overview

The system uses a gossip-based pull model for content dissemination together with a local database of AS distances, and a network-biased peer sampling service to provide ISP friendliness. The system is mainly built upon Gossip++, a gossip-based protocol for live streaming by Monod [18], for content dissemination, the `ISPF-Lite` algorithm provided by Hsu and Hefeeda [13] for constructing the locality database, and the neighbor selection approach by Shen and Zimmermann [26] to introduce a mechanism for reducing inter-AS traffic while keeping a good stream rate.

4.2 Content Dissemination Protocol: Gossip++

This section, including the protocol used for content dissemination, is based on the thesis Monod [18], which introduces Gossip++, a gossip-based dissemination protocol for live streaming in large-scale systems. Being a gossip-based protocol, it has the advantage of being simple to its structure and handles churn well thanks to its random, proactive nature. First, a background to gossip for high-bandwidth content dissemination is presented, then follows an outline of the core gossip protocol used in Gossip++, and improvements to this protocol, also depicted in [18].

Gossip-based protocols execute in rounds, commonly involving two phases: *(i)* a *communication phase* where a node choses a subset of its neighbors to exchange some information with, specified by the application, and *(ii)* a *processing phase*, where the node applies a state transition function on its current state and the data received during the last communication phase. When broadcasting, e.g. gossiping some content, the processing phase will evaluate if new content was received and what information should be gossiped in the next round.[18]

A consequence of disseminating data using gossip is that many nodes may receive duplicate messages due to the randomness of gossiping; as neighbors are chosen randomly when gossiping, there is always a probability that some node does not receive a certain message, and to make this probability small enough (according to system requirements) there is often a need for some redundancy of data, thus some nodes will receive the same information multiple times. When the information to disseminate becomes large, as in media applications, this may have negative impact on system performance. A protocol that addresses this issue is the *three-phase gossip protocol*, inspired by mesh-based protocols: only small messages containing metadata are gossiped as in the above description of

the communication phase. The actual data has to be requested by the recipient of the metadata.

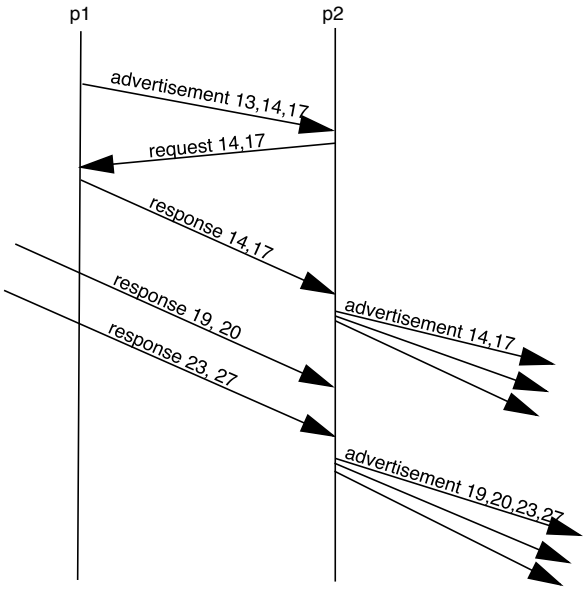

Figure 3: *Three-phase gossip protocol. The first round shows how peer* p2 *receives an advertisement from* p1*, and requests pieces 14 and 17 (it has previously received 13). In the next round, the IDs of the pieces received from* p1 *are advertised to a random subset of* p2*'s neighbors. It is then illustrated how* p2 *receives responses from some other peer(s) (advertisements and requests not shown), and advertises the received pieces' IDs together at the beginning of the following round.*

The typical scenario when doing live streaming using Gossip++ is to have a source node that is fed a stream of video data that the node repackages, or just adds metadata to, and then gossips an advertisement that new pieces are available to a subset of its neighbors. The stream can be produced by VLC[4], an input device, or similar, and be of any chosen quality. When a node receives an advertisement from the source it requests the pieces advertised immediately, and the source responds with a message containing the piece. By choosing a new subset of the neighbors for each new piece, they will each take different dissemination paths. A node that received any pieces since the last gossip round will advertise them to a subset of its neighbors in the coming round. When the protocol is executed between two regular nodes the exchange of messages is the same, except that the nodes then may already have some of the advertised pieces, and thus will not request those. By doing so requesting nodes make

[4]VideoLAN – VLC: Official site . http://www.videolan.org/

sure that they do not receive any duplicate pieces, and thus do not put any unnecessary strain on the advertising node.[18]

The gossiping protocol in Gossip++ follows the *infect-and-die model*—the name inspired by the term *epidemic protocol*, a common nickname for gossiping protocols—meaning that a certain advertisement is sent only once. To be able to gossip all advertisements to all nodes with high probability in this model, research have shown that the fanout, the number of neighbors to gossip with, has to be chosen as $ln(n) + c$, where n is the system size and c a constant that defines the probability of the protocol to result in a connected graph as $e^{-e^{-c}}$.

4.2.1 Core Protocol: Three-Phase Gossip with Retransmission

As informally described above, the phases in three-phase gossip are the following:

- *Advertisement phase*: every gossip period, each node picks and advertises newly received pieces to a set of f other nodes uniformly at random.
- *Request phase*: upon receipt of an advertisement for a set of pieces identifiers, a node evaluates the set and its already received pieces, and requests any pieces that it's missing.
- *Response phase*: as a node receives a request for a set of pieces, it sends a response containing the pieces.

An outline of the core functionality of the protocol is given in Algorithm 1.

Algorithm 1 Three-Phase Gossip with Retransmission

Initialization:
1: $f := ln(n) + c$
2: piecesToAdvertise := piecesDelivered := requestedPieces := $\emptyset$
3: **start**(GossipTimer)

Phase 1 – Gossip piece ids

procedure publish(c) **is**
4: deliverPiece(c)
5: gossip({c.id})

upon (GossipTimer mod gossipPeriod) = 0 **do**
6: gossip(piecesToAdvertise)
7: piecesToAdvertise = $\emptyset$

Phase 2 – Request chunks

upon receive [ADVERTISEMENT, piecesAdvertised] **do**
8: wantedPieces = $\emptyset$
9: **for all** $id \in$ piecesAdvertised **do**
10: **if** (($id \notin$ requestedPieces) **or** (isBeingRetransmitted(id)) **then**
11: wantedPieces := wantedPieces $\cup$ id
12: requestedPieces := requestedPieces $\cup$ wantedPieces
13: **reply** [REQUEST, wantedPieces]
14: **if** (id requested less than r times) **then**
15: **start**(RetTimer(piecesAdvertised))

Phase 3 – Push payload

upon receive [REQUEST, wantedPieces] **do**
16: askedPieces := $\emptyset$
17: **for all** $id \in$ wantedPieces **do**
18: askedPieces := askedPieces $\cup$ getPiece(id)
19: **reply** [RESPONSE, askedPieces]

upon receive [RESPONSE, pieces] **do**
20: **for all** $p \in$ pieces **do**
21: **if** ($p \notin$ piecesDelivered) **then**
22: piecesToAdvertise := piecesToAdvertise $\cup$ $p.id$
23: deliverPiece(p)
24: **cancel**(RetTimer(pieces))

Retransmission

upon (RetTimer(piecesAdvertised) **mod** retPeriod) = 0 **do**
25: **receive** [ADVERTISEMENT, piecesAdvertised]

function isBeingRetransmitted(id) **returns** boolean **is**
26: **return true** if a timer is scheduled with piece id id, **false** otherwise

Miscellaneous

function selectNodes(f) **returns** set of nodes **is**
27: **return** f uniformly random chosen nodes in the set of all nodes

function getPiece(id) **returns** piece **is**
28: **return** the piece corresponding to the id

procedure deliverPiece(p) **is**
29: deliveredPieces := deliveredPieces $\cup$ p
30: **deliver**(p)

procedure gossip(ids) **is**
31: communicationPartners := selectNodes(f)
32: **for all** node $\in$ communicationPartners **do**
33: **send**(node) [ADVERTISEMENT, ids]

By evaluating the key parameters of this algorithm, namely those regarding fanout and proactiveness, Monod [18] concludes that, when gossiping 700 – 2000 kbps between 230 nodes:

- In contrast with theoretical evaluations of fanout values, a too high fanout impacts performance negatively – a fanout value between 7 and 10 is preferred in this setting.
- To minimize stream lag and impact of churn, a new set of nodes should be chosen as recipients for advertisement each round.

