Supporting Transitions in Peer-to-Peer Video Streaming

Master-Arbeit
Björn Richerzhagen
PS-D-0004

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik
und Informationstechnik
Institut für Datentechnik

Fachgebiet P2P Systems Engineering
Prof. Dr. David Hausheer

Supporting Transitions in Peer-to-Peer Video Streaming
Master-Arbeit
PS-D-0004

Eingereicht von Björn Richerzhagen
Tag der Einreichung: 12. November 2012

Gutachter: Prof. Dr. David Hausheer

Betreuer: Julius Rückert, M. Sc.
Dipl.-Wirtsch.-Inform. Matthias Wichtlhuber

Technische Universität Darmstadt
Fachbereich Elektrotechnik und Informationstechnik
Institut für Datentechnik

Fachgebiet P2P Systems Engineering
Prof. Dr. David Hausheer

Ehrenwörtliche Erklärung

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

Darmstadt, den 12. November 2012 Björn Richerzhagen

Abstract

Multimedia streaming applications are the single largest cause of downstream traffic in the Internet. The load on such a streaming service depends highly on the time of the day as well as on the content of the stream. Real-world events can trigger spontaneous and massive increases in the amount of users over a short period of time. Thus, a streaming service has to operate under highly dynamic conditions and has to tolerate massive arrivals and departures of participants. Providing a service at a large scale via a Content Distribution Network (CDN) or cloud-based architecture implies significant operational cost. Here, the Peer-to-Peer (P2P) paradigm is a promising alternative, as it provides scalability by utilizing the consumers' resources. A multitude of P2P live streaming applications have been proposed in recent years. However, those systems are tailored towards specific scenarios, under which they expose their best performance.

In this thesis, we present a new design paradigm for P2P streaming systems, that enables seamless transitions between different mechanisms. We propose a layered architecture, with well-defined protocols for the neighborhood discovery and the scheduling mechanisms, to support different such mechanisms in one system. Thereby, the streaming system can adapt to the current conditions by always choosing the most suitable mechanism. We evaluate our system against two mesh-based and one tree-based overlay, showing that the transition-enabled system outperforms them under all environmental conditions. Even more important, it combines their strengths to provide consistently high performance, even during flash crowds and under high churn. Our work motivates the design of transition-enabled systems and thereby utilize the strengths of a multitude of mechanisms, rather than trying to build a single, *one-size-fits-all* solution.

Contents

List of Figures

List of Tables

1 Introduction and Motivation

Video streaming is the fastest growing application of today's Internet, with prominent examples being YouTube[1] and Netflix[2]. Netflix recently reached more than 27 million subscribers for its streaming plans[3]. But not only the number of subscribers increases rapidly. The traffic generated due to the demand in high quality video streaming already reaches about 50% of the overall downstream traffic in the United States [San12, Cis11], with Netflix accounting for roughly 20% [San11]. To provide a streaming service at that scale, Netflix negotiates contracts with Content Distribution Network (CDN) providers and recently announced that it builds its own dedicated CDN, *Open Connect*[4], in order to lower the operating costs.

However, besides CDNs, there exist a second approach to content delivery: Peer-to-Peer (P2P) systems. In a P2P system, consumers act as service providers at the same time. Thereby, the resources of each consumer become part of the infrastructure of the service. This enables better, and cheaper, scalability with the number of users – an important property for a video streaming system, as the number of users tends to vary significantly over time. This variation is due to the fact that streaming a video is a *foreground* application, where the user actively consumes the stream. Thus, the usage of the streaming service depends on the habits of the consumer. For example, a live streaming service for movies will see significant increases of concurrent users during the evening hours, as shown by various measurement studies of existing solutions [VGNL10, HLL+07, LXK+07]. A live streaming system does not only have to deal with this diurnal pattern of usage. Events in the real world can trigger significant and spontaneous increases in the number of consumers, for example after natural disasters or on special occasions, for example on New Year's Eve.

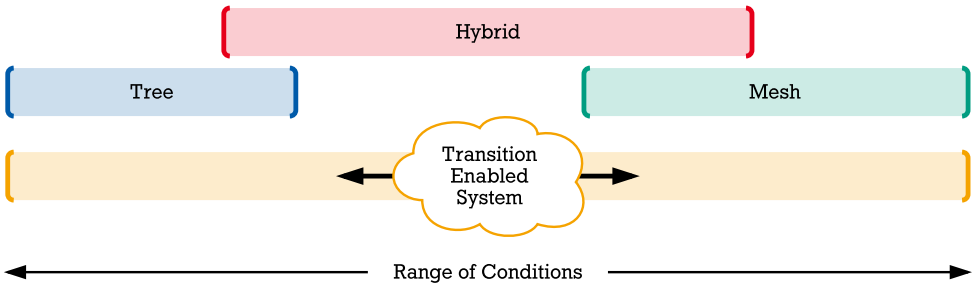


Figure 1.1.: Different mechanisms have different ranges of conditions, under which they exhibit their best performance. A transition-enabled system is able to switch to the mechanism that performs best in the current situation, thereby operating well under a very broad range of conditions.

Considering the plethora of streaming systems proposed by the research community over the last years, it remains an open question, which system design performs best under these highly variant conditions. While some authors argue that the mesh-based approaches offer superior performance [MRG07], others come to the conclusion, that multi-tree systems cope better with real-time requirements [SZFNR08].

1 `http://youtube.com` [Accessed 06/11/2012]
2 `http://netflix.com` [Accessed 06/11/2012]
3 `http://ir.netflix.com` [Accessed 06/11/2012]
4 `http://blog.netflix.com/2012/06/announcing-netflix-open-connect-network.html` [Accessed 06/11/2012]

Even worse: there is evidence that no design is able to achieve low overhead, low playback lag and high continuity at the same time [NBD09]. This uncertainty has caused a shift towards hybrid streaming overlays, as described by Shen et al. [SLZV11]. Those overlays try to combine advantages of the *primitive* solutions by avoiding some of their drawbacks. However, a hybrid system induces additional complexity and overhead, and it is often optimized towards a specific target application scenario. This results in different systems outperforming each other depending on the conditions. Figure 1.1 illustrates this problem: the mesh-based, tree-based, and hybrid systems each span a given range of conditions, under which they perform in a desired way. This range is indicated by the brackets, and ranges of systems might overlap, as both systems perform well in that particular conditions.

The basic idea presented in this thesis is rather simple: instead of trying to enlarge the areas one of those systems can span, why not just switch between different systems? This seamless switching between systems, or mechanisms, is called a *transition*. By allowing transitions between mechanisms, we should be able to span the whole range of conditions, as we can always choose the mechanism that performs best under the given conditions. Figure 1.1 shows this concept in a very simplified way: the transition-enabled system at a given point in time might operate under the range of conditions indicated by the cloud. However, as soon as the conditions change, the mechanisms in the transition-enabled system are changed as well. Thereby, the system changes the range of conditions it exhibits optimal performance in. This happens over the full range of conditions spanned by the *primitive* mechanisms, as indicated by the orange brackets. In this thesis, we propose a system that allows *seamless transitions* between different mechanisms, thereby allowing the system to utilize the mechanism that exhibits best performance under the current conditions.

1.1 Goal and Contribution

The goal of this thesis is to evaluate possibilities for transitions in P2P video streaming, with a focus on the overlay topology and the scheduling mechanisms. Therefore, a streaming system allowing seamless transitions between the aforementioned mechanisms is to be designed and implemented. The system and the impact of transitions under various scenarios is to be evaluated through simulative studies. In order to reach the described goal, the following contributions are stated in this thesis:

- An extensive literature survey on existing P2P live streaming systems, with a focus on design decisions and key optimization goals. This survey helps in determining key design aspects of a P2P live streaming system and, thus, in the identification of transition possibilities.
- The design of a transition-enabled P2P live streaming overlay, allowing transitions between tree-based and mesh-based topologies and scheduling schemes. Furthermore, the design includes the possibility to evaluate the transition-enabled system against the pure mechanisms under the same conditions.
- The realization of the proposed system design in Java for the simulator PeerfactSim.KOM, together with a visualization and an analyzing framework.
- An extensive simulative analysis of the implemented system and a comparison of the transition-enabled system against pure tree-based and mesh-based solutions. Thereby, insights into the benefits of transitions in P2P streaming systems are gained.

1.2 Outline

The remainder of this thesis is structured as follows. In the next section, relevant background on video codecs, content delivery and P2P video streaming is given. Afterwards, an extensive overview and analysis of existing P2P video streaming solutions is presented. The analysis of the related work presented in Chapter 3 serves as foundation for the requirements defined in Chapter 4. Based on these requirements, the design of the transition enabled streaming system TRANSIT is detailed in Chapter 5. We discuss the implementation of the proposed design for the P2P simulator PeerfactSim.KOM in Chapter 6. How the requirements are met by the proposed design is then evaluated in Chapter 7. Finally, in Chapter 8, we derive conclusions on how transitions in P2P video streaming systems benefit the system performance and pose interesting questions for future work.

2 Background

To motivate this thesis, relevant background on video streaming in general is given in this section. Video streaming is considered the *next big thing* on the Internet. Already today, streaming applications are responsible for around 50% of the overall downstream traffic on mobile and fixed links in the United States [San12, Cis11]. Streaming enables users to watch content while its download is still in progress, ideally with a very low startup delay and no further interruptions such as playback stalling. This is very similar to *progressive download*, where the user is able to start playback while still downloading the media file. However, a progressive download utilizes as much bandwidth as possible to download the full file, regardless of the playback position. Its sole purpose is to provide low playback delay. When streaming, on the other hand, the user only downloads parts of the file that are required near the current playback position. This leads to a smoother bandwidth utilization over the complete playback time. From a user's perspective, both approaches provide a low startup delay when compared to the download of the whole media file, as in traditional file sharing scenarios. To support streaming, the video has to be encoded in a way that allows playback even if the full file is not available during either the encoding process (live video) or the decoding process (streaming in general). In the following section an overview of recent coding techniques is presented, with a focus on their support for live streaming and heterogeneous playback devices.

2.1 Video Codecs for Streaming

Currently, there are several approaches to support heterogeneous devices from a video encoding point of view. One approach is to encode the content with different bit rates and serve each bit rate as a separate stream, as e.g. done on YouTube nowadays. The television would stream the highest available bit rate over the digital subscriber line, whereas the mobile device could stream a version with a low bit rate. This simultaneous sending, or *simulcasting*, of content adds significant load to the content servers, as the required bandwidth is the sum of the single streams' bit rates. Even if a modern codec such as Advanced Video Coding (AVC) [ITU10] is used, this results in significant load on the streaming server, as each copy has to be stored and streamed separately. Therefore, other codecs exits that support decoding a single video stream at different bit rates. This enables a streaming service to provide just one copy of the video file that contains multiple different quality versions of the video. Two such approaches are presented in the following: Multiple Description Coding (MDC) and Scalable Video Codec (SVC).

2.1.1 Multiple Description Coding

Multiple Description Coding (MDC) encodes a segment of data into multiple *descriptions* that can be decoded independently of each other. The more descriptions are available, the better the resulting quality is with respect to the original segment. The encoding process introduces overhead in total file size when compared to a single layer codec such as H.264/AVC. However, it enables receivers to deal with packet loss by playing the segment with a lower quality resulting from the subset of descriptions that have been received. In a live video streaming system, retransmissions are usually avoided as they would

introduce additional playback delay. Considering a P2P system that uses multiple paths in the overlay to disseminate packets, MDC would enable the overlay to deal with packet losses on a subset of those paths. At the cost of higher streaming traffic, those overlays tolerate moderate packet loss to lower the maintenance overhead and achieve higher reliability in the presence of churn [PWC03].