4.2.2 Improvements to Three-phase Gossip with Retransmission

Some issues still remain that can affect the performance of the protocol depicted in Algorithm 1. Since a peer receives advertisements only with a high probability, there is a risk of a peer not seeing some advertisements even when no message loss occurs. Moreover, if a response is not received for a certain period after sending a request, the peer has to resend the request, which, if not done with care, may exhaust the advertisers' upload capacity, or create significant communication overhead. To deal with these issues, Monod [18] proposes the two mechanisms *Codec*, an erasure coding scheme, and *Claim*, a retransmission scheme that leverages the duplication of advertisements in the system. In the following paragraphs, these two mechanisms are briefly described.

Codec uses erasure codes, a block-based forward error correction (FEC) mechanism. FEC operates by adding redundant data to the original stream data. Both are then sent together and the added data can then be used by the receiver to check and possibly correct the original data. Consider the usage of FEC at the source node: before dissemination of the video stream to the rest of the system, each group of k pieces is used to create c additional encoded pieces. These are disseminated as any original piece using gossip . When a node has received at least k pieces, which can be any mix of original and encoded pieces, from a certain group it can decode them to retrieve the original k pieces. Figure 4 shows a graphical representation of the encoding/decoding process.

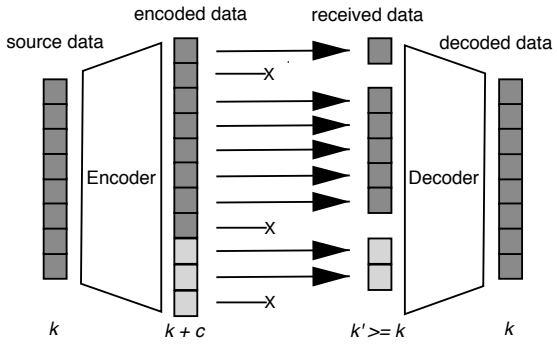

Figure 4: *An example of FEC behavior. Data is encoded, sent over an unreliable channel (some data is lost), and decoded at the receiver.*

Using three-phase gossip together with erasure codes has several benefits:

- As stated, all peers may not receive all advertisements even under ideal network conditions. Using FEC, it is possible to reconstruct video data that was never even requested from another peer.
- Message loss can occur in any of the three phases, and the data content of lost messages may still be recovered, eliminating the need for retransmission of those pieces, which would take longer time than the decoding.
- As the encoding procedure is deterministic, every node that has retrieved a group of pieces can encode it with the same result. By doing so and advertising the newly encoded pieces together with the rest of the pieces received in the last round, each node in the system can disseminate more information (although redundant) than it downloaded.

Regarding the added overhead when using FEC, given a group of $k + c$ encoded pieces, the overhead will be $\frac{c}{k+c}$. However, having peers to stop requesting pieces belonging to a certain group when k pieces of that group are received minimizes this overhead.

Claim implements a sophisticated retransmission scheme that uses the duplication of advertisements created by the gossiping protocol to its advantage. Typically, Claim is used to resend requests when more than c pieces in a group are lost (not delivered within a certain time). In this case, for each piece that is missing, requests are sent in a round-robin manner to the advertisers of the pieces, until the maximum number of retries r is reached. This increases load balancing, and minimizes the risk of sending requests to a peer that may have become unavailable, which could be the reason to the missing response. Each time a certain piece is detected as missing, the time after which the piece is considered missing is lowered by half, until it reaches some minimum value. An example of the retransmission mechanism is depicted in Figure 5.

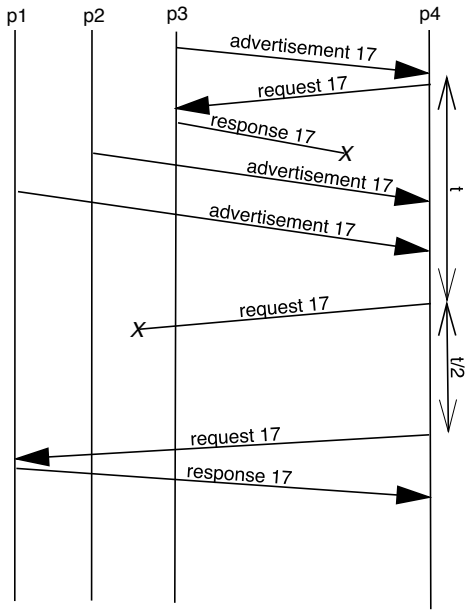

Figure 5: *An example of requests and timeout scheduling. While* p4 *waits for the piece with ID 17 two more advertisements for this piece are received. The reponse from* p3 *that contains the piece fails to be delivered, so when a request timeout is triggered a request for the piece is sent to the next advertiser,* p2*. This request fails to be delivered, and a new request timeout is triggered. The next advertiser,* p1*, is then requested to send the piece, which it does successfully.*

4.3 ISP Friendliness

To achieve ISP friendliness this thesis suggests using locality-biased (network-biased) neighbor selection to construct a clustered topology that reflects the network underlay, but using random, long-range, connections when close peers are not able to provide good stream quality. Intuitively this means constructing a small-world topology, meaning high clustering of peers and with small shortest path lengths, which can facilitate quick and stable data distribution through the entire topology[31]. In this section, a method for constructing the locality database is outlined, followed by a description of the NAT-aware PSS Croupier[10], and how to implement a network-biased PSS using Croupier.

4.3.1 A Network Model for AS Distance Lookup

The preferred solution for modelling the AS topology in this thesis is storing a database of distances between ASes locally at each peer. This approach provides a more accurate view and faster lookup than using round-trip times, or an approximation of the same, and is fully distributed, and again provides faster

lookup, compared to using a central service or collaborating with an ISP. The database also includes IP address to AS mapping for quick lookup of which AS a certain peer belongs to, and to prevent peers from reporting a false AS number, which could increase the amount of inter-AS traffic. The size of the database is not an issue – using efficient design and compression such a database takes between 5 and 10 MBs on disk. The main issue when using an AS topology model is instead how to accurately infer the topology from available data.

The network model and the algorithm to infer the AS topology into the model in this thesis is mainly based on [13] and their `ISPF-Lite` algorithm. Using public BGP data, the lengths of the shortest paths between all *transit* ASes are calculated – since the public data currently available contains roughly 40,000 ASes, storing the distance between all of them (assuming 1 B per pair of ASes) would require more than 1 GB. To reduce the size of the storage significantly, Hsu and Hefeeda suggest to only calculate the distances between transit ASes, and store parent relations for stub ASes separately. This approach reduces a large portion of the storage size thanks to the topology of ASes being scale-free – stub AS composes 83% of all ASes and only have 1.76 parents on average. To lookup the distance from a transit AS AS_t (can also be a stub AS) to a stub AS AS_s, first the parents of AS_s are retrieved from the database, followed by the distances from AS_t to AS_s's parents, and finally the distance to the closest parent is incremented by 1. A graphical representation of this is shown in Figure 6. Calculating the shortest paths between all transit ASes when only knowing their neighbor connections is an *all-pairs shortest path problem*, which is solved using the Floyd-Warshall algorithm[6]. Due to the uncertainty of AS relationship inference, only the distances between peers are considered. As inference methods become better, their usefulness in peer-to-peer systems will increase. This would however require considerable collaboration with ISPs, especially when inferring peer-to-peer relationships. Finally, according to `ISPF-Lite`, when the distance to two ASes is the same from a given reference AS, the one that shares the longest IP prefix with the reference AS is considered to be closer.

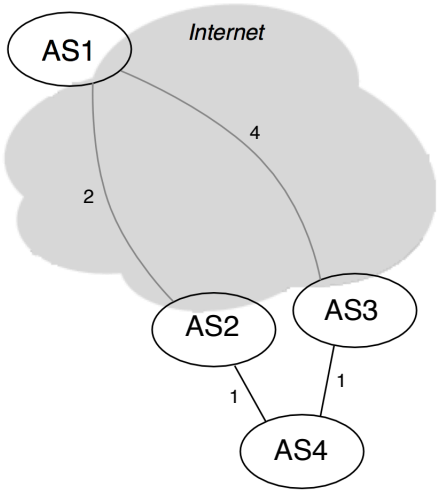

Figure 6: *Graphical representation of AS distance lookup. To find the shortest distance between* AS1 *and* AS4*, the distances to* AS4's *parent ASes are compared, and the shortest distance is incremented by 1. A distance of 2 means that traffic has to pass through one AS between* AS1 *and* AS2 *(4 means three ASes between* AS1 *and* AS3*). The shortest path between* AS1 *and* AS4 *is therefor through the parent* AS2*, and its length is 3.*

4.3.2 A Network-biased Peer Sampling Service

To provide peers with close neighbors a Network-biased PSS (NPSS) is used. It is implemented by using an existing PSS, Croupier[10], which serves the NPSS with random peers. Croupier is a gossip-based PSS that provides uniform random samples of peers even in the presence of NATs in the network, without using relaying or hole punching. It is thus more robust and has lower overhead than existing protocols with similar properties—which all do either relaying or hole punching—especially when the percentage of peers behind NATs is high.

NPSS is implemented in a similar fashion to the ISP-friendly peer selection algorithm by Shen and Zimmermann [26]. However, as Croupier provides random peers there is no risk of partitioning, and thus no need for using a PDF. Instead the NPSS has as its single goal to provide the closest peers possible in the system. Using gossip, and given a certain view size n, NPSS peers communicate all or a subset of their view to a randomly selected peer in the view, or occasionally from Croupier to find new peers. In the following processing phase, a peer uses the AS distances lookup tool to decide which n peers of its current and the newly received to keep, in other words which n peers that are closest. This results in the NPSS peers quickly converging into a very clustered topology.

4.3.3 ISP-Friendly Neighbor Selection

To achieve ISP friendliness the system implements a neighbor selection algorithm that aims at minimizing the average AS distance to the neighbors in each peer's view, provided that a sufficient download rate is preserved, that is, a download rate on average equal to the stream rate, plus overhead. The main motivation for using ISP-friendly neighbor selection over ISP-friendly chunk scheduling is that the former require no modification of the dissemination algorithm. The neighbor selection approach can therefor fairly easy be used by many peer-to-peer systems, including currently deployed.

Close peers are provided by the NPSS, but in the case that the stream rate becomes insufficient, or too few close peers are available, Croupier also provides random peers. This implies that some structure is introduced into the system's overlay, intuitively a small-world structure, where peers are clustered according to the underlying network topology, but have some random, possibly long-distance, connections.

A peer's view consists of neighbors that the peer either has a *close* or a *random* connection with, referred to as close and random neighbors, respectively. When sending advertisements to neighbors, there is not distinction between close and random neighbors, meaning that the probability of a random neighbor being chosen as a recipient is the same as for a close neighbor.

A close connection is connection that is established as a result of *(i)* one peer finding another through the NPSS, sends a connection request, and *(ii)* upon receiving the request, the other peer evaluates its current view and sees that the sender of the request is closer than the peer farthest away of its current *close* neighbors. Since these connections compose a structured overlay, without proactiveness, close neighbors will not be removed from the view unless explicitly disconnected, why it is advisable to use some timer or heartbeat mechanism to deal with failing peers.

To establish a random connection, a peer sends a connection request that is explicitly defined as a request for a random connection. The purpose of random connections is to provide peers with insufficient download rate from close peers with additional resources. Each peer has a few dedicated slots for outgoing random connections, and requests for random connections are always accepted. If all slots should be occupied when a new random connection is requested, one of the current random connections is disconnected in favor for the new request. Therefor, as long as a peer is experiencing an insufficient download rate it should continue to request random connections at a regular interval, and a peer that has outgoing random connections should disconnect them after a certain time. Doing so helps to prevent against churn, and respects the design decisions of Gossip++, while also minimizing inter-AS traffic.

The introduction of an additional neighbor set and the task of managing connections to achieve ISP friendliness while delivering a clear stream to all peers mean that the maximum number of connections of the two types is important. With too few close connections allowed, peers in the same AS won't be able to connect to each other effectively, and with too many the number of peers outside

the AS included in the set of close neighbors will increase for peers in smaller ASes. Random connections will always have an adverse effect on network-biased topology construction, however using only close connections may not be enough to provide all peers with enough connections to deliver a clear stream. The effect of varying the number of allowed connections to close and random peers is evaluated in Section 6.

5 System Implementation

This section provides implementation specific details about the system presented in this thesis. The system is implemented in Java and uses the Kompics P2P Framework[29]. It uses HTTP Live Streaming[19] (HLS), currently an Internet-Draft, for input and output of data. This section will first give an introduction to Kompics, and then present the system architecture, followed by the messages used by the components in the system. Finally, I/O and FEC handling, and the monitoring capabilities are described.

5.1 Implementation Framework: Kompics

Kompics is composed of a component model and a programming framework implemented in Java[2]. The components are reactive, event-driven state machines that execute in parallel. They communicate by passing events carrying data through bidirectional ports, connected by channels. This section will describe the Kompics abstraction model, followed by an introduction to the runtime environment, and finally present an overview of the provided network interface and the NAT traversal capabilities of Kompics.

5.1.1 Model

The fundamental conceptual entities in Kompics are components, events, ports, channels, event handlers, and subscriptions. Events are passive and immutable typed objects that have some attribtues. In the Java implementation of Kompics all events extend the root event type Event. For example, the Message type is an event with attributes *source* and *destination* addresses. A port is a bidirectional component interface that allows only a certain set of event types to pass through. Ports are provided by the component that implements the protocol that the port interfaces. A channel is a connection between two ports of the same type, one on a providing component and the other on the using component, i.e. the channel represents the possibility for the components to trigger messages to each other. To make it possible for an event handler to receive events there has to exist a subscription to the appropriate port.

All Kompics components provide a *control port*, which is used to initiate, start, and stop the component, and to trigger fault events in the case of uncaught exceptions in the component. Figure 7 shows a simplified diagram of the Network-biased PSS. The solid arrows represent subscriptions, the dashed arrows triggering of events, and the provider side of a port is denoted by a minus sign (-).

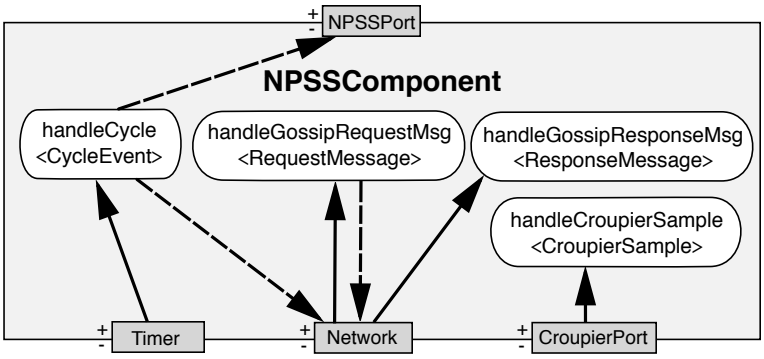

Figure 7: *An example of a Kompics component – the Network-biased Peer Sampling Service with its ports and handlers. The* CroupierSample *contains random peers provided by Croupier. A timer component triggers scheduled periodic* CycleEvents *to the* handleCycle *handler, which triggers a* RequestMessage *to another peer on the* Network *port, and may trigger a* NPSSSample *of close peers on the* NPSSPort. *The* RequestMessage *contains a set of the sender's neighbors. When a* RequestMessage *is received at a peer, the* handleGossipRequestMsg *handler will respond with neighbors from its own view by triggering a* ResponseMessage *on the* Network *port. The two peers participating in the exchange will then evaluate which peers to keep in their views.*

5.1.2 Runtime Environment and Scheduling

During execution, a Kompics component is either marked as *idle*, *ready*, or *busy*, indicating whether it has no events, has events waiting to be processed, or is currently executing an event, respectively. To execute the events, Kompics uses a pool of worker threads that each has a queue of components that are ready. A worker only executes one component at a time, and a component can be executed by at most one worker at any given time. If a worker's queue becomes empty it uses *work stealing* to steal half of the ready components in the queue of the worker with most ready components.