The exact functionality of a MDC encoder is beyond the scope of this thesis, the interested reader is referred to [Goy01] for an overview of MDC as well as to [WRL05] for an in-depth description of MDC in the context of video streaming. Akyol et al. [ATC07] propose a framework that allows a MDC encoder to adapt to changes in the network by varying the number of descriptions, which results in an altered level of redundancy. Thereby, the amount of overhead can be reduced in networks with high reliability as also less descriptions are required to generate a high quality version of the stream. However, the authors only consider a single receiver system, which poses the question whether such an approach is applicable in a large-scale P2P network as well. In such a large-scale network, packets would travel over multiple hops and link quality might vary significantly for each of those hops. Thus, it might be difficult to find the sweet spot for the amount of redundancy that should be introduced by MDC to cope with packet loss. To fully utilize the advantages of MDC, descriptions of a segment of data should always travel on diverse paths through the network. This is the motivation for multi-tree overlays that will be introduced in Section 3.1.1.

MDC in its core is not a coding scheme specifically designed for video streaming, but rather a generalized approach to support transmissions over unreliable networks without the need for retransmissions. Therefore, video streams encoded with MDC lack some functionality when it comes to the support for heterogeneous devices. Consider a video streaming system, where end users are able to receive a stream on their high definition television at home as well as on their smartphone while on the go. At home, the user prefers to receive the stream with a high resolution and overall perceived quality, as described in Section 2.2. On the go, a lower resolution might be sufficient as well, especially considering the limited bandwidth of the mobile communication infrastructure and the anyway smaller screen resolution of mobile devices. In both cases the user prefers not to experience interruptions in playback or high startup delays. MDC enables playing the stream at a lower bit rate by just dropping some descriptions of a segment. However, MDC does not allow to specify desired characteristics, such as the resolution that matches the screen size of a mobile phone. MDC only provides scalability in the Signal-to-Noise Ratio (SNR) domain. Dropping descriptions on purpose (and even worse: selecting those descriptions in a deterministic way) lowers the probability for other receivers to gather a sufficient amount of independent descriptions. This, in turn, affects the playback quality of the whole video streaming system. It is important to stress that MDC in the context of video streaming is designed to cope with packet loss and not to support heterogeneous receivers.

2.1.2 Scalable Video Coding

The Scalable Video Codec (SVC) [SW08] has been specified as an extension to the H.264/AVC [ITU10] video codec standard. SVC encoded content can be decoded at different quality layers, depending on the encoding settings. Thereby, only parts of the full bitstream are needed to encode the video at a lower quality. However, in contrast to MDC, all layers below the intended layer are needed to encode the video. A SVC encoded video stream can offer these discrete quality layers in three dimensions. To illustrate this concept, we model a SVC encoded video as a *video cube* as depicted in Figure 2.1. The three dimensions of scalability in the SVC stream are the *temporal*, *spatial* and *quality* resolution. Temporal resolution is the framerate at which the video is played, for example at 24 frames per second. Spatial resolution describes

the size of an image of the video in pixel. Quality resolution is determined by the SNR, i.e. the grade of compression of the image. These three dimensions are depicted by the axes in Figure 2.1. The stream at a given layer is illustrated as a cube at the corresponding position in the tree-dimensional space.

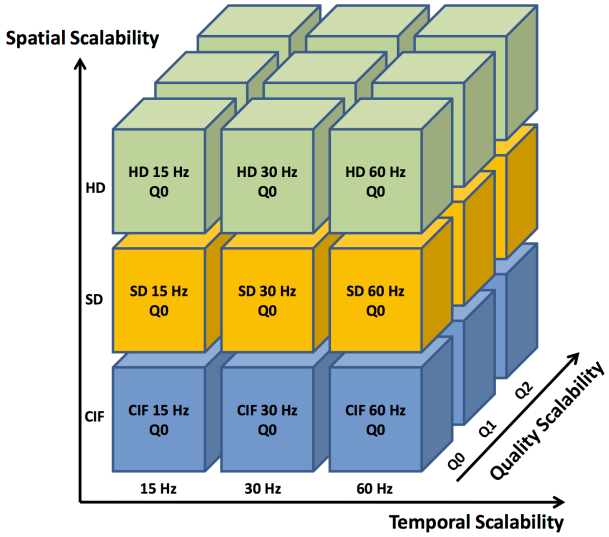


Figure 2.1.: Cube model of a SVC encoded video (adapted from [AZP+11]).

For easier reference, we address individual layers by their position in the SVC cube. A layer l out of all possible layers L is therefore addressed by its spatial (d), temporal (t), and quality (q) dimension as $l = \{d, t, q\}$. Consider for example the layer at the bottom left of the cube. This particular layer is called the *base layer*, as it contains the video stream at the lowest possible bit rate. This layer is needed to decode all other qualities of the stream. According to the aforementioned definition, the base layer is addressed as $l = \{0, 0, 0\}$. For this thesis, the exact format of the bitstream of a SVC encoded video is not of interest. Instead, we consider a simple model as depicted in Figure 2.2, where the video stream is divided into segments, or *chunks*. Each of those chunks consists of *blocks*, where each blocks belongs to exactly one SVC layer. The segment of the video can be played at a given quality layer l, if all blocks of this segment belonging to l and lower layers are available.

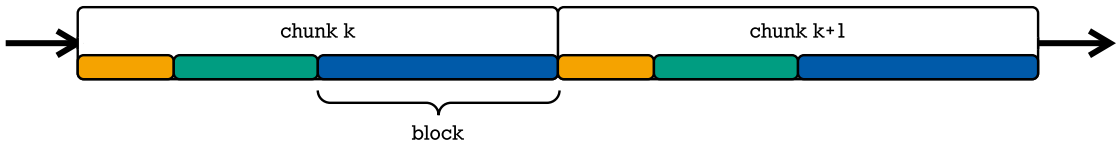


Figure 2.2.: Segmentation of a stream in chunks and blocks, where blocks hold information belonging to one specific layer, here illustrated by the colors of blocks. The depicted video stream offers three layers.

To provide a video at different bit rates, a content provider no longer needs to store multiple copies of the same video stream. Instead, the content is encoded using SVC, which results in a file that is on average 10 % larger than its highest quality AVC counterpart [SW08]. However, this single file contains the video at all layers – and thereby qualities – that were specified during the encoding process. AS the layers are configured during the encoding process, the content provider can select target bit rates and properties of the layers, such as the spatial resolutions contained in the stream. This allows consumers to receive a stream that is adjusted to their playback device or access link bandwidth. Consider YouTube,

were roughly 28 % of traffic is driven by smartphones and tablets [San12]. Here, SVC would allow the provision of layers for example at resolutions that match the device's screen size.

Even more interesting in the context of P2P streaming is the dependency on lower layers. As each interested viewer needs to receive at least the base layer, even peers that request only the lowest quality version of the stream have value they can provide to other peers. The native support for heterogeneous devices together with the availability to create substreams with different bit rates makes SVC the ideal choice for a P2P video streaming system. Such a system has to deal with heterogeneities not only in terms of device capabilities such as screen size and computing power, but also in terms of available bandwidth on the access link. By enabling each device to consume at least part of the stream issued by the source, all peers can participate in the overall content delivery. Before looking into the details of P2P-based video streaming, we need to define the term *quality* for later evaluation and comparison of streaming systems.

2.2 Quality of Service and Quality of Experience

The success and acceptance of any P2P system depends on the user satisfaction the system is able to achieve. How to satisfy a user depends on the service the system offers, but there are a number of metrics that contribute to the overall satisfaction and are common among most applications [KGK+08]. Those include classical Quality of Service (QoS) metrics such as *success*, *delay* and *recall*, which basically specify *how often* the system is able to fulfill its task, *how much time* is needed and *how complete* the task is fulfilled. Those metrics depend on network primitives such as latency and packet loss rate and give a general overview how well the system works from a functional point of view.

However, applying those metrics to a video streaming system does not necessarily lead to a good understanding of the perceived quality from a user's perspective, according to Winkler et al. [Win07]. Here, we have to consider additional metrics such as startup delay and amount as well as duration of stalls during playback. But even those metrics can only provide a hint on how satisfied the user is with the service. Measuring user satisfaction in terms of an evaluation of video quality by the user requires extensive studies and furthermore differs depending on the content itself and the environment [SM10]. In most cases, an extensive user study is not feasible when developing a video streaming application. Therefore, there is a high interest in automated metrics, so called *objective Quality of Experience (QoE) metrics*, that are able to, at least, estimate user perception to some extend.

One such *objective* QoE metric is the Video Quality Metric (VQM) proposed by Pinson et al. [PW04]. As the name suggests, the metric is intended to be usable with a broad range of codecs. It has been evaluated independently through extensive user studies which show a strong correlation between the score generated by VQM and the subjective score resulting from the studies. The exact workings of this metric are beyond the scope of this thesis, as we focus solely on objective metrics to gain an understanding of the quality delivered by the system. Rückert et al. [RAZ+12] propose a system that adapts the streamed SVC layer based on the estimated QoE, rather than on QoS metrics such as bit rate. As SVC layers with higher QoE do not necessarily imply higher bit rates, the overall bandwidth consumption of the system is decreased. At the same time, the performance in terms of playback delays and stalls is enhanced. To achieve a high QoE and, thereby, good user satisfaction, a video streaming system has to be carefully designed to cope with message loss, the heterogeneity of playback devices and limited capabilities of the involved infrastructure. First, traditional approaches to content delivery over the Internet are presented, as they motivate the design of a P2P-based solution.

2.3 Traditional Content Delivery for Streaming

When thinking about traditional content delivery, the television broadcast system comes to mind. In its simplest form, content is sent from an antenna to everybody within reach: it is *broadcasted*. However, when sending media content over the Internet, broadcasting is not applicable. Here, content should only reach those users that previously expressed their interest in the topic. In the context of a communication system, this is referred to as *multicasting*. Multicast can be seen as a publish/subscribe system where interested users join a group and receive updates targeted at the specific group. In the context of live video streaming, multicasting is the core communication paradigm as the video is to be transmitted from one source towards multiple receivers. This section explains Internet Protocol (IP) multicast and Application Layer Multicast (ALM) with a focus on their applicability for video streaming. A special case of ALM systems used in today's centralized video streaming portals such as YouTube are Content Distribution Networks (CDNs), which are briefly covered as well.

2.3.1 Internet Protocol Multicast

The Internet Protocol (IP) itself includes support for multicast groups. While this would be far superior in terms of bandwidth utilization when compared to other approaches (e.g. application layer multicast), it has never been widely deployed due to the added complexity in routers and the limitation of the address space for multicast groups. While IP multicast might be deployed inside an Autonomous System (AS) for example to support Internet Protocol Television (IPTV), it showed to be not applicable to Internet-wide streaming applications. Nevertheless, key design decisions in IP multicast inspired many application layer systems, and are, therefore, presented in this section.

First of all, IP multicast is a multi-source multicast. Each participant in a group is able to send data to all other members of the group. While this is a prerequisite for collaborative applications such as video conferencing, it is not a necessity in the context of video streaming. Multicast groups in IP multicast are addressed with a reserved range of IP addresses. If a node wants to join such a group, it registers with the respective IP address. The group itself is fully managed by the routers between the participants, which together form a spanning tree. Packets are duplicated at the routers only if needed, which results in superior bandwidth utilization. This advantage in terms of bandwidth requires additional router complexity, as each router has to maintain its associated group members and coordinate this information with other routers, which in turn violates the state-less principle of IP routers. To add features such as billing and membership control to an IP multicast, one needs to deploy additional protocols on all intermediate systems. This becomes nearly impossible outside the scope of a single Internet Service Provider (ISP). Diot et al. [DLL+00] provide an extensive overview of the problems with today's implementation and deployment of IP multicast. They come to the conclusion that IP multicast does not satisfy the requirements of ISPs and will, therefore, not see wide adoption outside an AS.

While IP multicast might not be feasible for deployment outside the domain of a single ISP, it is often used for services that are only provided to customers of a single ISP. One prominent example are the aforementioned IPTV services, where users receive the television program via their subscriber line. As the traffic for the IPTV service never leaves the network of the ISP, utilizing IP multicast is a more efficient solution than to deploy ALM solutions. Nevertheless, countless other services span multiple ISPs and therefore need to circumvent the limitations of IP multicast.