Kompics provides a deterministic simulation mode, which implements a special scheduler that guarantees deterministic execution, provided that no threads are created in the components themselves. This thesis uses this scheduler in the evaluation in Section 6.

5.1.3 Network and NAT traversal

To send messages between components Kompics provides implementations of UDP and TCP. UDP is used between all components implemented in this thesis. The reasons for using UDP instead of TCP are several; faster connection establishment, no cost for keeping connections open, and the possibility of NAT traversal. As seen in Section 6, FEC and the retransmission mechanism of the

system provide strong protection against loss of messages when using unreliable channels.

The NAT traversal component of Kompics is implemented according to the hole punching techniques provided by Roverso et al. [24].

5.2 System Architecture

The core components composing the system described in this thesis are shown in Figure 8. The following sections will describe implementation specific details about these components.

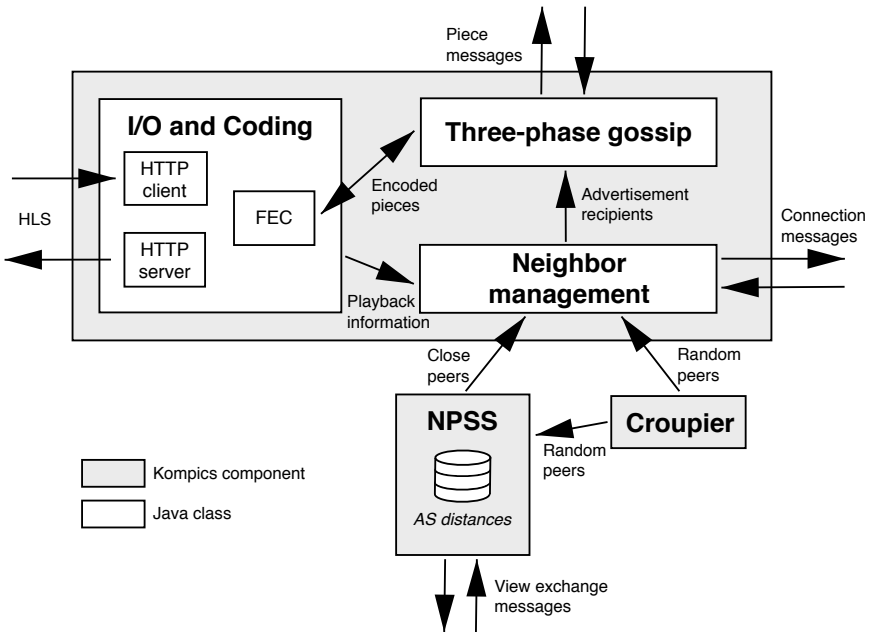

Figure 8: *System diagram of the core components and their communication in the system. For example, typical operation at the source is that the I/O component reads data from an HLS stream, encodes the video data into pieces, and sends advertisements to peers provided by the neighbor management component. Upon receiving a request with a certain piece identifier, the source will send a response containing the corresponding piece. The diagram also includes the Croupier component.*

5.3 Messages

The messages implemented in the system belongs to three different groups: view exchange messages used in the NPSS, messages used to establish connections between peers, and messages used in the content dissemination protocol.

5.3.1 View Exchange Messages

The NPSS component uses two messages, both used in the communication phase when gossiping information about neighbors:

GossipRequestMessage The message used to request an exchange of views with the recipient of the request. The payload of the message is a subset of the requesting peer's view.

GossipResponseMessage The message sent in response to a GossipRequestMessage. Its payload consists of a subset of the responding peer's view.

5.3.2 Connection Messages

Three types of messages are used to manage connections:

ConnectionRequestMessage A request to another peer to be be inserted into its view. A flag indicates whether it is a request for a random or a close connection.

ConnectionResponseMessage Sent in response to a request. The response message is only sent if the connection request was accepted.

DisconnectionMessage Sent to a peer that was removed from the sender's view. A disconnection message is triggered when an old connection is replaced with a new one, or when a timer associated with the connection experies, for example after that the connection has been inactive for a while.

5.3.3 Piece Messages

Piece messages are used in the content dissemination. Each sub-piece carries 1316 bytes video data, and is identified by an integer that is unique in the video stream.

PieceAdvertisementMessage A message sent to inform a subset of the peers in the sender's view that new pieces are available. The message contains a set of piece identifiers.

PieceRequestMessage A message sent to request pieces from a peer that previously advertised them. The request message contains a set of piece identifiers corresponding to the desired pieces.

PieceResponseMessage The message which contains the actual piece data. A response message contains a single piece, to keep the message's size below the maximum transmission unit (MTU) of 1500 bytes, which is imposed by the underlying network layer. A message larger than 1500 bytes has to be fragmented, which would increase the overhead in the video dissemination.

5.4 I/O and Piece Coding

The main task of the I/O component is to convert an HLS stream into encoded pieces and vice versa.

The source node in the system reads an HLS stream, for example from VLC, and stores the stream into pieces. These pieces are then encoded using FEC. The encoded pieces have an identifier, a group identifier that indicates which pieces that were encoded together, and a payload of 1316 bytes. The encoded pieces are then advertised by the dissemination component to some peers that are provided by the neighbor management component. When the peers receive the advertisements they request the pieces, and the source sends responses containing the pieces.

When a peer receives a response message it will advertise the piece to a subset of its own neighbors. As soon as enough encoded pieces from a certain group are received the I/O component decodes them, encodes the group again, and the dissemination component will then advertise the pieces in this group that were not already advertised. The I/O component uses the HTTP server to send the content of the decoded pieces as an HLS stream.

The main advantage of using HLS is the codec transparency – the system does not need to know how the video is encoded and encapsulated. At the source, the I/O component only reads some content from an HTTP server and does not have to analyze it further. The same applies when the I/O component has decoded a piece and it is streamed through the local HTTP server. Thereby, it is up to the users of the system to provide software that supports the codecs used in the stream, that is, can perform decompression the video data. This means that any compression mechanism can be used, including those that do not exist yet, without any altering of the system.

5.5 Monitoring

In addition to the core components, the system also implements monitoring capabilities for local and remote monitoring. The monitoring information includes the number of different messages sent, and whether they were sent to a close or a random peer, current connections, upload and download rate, stream lag, buffer length, and more.

Local monitoring is available using Java Management Extensions[5] (JMX). The *jconsole* application for monitoring of JMX enabled applications is part of the Java platform since version J2SE 5.0, released in 2004, and thus present on many systems.

Remote monitoring is implemented using REpresentational State Transfer[6] (REST). The same data that is exposed for local monitoring can also be sent to a RESTful web service, where it is possible to monitor the overall system execution in real time or analyze the collected data after dissemination end.

[5] Java Management Extensions (JMX) . http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html

[6] Representational State Transfer (REST) . http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Remote monitoring is mainly intended for aggregate monitoring of the system, but it is also possible to look at a single peer or a subset of the peers, such as peers from a certain AS. Remote monitoring has to be enabled by the user.

By using these fairly ubiquitous technologies for monitoring, it becomes possible for anyone interested in the system's behavior to get information about it through the use of common tools and interfaces.

6 Experiments and Evaluation

In this section, the system is evaluated with respect to this thesis' objectives. Section 6.1 gives an overview of the simulation framework is given, including network environment and locality, and Section 6.2 presents the simulations scenarios and their results.