2.3.2 Application Level Multicast

To overcome the deployment issue with IP multicast, multicasting in most of today's applications is shifted to the application layer. The application defines a protocol which is used by participants to organize data forwarding. While in an IP multicast packets are duplicated by the routers, in ALM packets are duplicated by the hosts. Therefore, ALM is often also referred to as End System Multicast (ESM). Communication between hosts does only involve standard IP unicasts, which does not imply any additional requirements on intermediate systems such as routers. However, when compared to IP multicast, an ALM adds additional traffic overhead, as packets might need to traverse the same link multiple times.

ALM fits into the definition of a P2P system, as nodes in the ALM contribute resources such as upload bandwidth and processing power to each other. Thus, construction of a multicast system might be centralized or fully distributed, depending on the application scenario. One prominent example for a general purpose ALM system on top of a Distributed Hash Table (DHT) is Scribe [CDKR02], which does not require any centralized elements. It is important to stress, that in a real application scenario, an ALM does not necessarily need to involve end-users as hosts. Instead, end-users might only connect to a participant in the ALM and receive their content in a pure client/server fashion. In this case, multicasting is used in the *back-end* of a system. Consider a Content Distribution Network (CDN), where a content provider adds new content. In the back-end, the CDN will move the content towards the edge of its network, i.e. towards the end-user. One possibility to do so might be the deployment of an ALM system where the CDN nodes form a multicast group. Such an approach is later on presented in Section 3.1.3.

An additional difference between IP multicast and ALM lies in the used topologies. IP multicast always creates a spanning tree to support multi-source multicasting, whereas an ALM can create any network topology. However, to prevent duplicates and unnecessary transmission, most systems try to maintain a delivery tree for the dissemination of packets. Live video streaming requires a high bandwidth and low latency from the source to all participants, which further influences the design of live streaming systems. The general characteristics of P2P streaming systems are presented hereafter. Existing solutions in the field of P2P live streaming are then presented in Chapter 3.

2.4 Peer-to-Peer Streaming

Before presenting existing solutions in the field of P2P-based video streaming, some additional background is given regarding the characteristics of such systems. As already mentioned in Section 2.3, media streaming on an Internet-wide scale is done at application level. P2P-based live video streaming systems are a subset of ALM systems, in that they distribute content form a source by implementing the multicast-behavior at an application level. In contrast to traditional ALM systems, a P2P-based system includes the end user in the overall delivery scheme. Thereby, users contribute their capacities in terms of processing power and upload bandwidth to distribute the content they receive to other users. This requires a coordination of the participating users, or *peers*[1], as it has to be ensured that each peer is able to retrieve all required packets to decode the video stream. Existing approaches in P2P video streaming can be classified as tree-based, mesh-based or hybrid, according to their overlay topology [LGL08b].

[1] The terms *peer* and *node* are used interchangeable in this thesis.

2.4.1 Tree and Multi-Tree Topologies

Tree-based systems closely resemble the design of IP multicast, in that they form a tree originating from the streaming source. A tree structure similar to IP multicast is highly desirable when distributing content, as it provides low latency and in-order delivery of packets, at least in an ideal case. Users interested in the stream join the tree at a position that is determined by the overlay protocol. The source node then sends packets to its children, which in turn forward them to their own children, and so on. Due to the in-order delivery of packets pushed down the tree, there is no complex scheduling mechanism involved. Instead, each node just forwards every packet it receives to its children. This is referred to as *push-based* transmission coordination. With a higher fan-out at intermediate nodes, and thereby a smaller tree height, the playback latency for the leaf nodes decreases. However, due to limited upload bandwidth at those nodes one has to find the right trade-off between latency and bandwidth utilization.

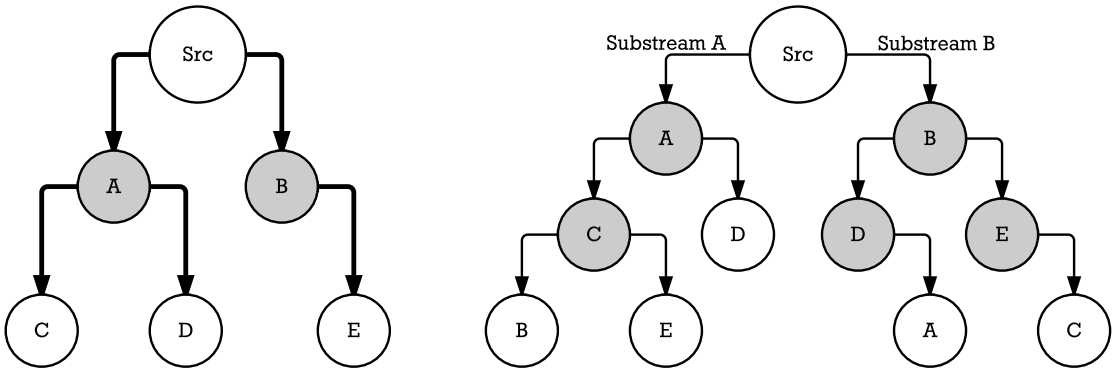


Figure 2.3.: Single-tree and multi-tree based content dissemination overlay with five distinct peers. Grey peers contribute upload bandwidth in the respective tree.

Under the idealized assumption that all nodes are reliable and do not leave the system while the source is streaming, nearly no overhead is induced by the overlay. In this case, data paths are considered stable. In reality, node churn might have a high impact on the performance of tree-based systems. The tree structure has to be maintained and actively repaired if a node leaves the system which might induce a significant maintenance overhead. If an intermediate node leaves the tree, all nodes down the tree no longer receive the stream from the source node. To guarantee uninterrupted playback, nodes have to detect such failures in order to reconnect to the healthy part of the tree. This also requires nodes to cache packets up to some extent in order to provide them to nodes whose parent just left. If detection and repair of broken links does not work reliable and fast, nodes further down the tree might experience interruptions in their playback as they are disconnected from the data flow. As soon as nodes are subject to churn, the protocol complexity grows in order to ensure timely and reliable delivery.

To increase resilience against churn, recent streaming systems maintain multiple trees, often in conjunction with a coding scheme such as MDC. The difference between a single-tree and a multi-tree system is shown in Figure 2.3. Both overlays consist of a streaming source `S` and five consumers, `A` to `E`. In the single-tree based system, only `A` and `B` contribute upload bandwidth. In the multi-tree system depicted on the right hand side, the source splits the stream into two substreams. A tree is maintained for each of those substreams, with the substream bit rate ideally being half of the stream's bit rate. A node is placed as an intermediate node in only one of the trees, whereas it is a leaf node in all other trees. This

ensures, that each node can contribute some upload capacity to the overall system throughput which makes it fairer than the single tree approach where leaf nodes do not contribute any upload capacity. Together with the aforementioned MDC video encoding, this offers the possibility to distribute subsets of the video descriptions over distinct trees. The stream can still be decoded if a tree fails, which could improve reliability of the system under higher churn. Therefore, the advantage of the multi-tree solution over a single tree is the better utilization of peer resources and the higher stability.

Most tree-based systems for video streaming that are presented in Chapter 3.1.1 require a (at least partly) centralized approach for tree maintenance and failure recovery. Most tree-based systems rely on the streaming server and, thereby, the root of the tree to provide this service. It is a reasonable assumption that the maintenance overhead is small in terms of bandwidth requirements when compared to the actual video payload. Nevertheless, if the level of churn increases, this centralized approach may become the bottleneck of the system. If nodes have to switch their parents too frequently, push-based data delivery becomes prohibitive, as nodes would anyhow have to request missing data from their parents. Here, mesh-based systems promise high reliability even in the presence of churn.

2.4.2 Mesh Topologies

In a mesh-based system, peers form short-lived connections between each other. They exchange information about their current status, i.e. the availability of video content, and request missing data based on this information. The receiver-driven requesting of video content is also known as *pull-based* scheduling. To gain knowledge of new peers in the system, peers periodically exchange information about their current neighborhood. If a peer gets to know a new contact in the process of this neighborhood exchange, he might initiate a new connection. Figure 2.4 shows the overlay topology of a mesh-based system at two distinct times. Dotted lines denote connections, where the shade of the node indicates its degree, i.e. the number of neighbors. The mesh in this example is bidirectional, with payload being exchanged in both directions. In most mesh-based live streaming systems, peers only open incoming connections and do not actively initiate outgoing connections.

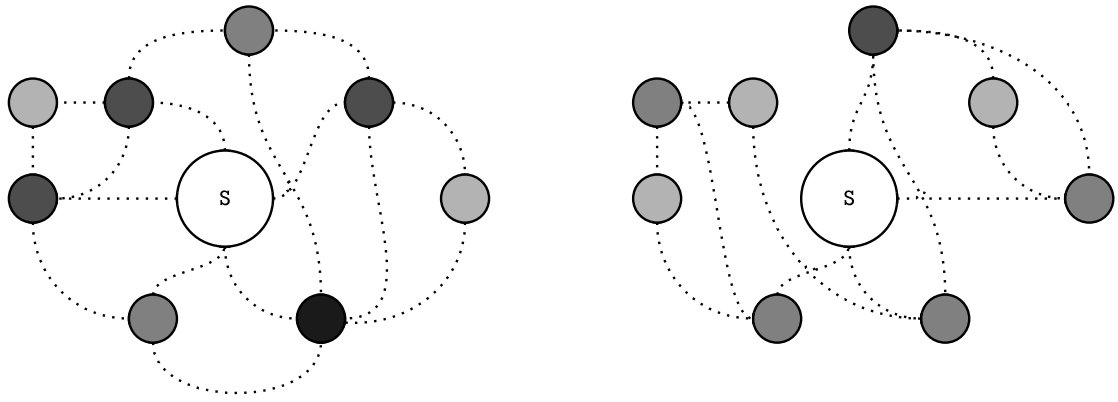


Figure 2.4.: Two snapshots of a mesh-based overlay. Connections are formed and discarded over time, darker peers have more connections.

Peers in a mesh tend to stay connected as long as the connection can be used for the transmission of video blocks. If the source of the connection has no packets to offer that are of interest for the receiver, he will most likely close the connection and initiate a new one to another peer. The age of a connection

does not influence the pull-based scheduling scheme, as content is advertised periodically. If a block has not been delivered due to a node failure, it is just requested from another neighbor that advertised its availability beforehand. Due to this behavior, mesh-based systems are inherently more robust to churn than tree-based systems. However, the pull-based scheduling introduces additional message overhead and higher latencies.

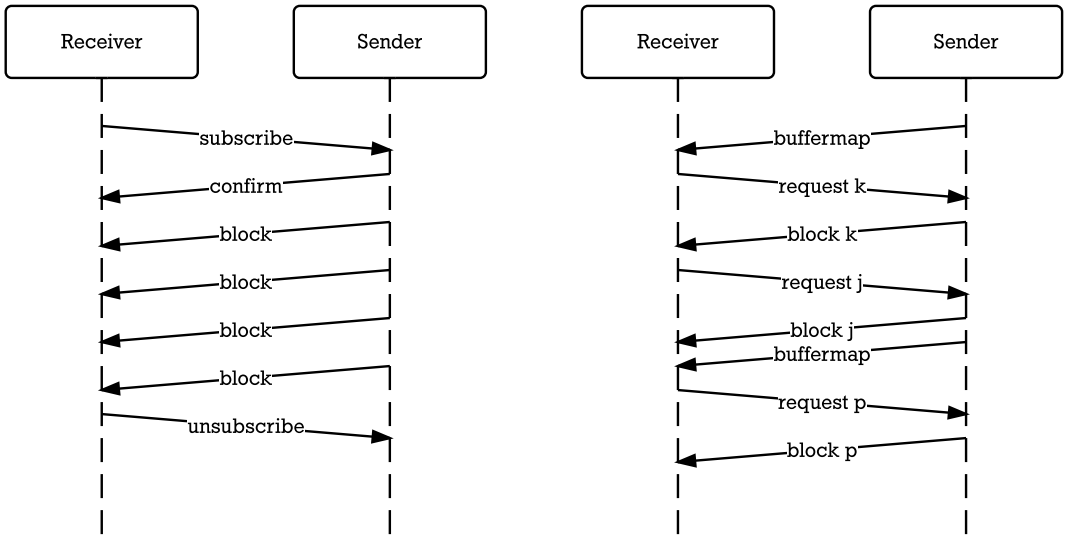


Figure 2.5.: Message sequence charts for push-based (left) and pull-based scheduling schemes (right).

Both scheduling schemes are shown in Figure 2.5, with push-based scheduling on the left. Here, the receiver sends a subscribe-message to the sender and will then receive a confirm-message followed by new packets as soon as they become available at the sender. If the receiver no longer wants to receive packets from the current sender, he just unsubscribes. On the right hand side pull-based scheduling is shown, where the soon-to-be sender advertises its current buffer contents by sending a buffermap-message. The receiver uses this information to request blocks that are missing in its own buffer. For each block, a new request-message is issued. After some time, the sender transmits an updated buffermap containing recently gathered packets. In the following chapter, different realizations of P2P streaming systems building upon these basic schemes are shown.

3 Related Work

In this section, existing solutions in P2P-based and assisted video streaming are presented. Following the conceptional introduction in Section 2.4, systems are categorized by their overlay topology as either tree-based, mesh-based or hybrid. Properties of the systems as well as key design decisions are analyzed and compared. A key component of any P2P video streaming system is the scheduler, with different methods from the related work being presented in Section 3.2. Finally, related work on topology transitions in P2P systems is presented in Section 3.3.

3.1 Peer-to-Peer Video Streaming Systems

In recent years many systems for P2P-based or peer-assisted live video streaming have been proposed. In this section, some representative systems are presented and challenges are discussed. Finally, the presented systems are compared regarding their key characteristics, which further motivate the design choices taken later on in Chapter 5.

3.1.1 Tree-Based Systems

In this section, tree-based streaming systems are presented, both single-tree and multi-tree variants. As already mentioned in Section 2.4, tree-based systems deploy a content delivery strategy that closely resembles IP multicast.

Overcast – Single-Tree Multicast

One example of a single-tree multicast overlay is OVERCAST [JGJ+00], which promises an overall system throughput comparable to IP multicast with a reasonable overhead. OVERCAST creates and maintains a high-bandwidth distribution tree from a source to multiple participants. Therefore, it utilizes a tree algorithm that tries to maximize the bandwidth from the root towards each node. This is achieved through low fan-out at intermediate nodes, thus leading to higher trees with higher latency for the leaf nodes. OVERCAST was not designed for time-critical live streaming such as television broadcasting, but as a way to distribute content to a large group of recipients that can tolerate modest delays of up to fifteen seconds. Its primary goal is to distribute content closer to the end users, that do not participate in the overlay structure but instead access content on one of the OVERCAST nodes by standard Hypertext Transfer Protocol (HTTP) requests. Therefore, the system can be compared to a CDN, as requests from end users are transparently redirected to the OVERCAST node that is best suited in terms of available bandwidth and network proximity.

Within OVERCAST, participating nodes are assumed to be dedicated machines that show stable behavior, i.e. there are no frequent joins or leaves. This allows the protocol to put more effort into tree optimization for high bandwidth rather than ensuring high churn tolerance. New nodes are placed at a position in the tree that is as far away as possible from the source, while still maximizing the delivered bandwidth for the node. Latency is not an issue within OVERCAST, as nodes act mainly as a content cache. Trees in OVERCAST therefore tend to be high with only a few outgoing links at every node in order to ensure

high system throughput. As the underlying network links may change their characteristics over time, nodes evaluate their position in the tree by probing the links to their parents and grandparents as well as their siblings on the same level. If a sibling offers the same bandwidth towards the root as the current parent does, a node will chose its sibling as the new parent, thereby making the tree higher. If the node probes its grandparent and detects that the current parent is limiting the bandwidth, it will move up in the tree and become a sibling of its former parent. These mechanisms allow OVERCAST to balance the overall bandwidth availability as close to the root's maximum bandwidth as possible. Furthermore, this mechanism enables nodes to maintain a connected tree and rebalance the system if a parent fails.

OVERCAST can be compared to a CDN to some extend, with nodes acting as content caches and proxies for end users. If an end user requests content, its request is directed to the root node. The root node then has to redirect the request to a node with sufficient resources, preferably close to the end user. Therefore, the root node has to maintain an up-to-date view of the current tree structure as well as the load of the nodes. OVERCAST introduces the *Up/Down* protocol in order to provide this information to the root node.

As first step of the *Up/Down* protocol, children ping their parents periodically. If a child does not ping its parent, it is assumed that it failed and left the network. This information is stored in a so called *death certificate* that is then aggregated with other such death certificates over a given time interval. At the end of each interval, the aggregated certificate is forwarded up the tree until it finally reaches the root node. Likewise, if a node joins the network, this is reported in a *birth-certificate*. Each node in the tree is therefore able to maintain a complete view of all nodes below it in the tree. Besides these certificates, one might add additional information that could for example be used for monitoring the system. However, the protocol can only be used for information that does not change rapidly, as otherwise the root node might get overloaded. The protocol allows the root node to redirect end user requests as well as join requests based on current information about the topology and node state.

The usage of standard Internet protocols such as HTTP makes OVERCAST easy to deploy on an existing infrastructure when compared to IP multicast. In contrast to IP multicast, OVERCAST only offers single-source multicast. In the context of video streaming systems this is sufficient. For applications where other participants contribute content every now and then, it might be easier to send this content to the root for further distribution rather than to maintain multiple trees, that would otherwise be necessary.

While the approach taken in OVERCAST achieves a high bandwidth between source and clients, its high trees increase the delay between the source and nodes further down the tree. For a video streaming overlay, especially in live streaming, one tries to achieve high bandwidth and low delay at the same time. This motivates the design of multi-tree streaming overlays such as COOPNET, which is presented in the following section.

CoopNet – Peer-to-Peer aided Video Streaming

COOPNET [PWCS02, PWC03] is an early approach in P2P aided video streaming. It does not replace the centralized streaming solution but instead supplements it to provide further bandwidth for example during a flash crowd. As clients may join and leave frequently, it is assumed that the overlay is inherently unreliable. Therefore, the authors utilize the video codec MDC to tolerate packet loss and, furthermore, construct multiple trees over which different descriptions of the same video segment are disseminated. As already discussed in Section 2.4, maintaining multiple trees helps to better utilize each node's upload bandwidth and to be more resilient towards churn. To reduce the complexity of tree management and maintenance, the authors propose a centralized solution. The video streaming server as source of all

trees maintains the topology and handles node churn. The overhead for the server in terms of bandwidth is low compared to the load that would arise if all clients were connected directly to the server.

The centralized coordination enables CoopNet to create balanced trees and maintain this property even under node churn. A balanced tree ensures low latency for all participants, especially if paths in the tree are optimized with respect to the underlying network. However, the authors stress that MDC does only work efficiently on diverse distribution trees which may interfere with the aforementioned network awareness. As the tree construction is centralized, it is ensured that a node has exactly one parent in each distribution tree. Thus, the trees are diverse as long as the assigned parents do not share a common path towards the source node. While the source does not enforce this property, its selection mechanism is random to some extent which aids in the construction of largely diverse paths. The authors propose an optimization where parents are furthermore preferred if they are located in close proximity to the new node. This leads to rudimentary support of network awareness while at the same time exposes sufficient tree diversity.

The presented evaluation shows that the proposed system is able to significantly reduce load on the server during flash crowd scenarios. In combination with MDC, the multi-tree approach is robust even under high node churn, as coordination is centralized on the server. While in P2P systems a major objective is a fully decentralization of mechanisms, a centralized coordination scheme is not necessarily a problem in the context of video streaming scenarios, as the content itself is usually also dependent on a centralized source. If the source fails, no new content is available, even if the rest of the system is fully decentralized. As such, live video streaming is inherently dependent on a central entity which justifies the design choices in CoopNet.

SplitStream – Multi-Tree on SCRIBE

SplitStream [CDK+03] differs from other solutions in that it does not construct its own overlay and group management protocol. Instead, it utilizes a DHT and a group communication system on top of this DHT. The authors chose Scribe [CDKR02] on top of Pastry [RD01] but point out that any other DHT and group communication system would work as well. Group communication systems such as Scribe initially were not designed for large amounts of data as in the video streaming scenario. Therefore, the authors of SplitStream propose using multiple trees to disseminate parts of the whole video stream encoded with MDC, which translates to using multiple groups in Scribe.

The way Scribe multicast groups are designed in SplitStream leads to the property mentioned above: each node participating as an intermediate node in one tree is a leaf node in all other trees, assuming that a node only consumes one video stream at a time. The authors show that this property balances load across different nodes and provides better resilience against churn when compared to a single-tree system. However, the robustness of SplitStream depends solely on the underlying DHT and group communication system, as SplitStream itself does not provide additional means to recover from node failure. As Pastry optimizes its routing tables with respect to transmission delays, the resulting trees in Scribe show in average less than twice the delay of an IP multicast. Castro et al. compared the delay of multiple application level multicast systems to that of an IP multicast in [CJK+03]. The authors showed that Pastry outperforms the P2P system CAN [RFH+01] with respect to this metric, as CAN exhibits three times the delay of an IP multicast.

Stanford Peer-to-Peer Multicast

The Stanford Peer-to-Peer Multicast (SPPM) protocol [BNSG07] shares key aspects with the aforementioned overlays CoopNet and SplitStream. It also maintains multiple trees, but it does not require MDC

encoded content. Instead, if a packet is dropped during its push-phase, the child node issues a request to the parent. The parent will then retransmit the missing packet. If the parent does not react for a given time, the missing packet is requested from other contacts. SPPM has been specifically optimized for the H.264/AVC [STL04] video codec in that it assigns priorities to packets based on the type of information they carry. Without going into too much detail, the basic idea is that key frames of the video are transmitted with higher priority than other types of frames. According to the authors, this enables SPPM to continue playback even in the absence of some intermediate packets. Nevertheless, the quality of the stream will drop in case of packet loss that cannot be compensated with retransmissions.

SPPM has been deployed and evaluated on PlanetLab, a large-scale testbed for prototype evaluations. The authors show that during the evaluation, nodes participating in the overlay experienced short startup delays of around one second. Experiments were conducted using only 100 nodes. Nodes that joined the overlay stayed online until the end of the streaming process. Thus, it remains unclear how the system performs under heavy churn or higher load and more realistic scenarios. Recently, an extension of SPPM to support mobile devices has been proposed [NBH+09]. With this extension, a mobile device can connect to the overlay as a leaf peer only, i.e. it will not contribute any upload capacity. Furthermore, the video stream is transcoded on the stationary nodes in order to better match the capabilities of the mobile node in terms of screen resolution and bandwidth. Compared to a SVC based solution, this requires additional processing capabilities at the intermediate nodes.

With the prioritization scheme in SPPM, SVC becomes increasingly interesting. In [BSWG07], the authors deploy SVC instead of H.264/AVC and configure the scheduler of SPPM to prioritize packets of the base layer. This way, network congestion and node churn can be tolerated, as long as the base layer packets still reach each node. During the evaluation, the SVC based version of SPPM outperformed the single-layer version in presence of congestion and heterogeneous bandwidths. Even considering the higher encoding overhead of SVC as described in Section 2.1.2, its application promises a further improvement of the system performance under real-world conditions.

Besides the systems discussed so far, tree-based systems are the natural choice inside a CDN or for the IPTV system of a single ISP. In such systems, end users participate only as leaf nodes and the P2P network is built on highly reliable and manageable infrastructure. Existing systems showed that it takes a lot of effort to make a tree-based system robust in presence of high churn. This is reflected by the increasing complexity of repair mechanisms as well as added protocol overhead such as the periodic exchange of keep-alive messages, as shown in this section. For truly decentralized P2P video streaming without a dedicated infrastructure of reliable nodes, mesh-based systems are becoming increasingly popular. They are detailed and discussed in the following.