6.1 Environment and Configuration

6.1.1 The Kompics Simulator

The system is evaluated by running scenarios using the Kompics simulator. This simulator executes the whole system deterministically, including the Network and Timer abstractions. The simulator intercepts all calls for current time and returns the simulated time. Thread creation calls are also intercepted, and causes the simulator to halt the simulation since deterministic execution can't be guaranteed.[2]

6.1.2 Network Model and Stream Dissemination

In the following scenarios, if not otherwise specified, 200 nodes are run with a fanout of 8, except the source, which have a fanout of 5. Each round the source advertises newly received data from a video stream, e.g. fed by VLC, to the system, the stream consisting of 70 encoded sub-pieces per round, 744 kbps, on average. When bandwidth is constrained, the source has the capability of serving the stream to 5 peers per round, and other nodes have a token bucket of 200 kB. Any packet sent after reaching an upload of 200 kB in one round is dropped. In the scenarios the FEC implementation handles sub-pieces in groups of 100 and uses 5 redundant pieces, meaning that $k = 100$ and $c = 5$ according the notation from Section 4.2.2. Link latency is roughly between 0 ms and 500 ms according to King's Latency Model[7], and 1% of all messages are dropped to simulate unreliable links.

6.1.3 Locality

Joining peers are assigned to an AS according the distribution of public BGP data available. This means that the distribution of peers in ASes follows that of the Internet topology, that is, a power law distribution. For example, in a simulation with 100 peers there are roughly 50 ASes populated – 3 of them containing 10 peers or more, and 30 of them containing only 1 peer.

[7] King : A tool to estimate latency between any two Internet hosts, from any other Internet host. . http://www.mpi-sws.org/~gummadi/king/

6.2 Evaluation

6.2.1 Ideal Scenario

In the ideal scenario, no message loss occur and all peers have unlimited bandwidth. The peers gossip advertisements with a fanout of 8. The ideal scenario evaluates the system's ability to broadcast enough advertisements to all peers in the system, which is a prerequisite for content delivery in a more constrained environment. Figure 9 shows that in an ideal setting, all 200 nodes receive a clear stream with a stream lag of at most 3 seconds.

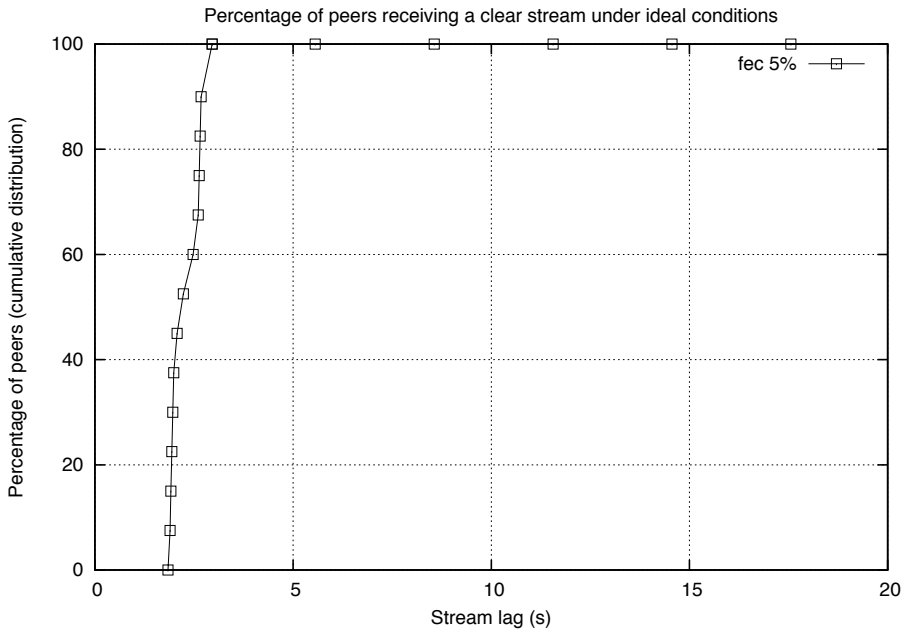

Figure 9: *In an ideal setting, all peers receive a clear stream.*

6.2.2 Message Loss Scenario

This scenario evaluates the system's behavior when upload bandwidth is limited and message loss occurs for 1% of all messages on average. In Figure 10, the two mechanisms used to prevent and recover from message loss, FEC and retransmission, are evaluated both separately and in combination.

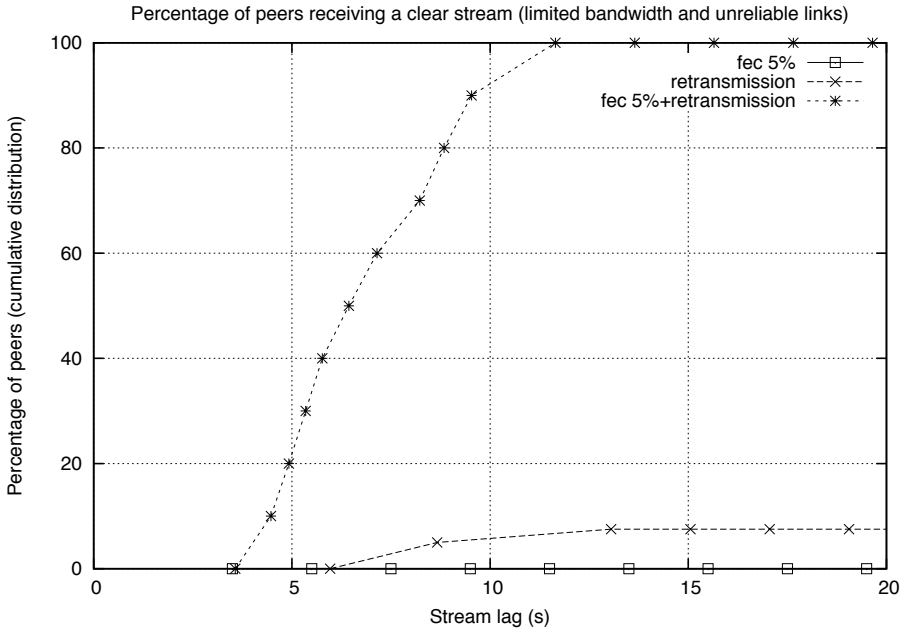

Figure 10: *Percentage of peers receiving a clear stream. Using FEC or the retransmission mechanism separately is not enough, however using them together achieves delivery of a clear stream to all peers.*

When only using FEC, the system is unable to leverage the duplication of advertisements in the gossiping process, and suffers greatly from the bandwidth constraints and message loss. When solely relying on retransmission instead, peers have to receive all pieces since reconstruction of any missed pieces is not possible. Only a few peers receive enough advertisements and responses to deliver a clear stream in this case.

The combined use of FEC and retransmission do however let the system deliver a clear stream to all peers. This combination allows peers to resend requests to any peer which they received an advertisement from, and FEC enables reconstruction of pieces that were not received.

6.2.3 Churn Scenario

In the churn scenario, the system's ability to cope with various churn rates is evaluated. The simulation environment is the same as in the limited bandwidth and message loss scenario, with the addition that each round there is N peers currently present in the system that fails, and N new ones join.

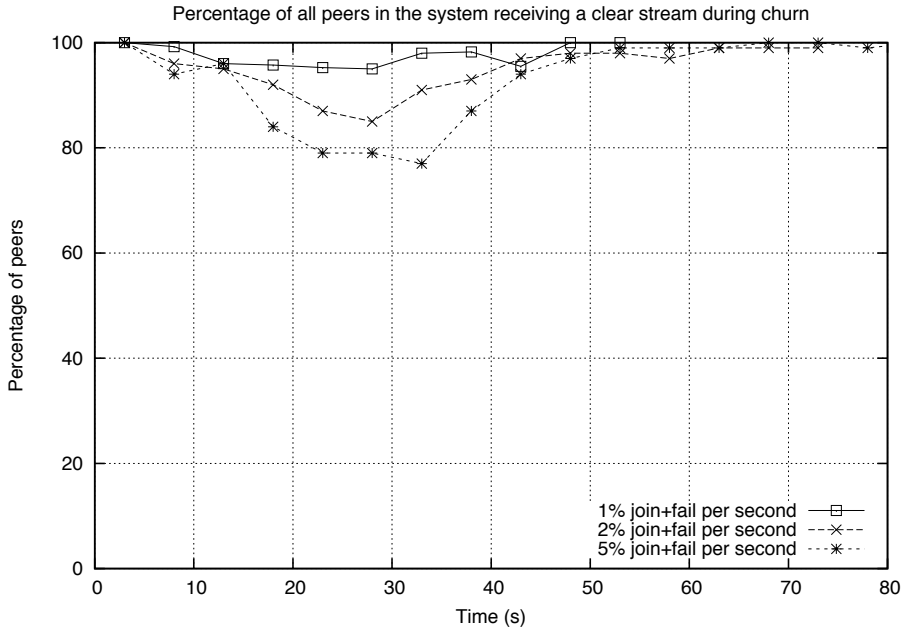

Figure 11: *Percentage of all peers currently in the system at a certain time that receives a clear stream.*

In Figure 11 the churn occurs roughly between 10 and 20 seconds, a total of 10 rounds. For example, with a system size of 100 peers and $N = 2\%$, a total of 20 peers will have failed, and 20 new ones joined after 10 seconds. The figure shows the percentage of peers that receive a clear stream at a certain time when 1%, 2%, and 5% of churn is experienced, respectively. Both peers present in the system before the churn events and new peers are included in the data. There is a notable effect on the system for roughly 30 seconds. Figure 12 shows the percentage of peers that are unaffected by the churn. Most of the peers are unaffected—that is, does not miss any piece—by 1% and 2% of churn, however a significant portion is when the system experiences 5% churn.

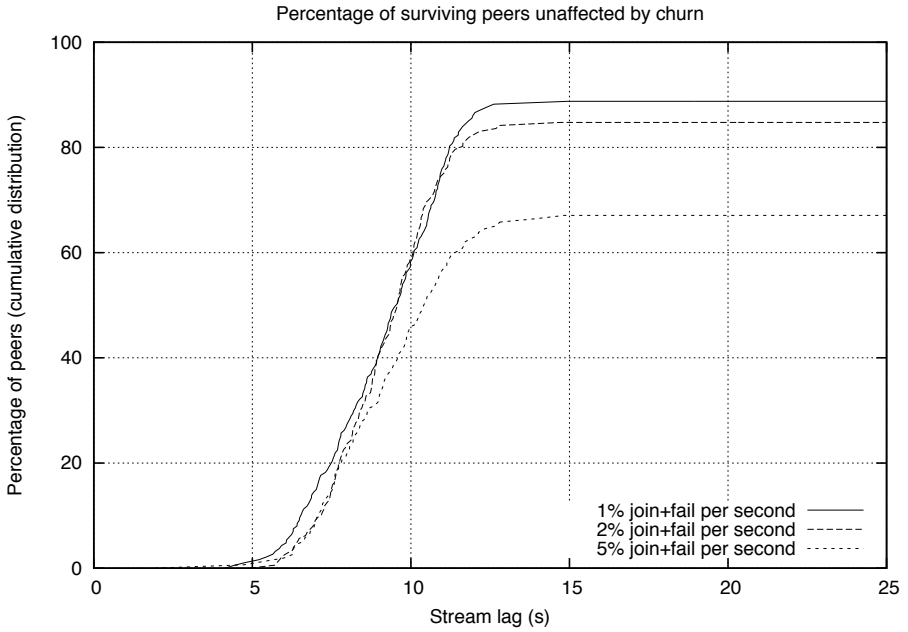

Figure 12: *Percentage of surviving peers that received a clear stream during and after 10 seconds of churn.*

6.2.4 Crash Scenario

To evaluate how the system handles a large portion of the peers failing simultaneously, for example due to the failure of a physical link that provides connection to a large network, this section evaluates how peers' stream rates are affected when 20% and 50% of the peers in the system fails at the same time, respectively. Figure 13 shows the percentage of peers initially in the system that receive a clear stream. The effects on the surviving peers when a certain portion of the peers crashes only last for a short period of time.

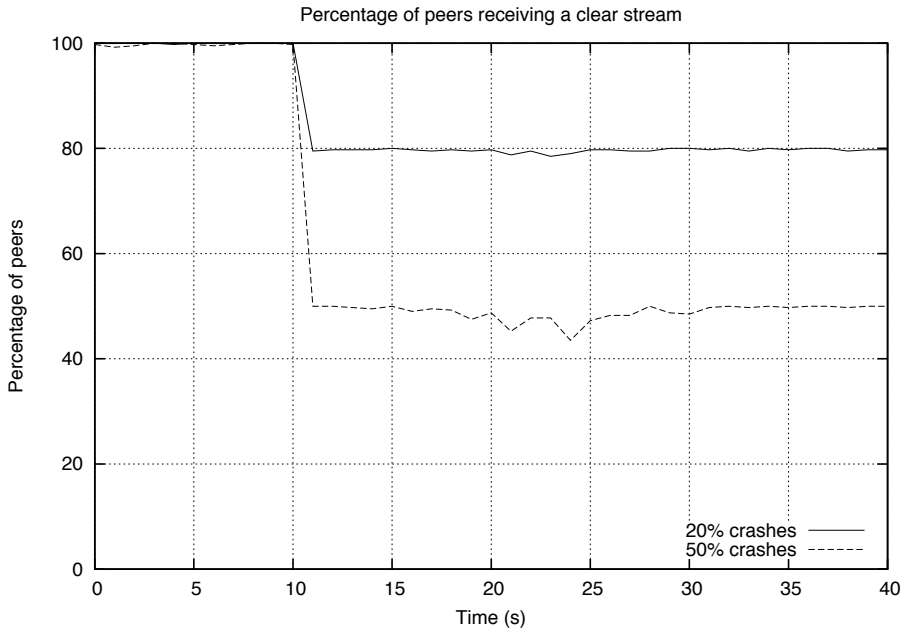

Figure 13: *Percentage of peers that receives a clear stream after a portion of the peers have failed.*

6.2.5 ISP Friendliness

In this section, the ISP friendliness achieved by the implementation of network-biased neighbor selection is evaluated. The importance of number of connections to close and random peers is evaluated in Tables 5 and 6. Table 5 shows the percentage of peers in a system of 500 peers that receive a clear stream for various combinations of maximum close and random connections allowed. Table 6 shows the average AS distance (hop count) to neighbors for all peers, and the percentage of intra-AS traffic exchanged in the system for the same combinations.

		Random			
		0	*1*	*2*	*3*
Close	*10*	85%	99%	100%	100%
	20	96%	99%	100%	100%
	30	97%	99%	100%	100%

Table 5: *Percentage of peers in a system of 500 peers that receive a clear stream for various combinations of maximum allowed close and random connections.*

		Random							
		0		*1*		*2*		*3*	
Close	*10*	1.8	45%	1.9	38%	1.9	33%	2.1	28%
	20	0.9	48%	1.0	40%	1.2	39%	1.8	36%
	30	1.0	43%	1.2	40%	1.2	38%	2.3	34%

Table 6: *Average AS distance to neighbors and percentage of intra-AS traffic in a system of 500 peers for various combinations of maximum allowed close and random connections.*

The evaluation of number of connections has two notable properties. First, the number of random connection greatly affects system performance. Using only close connections is not enough to provide all peers with a clear stream, and introducing a single random connection increases the percentage of peers that do significantly. Second, the percentage of intra-AS traffic does not always correlate with the average AS distance to neighbors. The explanation to this could be that if a peer has 10 close neighbors in the same AS and one random neighbor 2 hops away, the peer will receive the same amount of intra-AS traffic as a peer with the same number of close neighbors, but with one random neighbor 4 hops away, however their average neighbor distance will differ. The same reasoning can be applied to different distributions of close neighbors. Finally, it is also pointed out that 10 close connections seem insufficient to connect peers in the same AS to each other effectively, as there is less intra-AS traffic in that

case compared to that when allowing 20 close connections.

The evaluation shows that in a system of 500 peers, a combination of 20 close neighbors and 2 random neighbors provide the highest percentage of intra-AS traffic among the combinations that are able to deliver a clear stream to all peers. This is therefor the setting used in these experiments.

To measure the clustering effect on the topology when introducing an ISP-friendly neighbor selection mechanism, the average distance to neighbors, meaning the average number of hops between all connected peers in the system, is recorded for different system sizes when using random (unbiased) neighbor selection and ISP-friendly neighbor selection, respectively. The results are shown in Figure 14; the ISP-friendly approach is labeled *ispf*. The ISP-friendly structuring of peers lowers the average AS distance by 1/3 in a system of 100 peers, and by almost 1/2 in a system of 1000 peers, compared to the random neighbor selection.