3.1.2 Mesh-Based Systems

In contrast to tree-based systems, mesh-based overlays maintain a rather flexible topology. Links between nodes are established and dropped as needed. This can result in a different dissemination path for each packet of the stream and in turn lowers the impact of a node departure on the system performance. However, orchestrating nodes in a way that ensures timely and reliable arrival of all packets requires additional overhead when compared to the tree-based solution. Nodes have to actively *pull* packets from their neighbors, which introduces additional delay when compared to the *push* scheme of tree-based systems. Furthermore, the exchange of buffer maps, i.e. information about the availability of packets

might be necessary, which introduces even more message overhead. Three systems for mesh-based P2P live streaming and their corresponding content dissemination strategies are discussed in the following.

CoolStreaming/DONet – Data-driven Live Streaming

DONET [ZLLY05] is a data-driven overlay for live video streaming, which means that its nodes exchange information about content availability and form connections based on this information. The video stream is segmented into packets and the availability of packets is advertised to neighbors by the exchange of so called *buffer maps*. In the case of live streaming, a node is interested only in the packets directly following its current playback position. Therefore, the buffer is limited by the current playback position and the newest packet available at the source. The buffer size furthermore directly affects the playback delay of nodes compared to the source. In DONET, delays of up to one minute are accepted, which results in a sufficiently large buffer for continuous playback. Buffer maps consist of one bit for each packet that fits into the current buffer plus the sequence number of the first packet in the buffer for alignment with other nodes. Due to the bidirectional exchange of buffer maps and packets, there is no defined parent/child relationship. Instead, nodes are *partners* in the overlay. This leads to the creation of an undirected mesh.

As shown by the authors, the efficiency of the approach depends on the scheduling algorithm. To ensure fluid playback, packets that are close to the playback position have to be requested with a higher priority than other packets. Furthermore, it has to be ensured that the bandwidth constraints of the streaming partners are met, as otherwise nodes might get overloaded. DONET tries to satisfy both conditions by always first requesting packets that are only available at one partner, thus maximizing their replication. Other packets are requested from partners that have enough spare bandwidth. To prevent unnecessary message exchanges, the request contains a map of packets that are to be sent, similar to the buffer map used to indicate available packets.

DONET deploys SCAMP [GKM01], a group membership service based on gossiping. Using SCAMP, nodes periodically send membership messages containing their ID and the number of contacts in the overlay. This information is then stored in membership caches on other nodes, allowing them to periodically evaluate their current partners and switch to other partners if that results in better performance. To deal with node churn, nodes periodically establish new partnerships by contacting random nodes out of their membership cache. Furthermore, node failures and departures are actively announced over SCAMP. If a node leaves gracefully, it issues a leave message that will in turn lead to a removal of the node from all membership caches. If a node fails, the failure is detected by one or more of its partners that will then issue the leave message on behalf of the failed node.

DONET was also evaluated on PlanetLab and made publicly available under the name COOLSTREAMING later on. The evaluation shows, that using SCAMP leads to roughly 1% of maintenance overhead. Furthermore, due to the gossiping approach taken with SCAMP, this overhead is nearly independent of the actual size of the overlay, as buffer maps are only exchanged with a subset of the neighborhood. When comparing DONET to a simple tree-based overlay, the authors observed that playback continuity is much better, without significant differences in playback delay. However, the tree-based overlay did not include means to create and maintain a balanced tree, which is an explanation of the high latency. In 2008, Li et al. presented a new version of COOLSTREAMING [LXQ+08], where the scheduling scheme was altered to allow for partial pushing of packets to achieve lower latency. This approach is very similar to PRIME, which is explained in detail in the next section.

Prime – Mesh-Based Live Streaming

In PRIME [MR09], nodes form a randomly connected, directed mesh to distribute content. The number of incoming and outgoing links ($deg_{in,i}$, $deg_{out,i}$) of a node i is determined by its available bandwidth such that the bandwidth per flow ($bwpf$) is roughly the same for all nodes. The authors require the *bandwidth-degree condition* for any two randomly selected nodes i and j to hold as

$$bwpf = \frac{bw_{out,i}}{deg_{out,i}} = \frac{bw_{in,j}}{deg_{in,j}} \quad . \tag{3.1}$$

If this condition is not satisfied, nodes experience a so called *bandwidth bottleneck*, where either the outgoing or the incoming bandwidth of a node is not fully utilized. Thus, satisfying the *bandwidth-degree condition* leads to a better overall system throughput, assuming that there is enough relevant content available at each node. In a live streaming application, relevant content consists of the packets around the current playback position of the stream. The limited number of packets might lead to a *content bottleneck*, where a node might have plenty of free bandwidth but no relevant content to distribute. To minimize the probability of content bottlenecks, in PRIME, packet delivery is divided into two phases, namely a *diffusion* phase and a *swarming* phase.

Diffusion Phase: When the source creates a new segment containing the most recent part of the live stream, packets of this segment are not available on any other node in the overlay. During the diffusion phase, these newly generated packets are disseminated in a way that ensures each node receives at least one packet of the segment. In PRIME, nodes actively push the information on their available packets to their neighbors, which then request missing packets depending on their own buffer state. Assuming that data dissemination over one hop takes place in an interval Δ, it takes at least $n\Delta$ for a packet to reach a node whose shortest path from the source consists of n hops. The value of n determines the *level* of a node in the PRIME overlay.

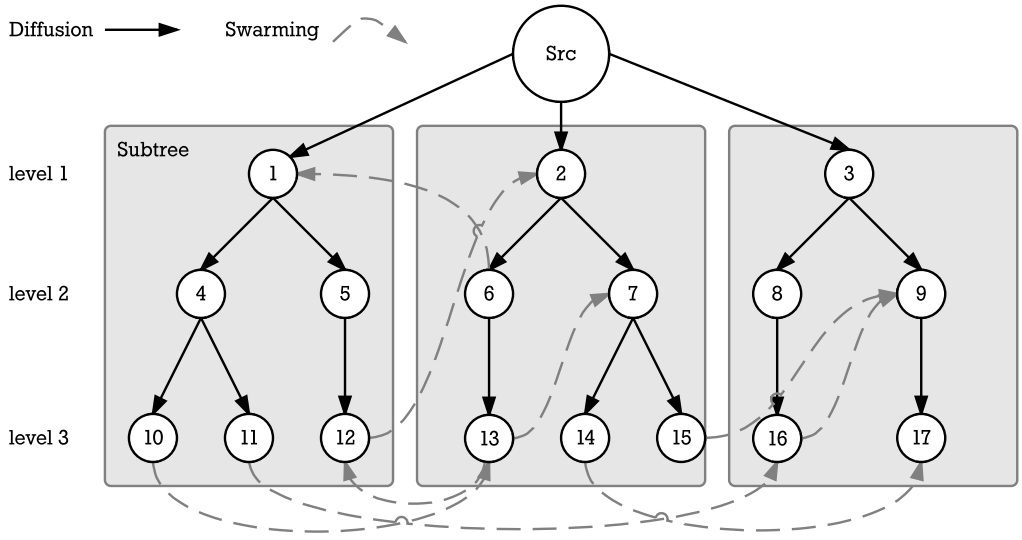


Figure 3.1.: Diffusion subtrees in PRIME (adapted from [MR09])

If a new packet is available at a node, all its neighbors will request this newly available packet. This leads to the implicit creation of *diffusion sub-trees* as shown in Figure 3.1, where new packets are propa-

gated from nodes on level n to their children on level $n+1$. The source node can control the distribution of a new segment by splitting the segment into different packets and sending each packet to a subset of its direct children. At the end of the diffusion phase at least one packet of the new segment is available at every node, as every node has at least one parent that is located on the shortest path towards the source. This ensures data availability during the swarming phase and helps preventing content bottlenecks.

Swarming Phase: After each interval Δ in the swarming phase, peers request missing packets of the segment from their remaining parents on the same or lower levels, as soon as they are available. Most of these *swarming connections* originate from nodes at the bottom level of a diffusion sub-tree, as those nodes did not utilize their outgoing connections during the diffusion phase. As a consequence, only parents in a different diffusion sub-tree can provide new data units directly after each interval Δ in the swarming phase. If a node has diffusion parents in the same sub-tree, they might not be able to provide a new segment. In such a case, additional swarming rounds for the segment are needed such that missing packets become available at one of the swarming nodes. The minimal number of rounds needed to ensure that the majority of nodes received all required packets of a segment is denoted as $K_{\min}$. The value of $K_{\min}$ has a direct impact on the required buffer size and thus the relative playout delay ω of peers to ensure fluid playback.

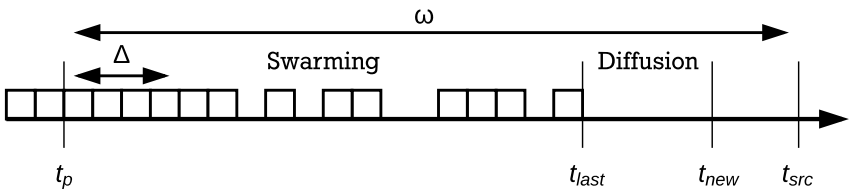


Figure 3.2.: Buffer state in PRIME (adapted from [MR09])

Figure 3.2 shows the state of a buffer, where t_{src} denotes the timestamp of the most recent packet at the source. All packets that arrive with a newer timestamp than the last received packet t_{last} are treated as diffusion packets, whereas missing packets for older segments are requested from swarming parents. The overall difference between the current playout time t_p and the most recent packet at the source is the aforementioned playout delay ω. The number of rounds needed during the diffusion phase is solely determined by the depth of the diffusion sub-trees. Together with the minimal number of rounds in the swarming phase, $K_{\min}$, the overall playout delay ω should satisfy the following condition:

$$\omega \geq (\text{depth} + K_{\min}) \quad . \tag{3.2}$$

While the diffusion phase of a segment resembles content distribution in a multi-tree system as presented in Section 3.1.1, it is important to note that diffusion sub-trees are not actively maintained. They are a result of the current neighborhood of a node in the mesh and as such subject to change. As the source continuously publishes new segments, diffusion and swarming of the respective packets take place in a sliding window determined by the current buffer state of nodes.

To cope with packet loss, PRIME assumes MDC encoded video streams to ensure playback of the stream even if not all packets arrived at all nodes in time. As mentioned before, the source node assigns packets of a segment to its direct children during the diffusion phase. While this offers the opportunity to maximize the available throughput, it bears the risk of packets not being distributed at all if packet loss occurs on the first level. To circumvent this problem, the source node performs loss detection on requested pack-

ets. If a child requests a packet that has already been delivered to other children of the source, the source instead delivers a packet that is rare to keep the number of diffusion sub-trees per packet balanced.

With MDC, a peer can request a subset of packets based on its capabilities to stream a video with lower quality. This, in turn, limits the availability of packets on the peer, which might lead to a content bottleneck for peers with higher capabilities. However, as peers with higher bandwidths form more connections due to the normalization with $bwpf$ (see Equation 3.1), they are able to retrieve more packets during the swarming phase to achieve the desired quality. As long as nodes have a symmetric access link ($\forall i : bw_{in,i} = bw_{out,i}$), the position of peers with higher bandwidth has little impact on the overall performance. Often a peer's access link is asymmetric, with the download bandwidth being higher than the upload bandwidth ($bw_{in,i} > bw_{out,i}$). This results in a shortage of outgoing bandwidth, which in turn might lower the received quality for some peers. The system as a whole does not suffer significantly from asymmetric links, as long as the total outgoing bandwidth is high enough to deliver the desired content quality in terms of MDC descriptions.

In case of bandwidth shortages or, more generally, missing MDC descriptions of the content, a peer adapts its target quality. As MDC is used, the target quality is determined by the number of descriptions. Correspondingly, to lower the target quality, a peer decreases the number of requested descriptions. The current rate of delivery from all parents is evaluated once in every interval Δ and smoothened with an exponentially weighted moving average to prevent too frequent changes in the target quality.

PRIME presents a scheme for mesh-based live streaming that is able to utilize the outgoing bandwidth of peers during the swarming phase and at the same time maintains low latency by rapidly spreading new packets during the diffusion phase. Another key design decision is the assignment of a fixed bandwidth per flow to better cope with heterogeneous peers. While PRIME is designed to work with MDC, its goal of minimizing content bottlenecks and bandwidth bottlenecks is valid for SVC encoded data as well. One system that uses SVC and tries to minimize the same bottlenecks but uses a significantly different approach is CHAMELEON.