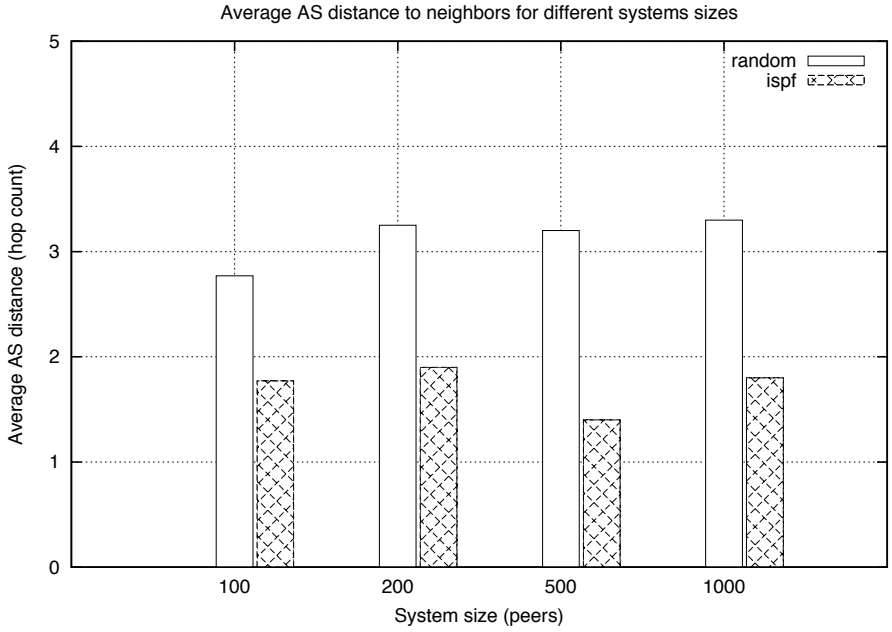

Figure 14: *The average AS distance to neighbors indicates the amount of clustering in the system.*

In Figure 15 the distribution of traffic in the system is presented, categorized as *intra-AS traffic*, *neighbor traffic*, or *other traffic*, meaning traffic between peers that have an AS distance of 0, 1 or above 1, respectively. As the system size increases, there is a clear trend of increasing intra-AS traffic when using the ISP-friendly neighbor selection. Random neighbor selection results in at most 4% of intra-AS traffic – in a system of 200 peers, the ISP-friendly approach

achieves 10 times more intra-AS traffic compared to a random one. In addition to increasing the intra-AS traffic, the evaluation shows that neighbor traffic also benefits from the topology-aware neighbor selection, meaning that inter-AS traffic is minimized further.

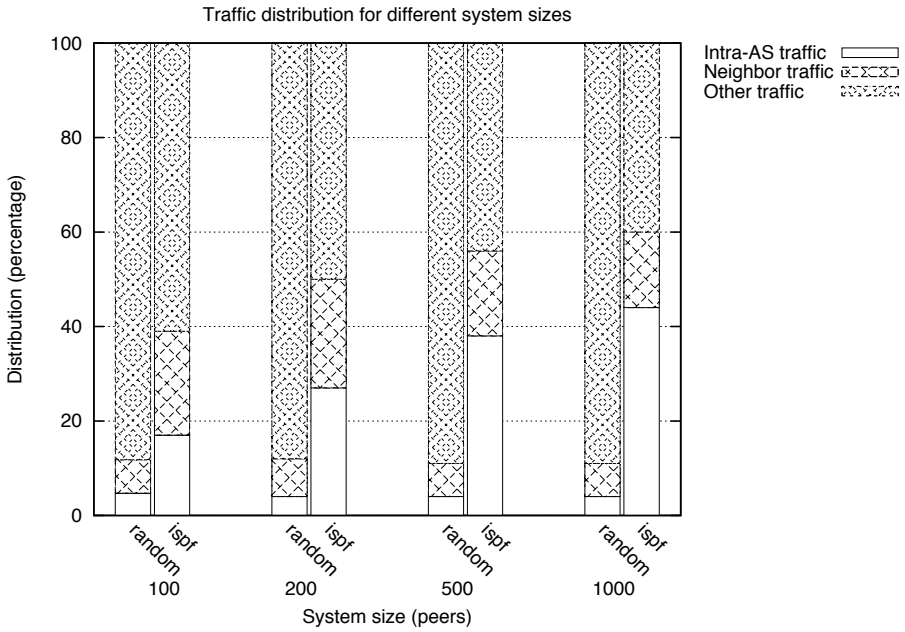

Figure 15: *Traffic distribution for different system sizes – intra-AS traffic is traffic between peers with an AS distance of 0, neighbor traffic between peers with a distance of 1, and other traffic between peers with a distance of at least 2 hops.*

While Figure 15 shows that there is an increase of intra-AS traffic proportional to the system size, there is a lower increase per additional peer as the system grows larger. This is because the power law relationship of AS sizes, meaning that there are many ASes with few peers. In large ASes, which are populated by many peers, the percentage of intra-AS traffic becomes higher. For example, in a system of size 500, the peers in the largest AS, which contain 56 peers, experience 75% intra-AS traffic.

Finally, Figure 16 illustrates how the average AS distance between all connected peers converges in a sytem of 500 peers.

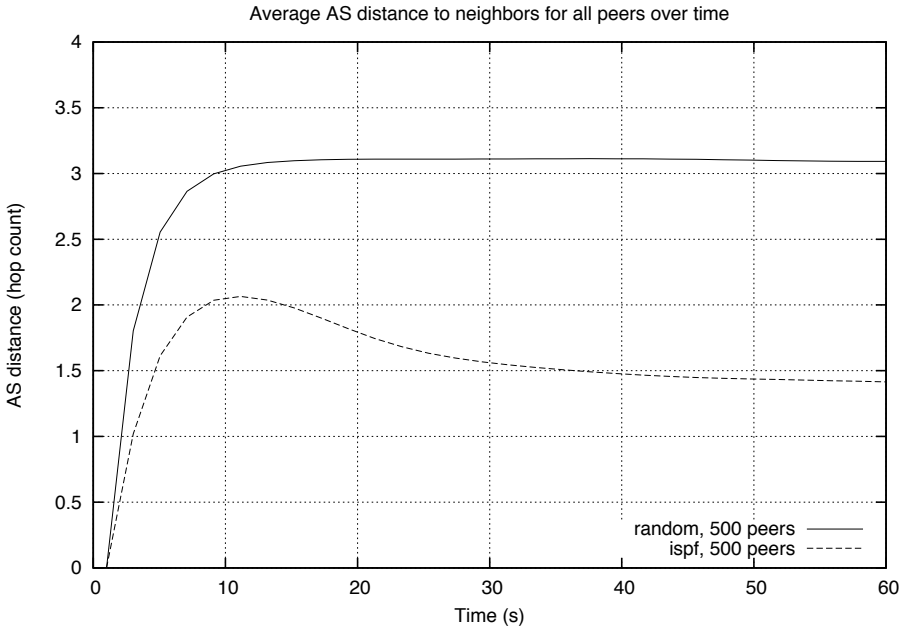

Figure 16: *Average AS distance of connected peers over time for the first 60 seconds in a stream. Peers are joining the system during the first 5 seconds.*

6.2.6 Summary

In this section the implementation of the system design presented in this thesis was evaluated by running simulations using the Kompics simulator. The experiments show that the system is able to provide all peers with a clear stream in a constrained environment, thanks to the combination of the FEC and retransmission mechanisms, and provides robustness in cases of churn or a significant number of peers crashing.

The introduction of a network-biased peer sampling service creates a more clustered topology and achieves a considerable increase of intra-AS traffic compared to random neighbor selection. The evaluation also shows the importance of allowing peers to create a some random connections when peers provided by the network-biased PSS is not enough to deliver a clear stream. Moreover, the increase of traffic between neighboring ASes shows the importance of always minimizing the distance to neighbors, regardless of whether they are inside or outside the own AS.