Chameleon – Utilizing Network Coding

While MDC splits a stream in independent descriptions, with each description equally increasing the quality of the decoded video, SVC requires a particular subset of packets in order to decode the stream with a specific quality, as described in Section 2.1.2. This imposes additional complexity on the scheduling algorithm of the streaming overlay, as all required layers of the video have to be delivered or otherwise the content cannot be decoded in the requested quality. More specifically, assigning layers to senders in a way to achieve the maximum overall throughput was proven to be a NP-hard problem [CN03].

CHAMELEON [NLE10] follows a different approach to ensure high throughput without the complexity imposed by the requirements of SVC. Instead of assigning one layer to exactly one sender, multiple senders deliver network coded content for a given layer. The concept of network coding resembles the approach taken with MDC: for each segment of information, multiple, linearly independent descriptions are created. However, MDC requires only one description to retrieve a useable representation of the original data and enhances this representation with each additional description. Network coding, on the other hand, requires a fixed amount of independent descriptions in order to restore the original data. Additional descriptions do not enhance or alter the already decoded information. In a nutshell, MDC is a data-aware coding scheme, whereas network coding is completely unaware of the meaning of individual segments. Network coding enables a receiver to deal with moderate packet loss without the need for retransmissions.

In CHAMELEON, each layer of the SVC stream is independently packetized and encoded to decouple the inter-frame dependencies of SVC from the inherent equality of network coded packets. This process is shown in Figure 3.3, where information inside each Access Unit (AU), i.e. a segment of the stream that is independently decodeable, is grouped by its corresponding SVC layer. This results in one packet for each layer inside the given segment which can then be network encoded into several units. Thereby, the aforementioned decoupling of network coding from inter-layer dependencies is ensured.

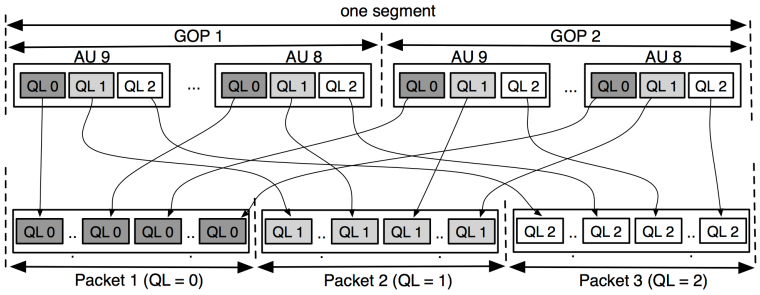


Figure 3.3.: Packetizing in CHAMELEON (adapted from [NLE10])

As SVC is intended for streaming systems with heterogeneous receivers, CHAMELEON addresses the question on how to select neighbors based on the desired quality of the video stream. One straight forward approach would be to assign each peer to a fixed given class based on its capabilities. Peers would then prefer neighbors of the same or a higher class to prevent content bottlenecks. Class-based neighborhood selection ignores the current state of the overlay in that dynamic changes in the available bandwidth are not taken into account. Therefore, CHAMELEON uses a hybrid approach, where peers are grouped into classes but at the same time also evaluate their actual received quality. A peer selects neighbors such that the actual received quality of the neighbor is close to the class of the peer. This combination of class-based and quality-based neighborhood selection leads to a high quality satisfaction together with a low skip rate due to lost packets or insufficient bandwidth capabilities.

Ponder – Mesh-Based Video on Demand

While the aforementioned systems are intended for live streaming, there exists a multitude of systems designed for Video on Demand (VoD) as well. One example of such a system is PONDER [GYL+08]. In a VoD setting, the peer selection algorithm and the scheduling process differ from the systems presented previously. First of all, caching is no longer done in a sliding window approach. Instead, peers cache all segments they received so far in order to be able to provide them to other peers that watch the video at an earlier playback position. While this greatly increases availability of segments in the overall system, it requires additional storage space. To lower the startup delay for new clients and after a skip to another playback position, PONDER utilizes the central streaming server to fetch the most urgent segments after such an operation. Segments are fetched in a priority based scheduling scheme, with segments close to the playback position having the highest priority. If a segment is not delivered in time, it is requested from the central server.

Peers report segment availabilities to a central tracker (which might be the streaming server). A peer requests a set of contacts for a given segment. Basically, the tracker provides contacts for each requested segment, thereby forming separate overlays for each segment. This approach closely resembles the original BitTorrent [Coh03] with its centralized tracker approach. However, in a typical BitTorrent application, all missing segments (or *chunks*) of a file are equally important for each peer, whereas in a video stream-

ing scenario the importance of a segment is determined by the playback position of the peer. In PONDER, the decision on whether a peer uploads a file to another peer or not is based on an urgency metric. This metric is calculated for every requesting peer and is based on the playback position with regard to the requested segment.

The presented scheme with a centralized tracker and support by the streaming server enables operations such as *pause*, *resume* and *jump forward*. Essentially, these operations alter the deadlines for segments, resulting in changed priorities for the scheduling process. Again, if the next segment to be played is not already present in the cache, it is retrieved from the central server. To prevent the server from getting overloaded with direct streaming requests, those requests have to pass through the tracker. If the tracker decides that the server is overloaded and the segments can also be provided via the P2P overlay with reasonable delay, the direct streaming request is rejected.

Despite their complexity when it comes to efficient packet scheduling, mesh-based approaches have shown to offer better performance in dynamic environments with higher node churn [MRG07]. To benefit from advantages of tree-based and mesh-based systems at the same time, many current P2P video streaming systems are based on a hybrid approach. A number of such systems is discussed in the following.

3.1.3 Hybrid Systems

Hybrid systems often rely on the presence of *superpeers*, i.e. nodes that have shown to be stable and can contribute high bandwidth. Superpeers form an overlay that disseminates content from the source to each of the superpeers, similar to a CDN. Regular nodes connect to one of the superpeers and additionally form an overlay network with other nodes connected to the same superpeer. This concept is explained in the following examples of hybrid systems.

ToMo – Mesh over Tree Hybrid Overlay

TOMO [ASK10] tries to combine the low overhead of push-based data delivery with the inherent stability of a mesh-based system. It therefore divides the overlay into a mesh-supported part and a pure tree-based part, as shown in Figure 3.4. The source node divides the packets of the video stream onto several *sub-trees*, with each packet being sent to multiple sub-trees. Each group of sub-trees that receives the complete stream forms an overlay independent from other groups, as indicated by the dashed line. Inside such a *sub-tree group*, nodes on higher levels form a mesh by choosing a child in another sub-tree. This allows nodes in the mesh to gather all packets of the stream.

TOMO does not allow nodes to forward packets they received from another sub-tree to avoid duplicates. Instead, the sub-trees are solely used for the distribution of the payload provided by the source to the given sub-tree. Only at the very last level inside the mesh-based part of the overlay, the complete stream is forwarded to the children inside the sub-tree. This leads to pure, single-tree based data delivery for the lower levels, i.e. nodes not participating in the mesh-based part. In contrast to systems such as CHAMELEON [NLE10] or PRIME [MR09], the authors do not imply any specific coding scheme. However, as they motivate their system for use in scenarios with peers having a connection rate below the streaming rate, some form of SVC or MDC is a key requirement. It remains questionable how the single-tree approach used for lower levels can cope with low-bandwidth peers, besides from placing them solely as leafs. The authors show that the received quality in their evaluation scenario does not change significantly with varying mesh layer depth, whereas the amount of control traffic increases with a deeper mesh. A

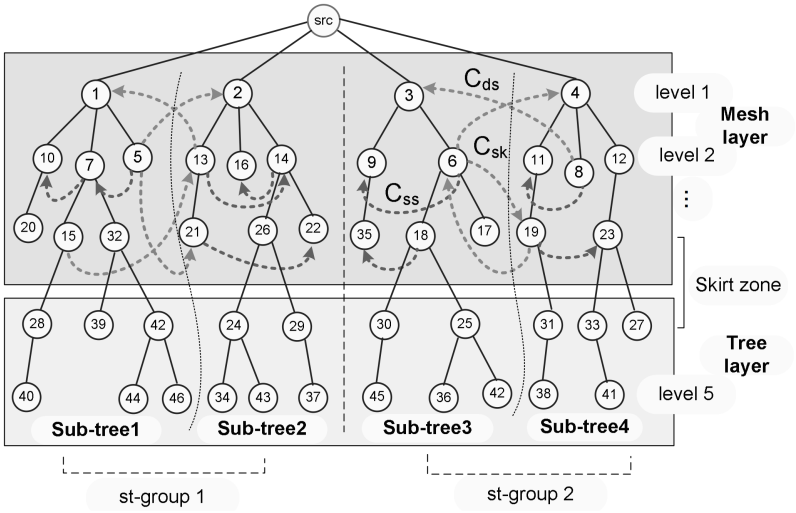


Figure 3.4.: Structure of the ToMo overlay (adapted from [ASK10])

comparison with a multi-tree approach was not part of the evaluation, even if ToMo resembles such a system.

In order to maintain the right balance between mesh-based and tree-based nodes, ToMo maintains a complete list of participants on each node. This information is exchanged between neighbors and eventually reaches the source, which manages the join requests of new nodes by assigning them to a level. This approach is one explanation of the limited scalability in terms of control traffic if the mesh grows. Overlay maintenance is managed solely by the source node, which also implies a significant control overhead in larger networks. While ToMo should not be considered a fully-fledged P2P video streaming system, it provides a starting point for systems that want to deploy push-based data delivery in a mesh-based topology. However, the choice of mesh over tree as the overall structure seems counterintuitive, as a tree-based structure exposes its strengths mostly in stable environments. ToMo, on the other hand, utilizes the mesh-based structure for stable nodes close to the source, whereas it maintains trees for short-living nodes far away from the source.

mTreebone – Tree over Mesh Hybrid Overlay

In mTreebone [WXL07], stable nodes maintain a tree-based overlay to support data delivery with low latency. To increase robustness of the system when compared to a single-tree approach, nodes additionally form a mesh overlay which aids in tree construction and repair as well as in packet delivery. By allowing only stable nodes to participate as intermediate nodes in the tree, overhead to maintain a balanced tree structure lies within reasonable boundaries. All unstable nodes will participate in the tree as leaf nodes. The authors propose mechanisms to identify stable nodes as well as means to optimize the resulting tree which will be outlined in the following.

Identifying stable nodes: Nodes participating in the backbone, i.e. as intermediate nodes in the tree, should preferably stay connected as long as possible to avoid frequent tree restructuring. In the context of video streaming, the older a peer, the higher the probability that it will stay online even longer [AA96]. In mTreebone, a node is added to the backbone as soon as its lifetime exceeds a threshold. This threshold is not statically configured but instead depends on the time a node joined the session as well as the total length of the session. The problem of finding superpeers in a video streaming network has been extensively addressed by Ziu et al. in [LWLZ09], where the authors not only consider the lifetime but also

the bandwidth capabilities of a node. Want et al. showed the importance of stable nodes in [WLX08], where they studied the impact of long-living nodes on the overall system performance. They stress the importance of selecting and utilizing such peers in a streaming system.

Optimizing the tree: To minimize the overall latency, it is desirable to keep the tree balanced and as short as possible. Therefore, mTreebone utilizes two localized algorithms to detect and fix suboptimal configurations. If a node has more children than its current parent, it will move up in the tree and take its former parent as a new child. Second, a node will always try to minimize its distance towards the source node if there is a parent with sufficient bandwidth. Both algorithms will be executed periodically, which will finally result in a tree with minimal average depth.

Push and Pull Packet Scheduling: While the primary method of content delivery is the push-based approach over the backbone tree, nodes have the possibility to request missing packets from their mesh neighbors. To avoid duplicates resulting in a pull request followed by a push delivery over the tree, each peer is only allowed to request packets via the mesh if it already received a newer packet via the tree. In essence, the mesh is used to fill gaps in the buffer that resulted from either packet drop or a restructuring of the tree.

With a stable tree as the backbone of the system and a mesh to aid in packet delivery and tree maintenance, mTreebone can best be described as an evolution of a single-tree approach. The system is designed to achieve high reliability as well as low latency by promoting stable nodes to participants in the backbone. The underlying mesh network allows for signaling and neighborhood exchange while being robust to churn at the same time.

LiveSky – a Hybrid CDN-P2P System for Live Streaming

While mTreebone constructs a stable backbone by distilling more reliable peers, LiveSky [YLZ+09] deploys a CDN to provide reliable delivery towards the edges of the overlay network. The operating principle of a CDN was described in Section 2.3.2. In the following we focus only on the P2P part of LiveSky. Assuming that a full version of the stream is available at every edge node of the CDN, interested users connect to one of the edge nodes in order to download the stream, as shown in Figure 3.5. In LiveSky, peers connected to the same CDN node additionally form a P2P overlay. This lowers the load on the CDN and allows it to handle even more users. For the P2P part, LiveSky uses a hybrid approach, where the stream is split into different substreams which are then distributed in a multi-tree fashion. Lost or missing packets are received by means of an additional pull-based mesh overlay.

Contrarily to mTreebone, this hybrid P2P overlay is limited to users that are connected to the same CDN node. Furthermore, participation in the P2P overlay is not mandatory, as legacy nodes can still be supported by some of the CDN nodes. Those nodes will just have to provide the full stream to interested legacy peers. One interesting effect of the separated P2P overlays for each CDN edge node is the locality awareness that can be achieved by the Domain Name System (DNS)-based redirects. The evaluation shows that most users participating in one specific P2P overlay are connected to the same ISP, which in turn reduces the latency. Achieving locality awareness in a P2P overlay is considered beneficial in order to prevent *zig-zag routes* in the underlying network. LiveSky efficiently circumvents this problem by forming multiple independent, locally bounded overlays.

For most video streaming application scenarios one can assume that there is some additional infrastructure installed by the content provider. Therefore, utilizing this infrastructure in a way that also enables the remaining part of the system to perform better is highly desirable. The concepts introduced in LiveSky, namely the hierarchical hybrid overlay, might also be applicable outside the specific context of a CDN. Overall, hybrid systems combine strengths of mesh- and tree-based solutions and try to eliminate

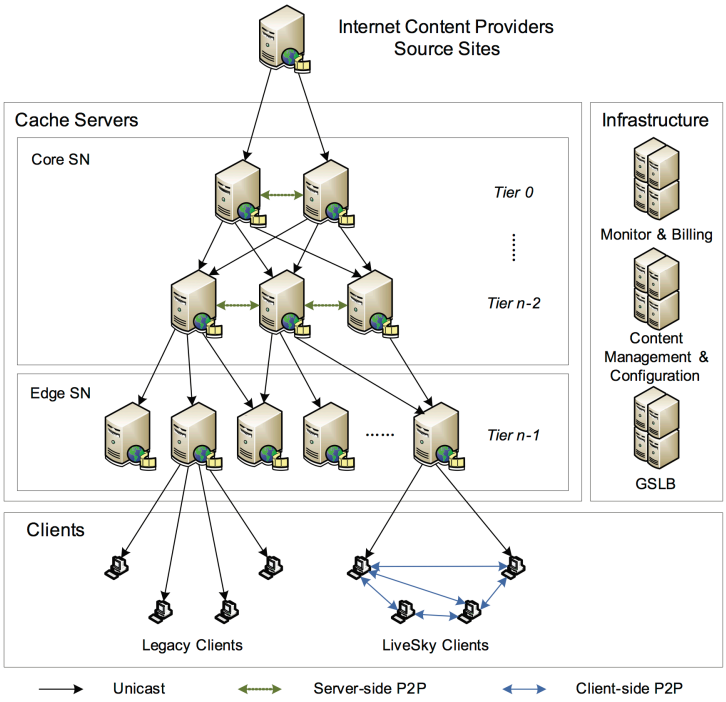


Figure 3.5.: Structure of the LiveSky system (adapted from [YLZ+09])

most of their weaknesses. This might be a promising approach when compared to the simple solutions, especially in more generalized settings with heterogeneous nodes.

All previously shown streaming systems contribute ideas to the design of a transition-enabled video streaming overlay in one way or the other. In the following, their key design decisions and resulting properties are summarized, which provides the foundation for the system design shown in Section 5. Generally, a video streaming system can be categorized based on the topology and the scheduling scheme as well as the requirements regarding the video coding. An overview on the aforementioned systems and their categorization is given in Table 3.1. The systems are rated regarding their properties as either *optimized* (+), *unaware* (∘) or *inefficient* (−), which in most cases is directly related to the optimization goal defined for the particular system. These ratings are derived from the evaluation results presented by the authors and the comparison to other streaming systems. Furthermore, known trade-offs are considered, such as the relation between the height of the tree and the delay between source and leaf nodes. They provide a general overview on the strengths and weaknesses of a system regardless of the explicit scenario the authors had in mind when designing it.

The considered properties are explained using the example of the P2P video streaming system Overcast. The authors of Overcast designed a system that is explicitly optimized for high throughput, which is why the maintained tree is deep with a small number of outgoing links at each node. This directly affects the observable latency between the source and participating nodes, which is high due to the number of hops needed to reach the leaf nodes. As Overcast was designed as an intermediate, proxy-like system for multicasting, it assumes that peers are rather stable. Therefore, the protocol is not intended to be robust against high node churn. Overcast is considered fair, as all nodes contribute upload bandwidth and the protocol tries to maintain a balanced tree. One could argue that leaf nodes in a single-tree do not contribute, but in Overcast the leaf nodes of the system use their upload bandwidth to provide content

System	Topology	Scheduling	throughput	low latency	robustness	fairness	scalability	heterogeneity	Video Codec
Overcast (2000)	single-tree	push	+	−	−	+	−	−	unaware
CoopNet (2002)	multi-tree	push	∘	+	+	+	∘	−	MDC
SplitStream (2003)	multi-tree	push	∘	+	+	+	+	−	MDC
Stanford P2PM (2007)	multi-tree	push/pull	∘	+	+	∘	∘	∘	AVC or SVC
CoolStreaming (2005)	undirected mesh	pull	∘	∘	+	+	+	∘	unaware
Prime (2009)	directed mesh	pull	∘	+	+	+	+	∘	MDC
Chameleon (2010)	directed mesh	pull	+	∘	+	∘	+	+	SVC
ToMo (2010)	mesh over tree	hybrid	+	+	∘	∘	−	+	unaware
mTreebone (2007)	tree over mesh	push/pull	+	+	+	∘	+	+	unaware
LiveSky (2009)	CDN + hybrid	push/pull	+	+	+	∘	∘	+	unaware

Table 3.1.: Overview of P2P live video streaming systems and their support for key characteristics as *optimized* (+), *unaware* (∘) or *inefficient* (−).

to the end-users, which are not part of the overlay itself. In terms of scalability, the system is limited due to the resulting delay and maintenance overhead of the Up/Down protocol. Finally, the system does not consider heterogeneity of devices, neither as participants in the overlay nor as clients to a leaf node.

In contrast to OVERCAST, the more recently proposed system CHAMELEON utilizes SVC as video codec, thereby supporting heterogeneous peers by design. While robustness is a property that is striven for by most designs, especially mesh-based systems are able to achieve both, robustness and scalability. This is due to the fact that such systems do not rely on any centralized coordination, as for example OVERCAST does with its Up/Down protocol. Furthermore, they do not try to maintain a fixed dissemination path for new packets. By design, mesh based systems expect connections between peers to be short-lived. Therefore, the impact of churn on such systems is far less significant than on a tree-based system.

From the overview given in this section it becomes clear that each solution has its niche where the overall performance is very well but shows significant drawbacks as soon as those conditions are no longer met. This motivates the design of a video streaming overlay that is able to adapt itself to a multitude of these conditions by transitioning between different states. The chosen state should always exhibit the best possible performance for the current conditions. While existing systems may provide adaptivity to some extent, they are still tailored towards one specific setting as shown in Table 3.1.

3.2 Scheduling Concepts in P2P Video Streaming

Packet scheduling algorithms can be categorized as either *pull*, *push* or *hybrid*. As seen in the previous examples of P2P video streaming systems, the scheduling mechanism is closely related to the chosen overlay topology. In a tree-based topology, push is the natural choice, whereas in a mesh a pull-based approach is required. However, as hybrid approaches have proven to be superior under real-world environmental settings, the need for an equally flexible scheduling scheme arises. This leads to the hybrid *push-pull* scheme, where a subset of packets is pushed down a tree whereas the remaining packets are pulled over mesh connections. This scheme is used for example in MTREEBONE as presented in the previous section. However, the efficiency of the scheduling algorithm does not only depend on the overlay

topology but also on the characteristics of the content that is to be streamed. SVC encoded material for example shows dependencies between different packets that have to be resolved by the scheduler.

Liang et al. [LGL08a] evaluate different scheduling mechanisms under varying system conditions. They show, that the design of a scheduling mechanism becomes increasingly important in a nearly saturated system. For systems with low load and low timing constraints, even a random scheduling mechanism provides good performance. These findings motivate the careful design of a scheduler for a SVC enabled live video streaming system, where both the streaming bit rate and the timing constraints are significant. Furthermore, the authors show that a video streaming system needs a bootstrapping mechanism that provides the initial chunks of video data with high bandwidth and low delay to ensure low startup times. This mechanism has to be included in the scheduler and motivates the coordinated search for peers with higher capabilities that can be used as superpeers to aid other peers during their startup phase.

The push-based approach offers high performance in terms of delay, as packets can be forwarded the moment they arrive at a node. Using pull-based scheduling, each packet has to be requested explicitly, adding additional time that is needed to advertise the packet and to send the request. Therefore, hybrid overlays like mTreebone [WXL07] try to utilize push-based scheduling at least to some extend. In the traditional pull approach it is assumed that the source for each packet is determined based on the exchange of buffer maps. However, if those selected parents show decent stability and are interested in the same content as the child node, they will with high probability be able to provide future packets as well. By exploiting this probability, nodes can form longer-lasting connections between each other that result in the implicit creation of distribution trees for a subset of packets. This property is for example used by the mesh-based streaming overlay Prime [MR09] and its diffusion subtrees, as presented above. This becomes increasingly interesting in the context of SVC, where those distribution trees could easily be related to a specific layer of the stream.

Zhang et al. [ZZSY07] show, that a pull-based mechanism leads to very efficient utilization of the peer resources. In turn, this enables the system to reach nearly optimal throughput behavior. Therefore, a hybrid solution that uses pull-based coordination but push-based packet delivery based on longer lasting schedules exhibits the desirable properties of low delay and high throughput. Furthermore, such a scheme can easily adapt to varying network conditions. If a network becomes unstable, the schedules will have shorter lifetimes when compared to a stable network. This finding motivates the decoupling of the scheduling mechanism and the neighborhood management in the system that is proposed in this thesis, as explained in detail in Section 5.

3.3 Topology Transitions in Peer-to-Peer Systems

Most overlays presented in the previous section perform self-optimization of their topology to ensure high performance even in the presence of churn. They do not, however, completely alter their topology or their scheduling scheme if the environment changes. As shown in the previous section, the efficiency of a specific system significantly depends on the environmental parameters. If those parameters are subject to change, for example due to an increasing demand or network failures, it would be desirable to transit to a system that performs better in the new environment. While there is currently no such system in the context of P2P video streaming, there are some approaches that provide starting points for topology transitions in a more generalized P2P system.

T-Man – Gossip-based Topology Construction

T-MAN [JMB09] is a Gossip-based protocol that allows to create arbitrary overlays. An overlay is defined by the interconnections of nodes, i.e. the routing table entries at each node. This set of interconnections is called a *view* in T-MAN. To create an overlay with the desired topology, it is sufficient to specify how to form these views out of a given set of node contacts. Therefore, T-MAN introduces the *rank* function. This function sorts a set of nodes according to some metric and thereby allows each node to generate a more refined view with each set of new contact information received by its neighbors.