7 Conclusion

In this thesis, a peer-to-peer live streaming system was designed with the main goal of being ISP friendly. The design uses a gossip-based protocol for content dissemination, and biases neighbor selection towards closer peers to achieve ISP friendliness. A network-aware peer sampling service was implemented to provide the system with close peers, and to evaluate which peers that are closer the service consults a database of AS distances that is constructed from public BGP data and stored locally at each peer. Implementing biased neighbor selection in a system does not require any modification of the dissemination protocol, and can therefor be applied to a wide range of peer-to-peer systems.

The system was evaluated in various simulation scenarios, and proved to operate well in a constrained environment as well as during peer failures. The thesis also investigates to what extent peers can be clustered and still be able to deliver a clear stream, and the number of random, long-range connections is observed to be especially important. It is possible to have high clustering as long as a few random connections are allowed to be created when close neighbors can't provide a sufficient download rate. Comparing the ISP-friendly neighbor selection to random selection, inter-AS traffic is efficiently reduced in the overall system, and in larger ASes most traffic is exchanged within the AS.

Moreover, the system implements ubiquitous monitoring capabilities using JMX and REST. These techniques are available on all modern systems, provided a Java runtime environment for using JMX. This allows users, researchers, system operators, and others who are interested to monitor the application locally or remotely.

To summarize, the system design proposed in this thesis successfully provides ISP friendliness, and this or similar designs should be considered by content providers and developers of all large-scale peer-to-peer applications. By doing so, the cost for ISPs to allow peer-to-peer applications to exchange traffic in their networks is lowered, and thus so is the risk of ISPs limiting or blocking peer-to-peer traffic completely.

8 Future Work

Neighbor selection at the source This thesis does not incorporate or evaluate any special dissemination protocol or neighbor selection for the source. Considering clustered topologies, it may be beneficial to let the source have mainly random connections, which could accelerate propagation of information in the overall system, and in the case of video streaming minimize stream lag.

Distributed incentive mechanism In the design depicted in this thesis there is no incentive mechanism to prevent free riding. If a too large portion of peers are free riders none of the peers will be able to deliver a clear stream, since the total upload capacity of the system becomes to low. To design an effective incentive mechanism is therefor a key issue for many peer-to-peer systems. Monod [18] provides such a mechanism, however it assumes a random, non-biased, neighbor selection – peers that show signs of biasing their neighbor selection, sending of advertisements, or responses, are assumed to not follow the protocol correctly, and possibly collaborating with other free riders. This design would therefor not work in a system that biases its neighbor selection, unless some modification to account for the clustering of peers is introduced.

Inferring AS relationships This thesis' notion of ISP friendliness only regards the closeness of peers, meaning any traffic leaving the AS is bad, increasingly so with the number of other ASes it has to pass through. In reality ISPs can have different agreements on the traffic passing between ASes, some worse than others and some not bad at all. Thus, when choosing peers to exchange data with in an ISP-friendly way the system could benefit from taking these relationships into account. Algorithms and thoughts on inferring these relationships from public BGP data (the source for calculating distances in this thesis) is provided by Gao [12] and Dimitropoulos et al. [8].

Piece scheduling To further minimize inter-AS traffic, modifications of the gossiping protocol should be considered. One possibility is for peers to sort advertisements on AS distance to the advertiser. Doing so, peers would resend timed out requests to closer peers first, and they could also wait for a short period of time before requesting a piece for the first time, to see if any more advertisements, possibly from closer peers, are received. The latter would however result in some increase of stream lag.

References

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

[2] C. Arad and S. Haridi. Kompics: a message-passing component model for building distributed systems. 2010.

[3] F. Boulos, B. Parrein, P. Le Callet, D. Hands, et al. Perceptual effects of packet loss on h. 264/avc encoded videos. 2009.

[4] D.R. Choffnes, M.A. Sánchez, and F.E. Bustamante. Network positioning from the edge-an empirical study of the effectiveness of network positioning in p2p systems. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–5. IEEE, 2010.

[5] D. Ciullo, M.A. Garcia, A. Horvath, E. Leonardi, M. Mellia, D. Rossi, M. Telek, and P. Veglia. Network awareness of p2p live streaming applications: a measurement study. *Multimedia, IEEE Transactions on*, 12(1): 54–63, 2010.

[6] T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[7] L. D'Acunto, J. Pouwelse, and H. Sips. A measurement of nat and firewall characteristics in peer-to-peer systems. In *Proc. 15-th ASCI Conference*, volume 5031, pages 1–5. PO Box, 2009.

[8] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, G. Riley, et al. As relationships: Inference and validation. *ACM SIGCOMM Computer Communication Review*, 37(1):29–40, 2007.

[9] B. Donnet, B. Gueye, and M.A. Kaafar. A survey on network coordinates systems, design, and security. *Communications Surveys & Tutorials, IEEE*, 12(4):488–503, 2010.

[10] J. Dowling and A.H. Payberah. Shuffling with a croupier: Nat-aware peer-sampling. 2012.

[11] UPnP Forum. Internet gateway device (igd) v 2.0. URL `http://upnp.org/specs/gw/igd2/`.

[12] L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking (ToN)*, 9(6):733–745, 2001.

[13] C.H. Hsu and M. Hefeeda. Isp-friendly peer matching without isp collaboration. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 75. ACM, 2008.

[14] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 6–6. USENIX Association, 2005.

[15] A. Legout, G. Urvoy-Keller, P. Michiardi, et al. Understanding bittorrent: An experimental perspective. *INRIA Sophia Antipolis/INRIA Rhne-Alpes-PLANETE INRIA France, EURECOM-Institut Eurecom, Tech. Rep*, 2005.

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

[17] N. Magharei, R. Rejaie, V. Hilt, I. Rimac, and M. Hofmann. Isp-friendly live p2p streaming. *University of Oregon, Tech. Rep*, 2009.

[18] M. Monod. *Live Streaming with Gossip*. PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2010.

[19] R. Pantos and W. May. Http live streaming. Internet-Draft draft-pantos-http-live-streaming-08.txt, Apple Inc., March 2012.

[20] A.H. Payberah and J.D.S. Haridi. *Distributed Optimization of P2P Media Delivery Overlays*. Information and Communication Technology, KTH Royal Institute of Technology, 2011.

[21] M. Piatek, H.V. Madhyastha, J.P. John, A. Krishnamurthy, and T. Anderson. Pitfalls for isp-friendly p2p design. In *Proc. of HotNets*, 2009.

[22] F. Picconi and L. Massoulie. Isp friend or foe? making p2p live streaming isp-aware. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 413–422. IEEE, 2009.

[23] R. Rodrigues and P. Druschel. Peer-to-peer systems. *Communications of the ACM*, 53(10):72–82, 2010.

[24] R. Roverso, S. El-Ansary, and S. Haridi. Natcracker: Nat combinations matter. In *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, pages 1–7. IEEE, 2009.

[25] H. Schulze and K. Mochalski. Internet study 2008/2009. *IPOQUE Report*, 2009.

[26] Z. Shen and R. Zimmermann. Isp-friendly peer selection in p2p networks. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 869–872. ACM, 2009.

[27] Z. Shen, J. Luo, R. Zimmermann, and A.V. Vasilakos. Peer-to-peer media streaming: Insights and new developments. *Proceedings of the IEEE*, 99 (99):1–21, 2011.

[28] M. Steiner and E. Biersack. Where is my peer? evaluation of the vivaldi network coordinate system in azureus. *NETWORKING 2009*, pages 145–156, 2009.

[29] The Kompics Team. Kompics. URL `http://kompics.sics.se/`.

[30] X.F. Wang and G. Chen. Complex networks: small-world, scale-free and beyond. *Circuits and Systems Magazine, IEEE*, 3(1):6–20, 2003.

[31] C. Wu, B. Li, and S. Zhao. Magellan: Charting large-scale peer-to-peer live streaming topologies. In *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*, pages 62–62. IEEE, 2007.

[32] S. Xie, G.Y. Keung, and B. Li. A measurement of a large-scale peer-to-peer live video streaming system. In *Packet Video 2007*, pages 153–162. IEEE, 2007.