A node n in T-MAN periodically selects one of its neighbors, n_r, at random. It then sorts its own view together with its own contact information by applying the rank function. The first m entries are sent to node n_r which will in turn answer with the m foremost entries in its view. These new contacts will then be added to the view of node n. Due to this exchange and processing of neighborhood information, neighborhoods will converge to a global state that is specified by the rank function. The authors show the creation of ring-based and tree-based topologies and the corresponding rank functions and provide extensive evaluation showing fast convergence even in large networks. The protocol itself is resistant to message loss, as it creates around 99% of the connections needed in the ideal case even in the presence of up to 20% message loss. To prove the capabilities of their solution, the authors construct a CHORD overlay with finger links. Compared with a traditional CHORD overlay T-MAN achieves only slightly worse forwarding behavior due to minor inaccuracies in the routing table.

The focus of T-MAN lies on the construction of various overlays, not in transitions between them. However, it would be conceivable to extend the idea of a rank function to support transitions. One approach would be to distribute a new rank function together with the neighborhood information, leading to a convergence towards the new topology. This shift in topologies would have to be triggered by a central entity, which in the case of P2P streaming could be the source node or the tracker. However, it remains questionable if reliable packet delivery is still possible during the transition phase, especially with the added timing requirements of a video streaming application.

Besides the overlay topology, additional possibilities for transitions exist in the context of video streaming. One example would be a transition from pull- to push-based packet delivery, which could additionally be further differentiated in the context of SVC and its support for heterogeneous peers. Possible transitions are identified in the following section, together with the requirements they add to the design of a general P2P live streaming system. Based on those requirements, the design of a transition-enabled P2P video streaming system is presented in Chapter 5.

4 Requirements for P2P Streaming and Transition Support

The systems presented in the related work differ in their requirements and derived assumptions. Achieving fluid playback is one of the design goals all those systems have in common. However, there are different strategies on how this requirement is met. Some systems assume MDC as video codec, which enables them to deal with a certain fraction of missing packets without the need for retransmissions. Here, fluid playback comes at the cost of quality degradation if not all descriptions of a segment are retrieved in time. If the assumption of MDC encoded material does not hold, the system may not be functional at all. Other systems try to achieve fluid playback by actively retransmitting missing packets or by utilizing network coding. While such systems do not assume or require a specific video codec, they might introduce significant signaling overhead and packet delay. Obviously, assumptions such as the choice of the video codec have direct impact on the design of a video streaming system and its applicability in different scenarios.

The requirements and assumptions that influence the design of the streaming system presented in this thesis are detailed in the following, starting with general requirements for a P2P live streaming system.

4.1 General Requirements for a Live Streaming System

Requirements for live streaming systems can be divided into two categories. Application or user-defined requirements are top-level requirements that are specific to the application scenario. System requirements are more general and might be valid for a broad range of applications and overlay networks. Liu et al. [LRL08] present unique characteristics of a P2P video streaming system. These characteristics include real-time constraints arising from the need for uninterrupted playback as well as high bandwidth requirements due to the content that is to be disseminated. The authors furthermore stress that a streaming system should adapt to bandwidth heterogeneities. The application requirements specified in the following section are directly derived from Liu et al., as well as from the requirements used for streaming systems presented in the related work.

4.1.1 Application- and User-Defined Requirements

As already mentioned, many application- or user-level requirements can be derived from the related work presented in Chapter 3. Requirements such as a *low startup delay* and *continuous playback* are targeted by all streaming systems, whereas for example the support for heterogeneous devices is only addressed by some of them. The system presented in this thesis has to address the following requirements:

Low Startup Delay A user wants to experience low startup delay when joining a stream. The startup delay is affected by the size of the buffer that needs to be filled before playback starts and the effectiveness of the joining process of the overlay.

Continuous Playback Once the stream starts playing, it should not be interrupted due to missing packets or an empty buffer. The number and length of such interruptions or *stalls* affects the overall perceived quality of the service.

Quality of the Video Stream The user wants to retrieve the best possible quality when watching a layered video stream. The quality of the stream is confined by the network speed and the resources of the playback device, such as screen size or processing power.

Support for Heterogeneous Devices The application should use the resources of devices in a reasonable way. This includes the utilization of the access link bandwidth, especially if the access link bandwidth is asymmetric or insufficient to stream the highest quality of the video.

Obviously, some of these requirements affect each other: if peers experience variant startup delays and interruptions in their playback, they are no longer aligned in their playback position. On the other hand, if the startup delay remains constant across all peers and if peers do not experience any interruptions at all, they are automatically aligned. As the video stream in this thesis is assumed to be encoded using SVC, its quality is directly correlated to the layer that is being played. The SVC encoded video stream also provides a good starting point to tackle the problems that arise with heterogeneous devices and asymmetric access link bandwidths. Some of these application-level requirements directly translate into system-level requirements for the design of a P2P streaming overlay.

4.1.2 Streaming Overlay Requirements

As mentioned above, support for heterogeneous devices is eased by the utilization of SVC encoded material. Therefore, one key requirement for the proposed streaming overlay is the awareness of the video codec.

Video Codec Awareness As content is encoded using SVC, the streaming overlay should be aware of this fact and prioritize segments of the video according to the layer and the importance of that layer. One simple mechanism would be to always prioritize lower layers and especially the base layer, as those packets are needed on all peers to decode higher layers.

Low Control Overhead The amount of control traffic should be negligible when compared to the traffic that arises from the transmission of actual video payload. Therefore, the overlay has to be designed in a way that minimizes the overhead while still achieving the desired functionality.

High System Throughput To provide support for high quality video streams the system has to provide high end-to-end throughput, even in the face of heterogeneous access link bandwidths and peer churn.

Low Source-to-Client Delay This is directly related to the application-level requirements of low startup delay and playback alignment. New packets need to be disseminated throughout the overlay as fast as possible to ensure fluid playback with low delay.

Reliable Delivery Since SVC encoded material can only be played if at least the base layer is fully available, the overlay has to provide reliable delivery of these packets. Missing packets need to be retrieved in time, as otherwise the playback would stall.

Resilience against Churn Churn describes the frequency of peer arrivals and departures in a system. Especially in a live video streaming system, arrival rates might depend heavily on the time of day. Even more critical, events in the real world can trigger tremendous and spontaneous growth of user numbers – a phenomenon known as *flash crowd*.

Scalability As mentioned above, the amount of users of a live video streaming system can vary significantly over time. Therefore, the system should be inherently scalable from a few hundreds to thousands of peers streaming the video simultaneously.

Other requirements, such as fairness, are not especially considered in this thesis. Hu et al. [HGL11] provide extensive insights into how fairness and incentives can be provided by using SVC encoded material. Integrating explicit means for fairness and incentives as presented by Hu et al. is considered future work. Besides the aforementioned requirements for a P2P live streaming overlay, additional requirements arise from the support of transitions in such a system.

4.2 Concept of Transitions and Derived Requirements

Most of the requirements stated above hold true for any live streaming system. However, in this thesis, the possibility of transitions in such systems and their impact on the system performance is studied. In the following, we define the term *transition* in the context of a live streaming system. Based on the definition, we identify key possibilities for transitions and state the requirements that arise in order to support those transitions in our design.

4.2.1 Definition of a Transition

For most problems, there exist several mechanisms to solve them, each having its own set of advantages and disadvantages. In the context of a P2P streaming system, the problem would be the efficient content delivery from the source towards all interested parties. All streaming systems presented in the related work are essentially just mechanisms that solve this problem. However, they differ in their assumptions on the environment and optimization goals, as already pointed out in Chapter 3 in the discussion of Table 3.1. Therefore, if one wants to know which mechanism is best suited to implement a streaming system, one has to define the environment and the desired characteristics. In this thesis, we propose a different approach following initial ideas presented by Rückert et al. [RH12]. Instead of selecting one mechanism, we would like the system to always utilize the mechanism that best fits the current environment.

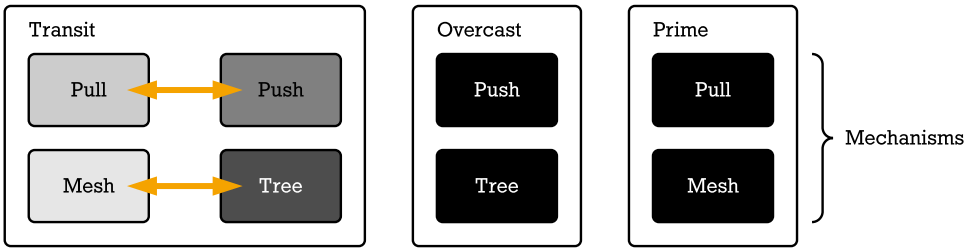


Figure 4.1.: Scheduling and topology concepts within a streaming overlay. TRANSIT enables fine-grained combinations changing over time and space, while other overlays stick to one mechanism or a fixed combination of mechanisms.

Considering a streaming system, one can differentiate between different scheduling strategies and overlay topologies, as shown in the related work. The choice of scheduling strategy and topology determines the performance of the overlay under various conditions. For the systems presented in the related

work, those mechanisms are fixed, as illustrated in Figure 4.1. The proposed, transition-enabled system TRANSIT, however, uses fine-grained combinations of push and pull-based scheduling as well as topologies somewhere in between mesh and tree. In TRANSIT, the utilization of mechanisms implied by the shade of the boxes can change over time, as mechanisms are determined by the environmental conditions.

As the environment may change over time or even differ depending on a peer's location, we have to switch mechanisms during operation and maybe only on a subset of all peers. However, the user should not notice such a switch of mechanisms at all. We therefore define a transition to be the *seamless* switching from one mechanism to another. Regarding the use case of a video streaming system, such a transition could for example be the switch from a tree-based system such as OVERCAST to a mesh-based system like PRIME.

4.2.2 Requirements for Transitions in P2P Live Streaming

One interesting transition possibility in P2P video streaming arises from the different topologies and scheduling schemes that are presented in the related work. As shown, each of those systems has its strengths under specific environmental conditions and application scenarios. In a transition-enabled system, one could seamlessly switch between any of those topologies and scheduling schemes to always utilize the scheme that provides best performance under the current conditions.

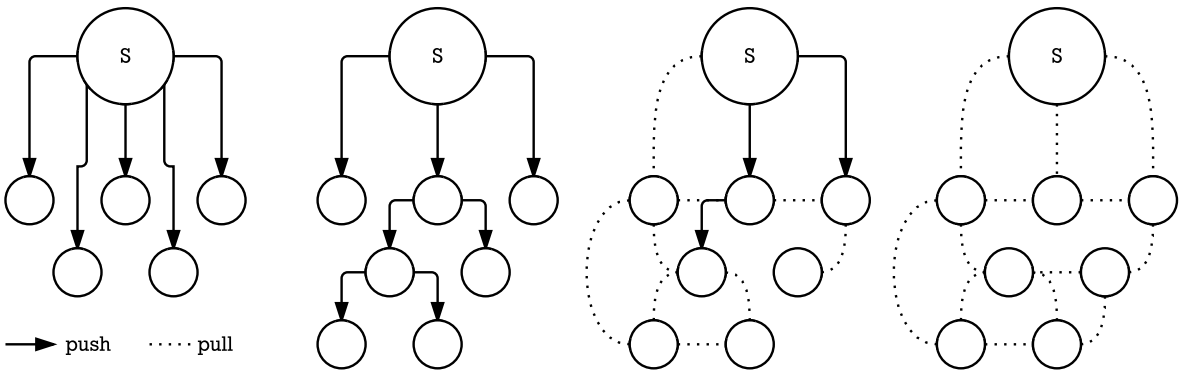


Figure 4.2.: Topology transitions from pure client/server to tree, hybrid and mesh (from left to right)

Figure 4.2 shows four possible transition states of a video streaming system. Starting with a pure client/server system on the left, nodes begin to form trees for content distribution and thereby behave as a tree-based system with push-based scheduling. While the depicted system uses a single tree for packet distribution, multi-tree systems would be possible as well. The third stage shows a hybrid tree and mesh overlay, where a subset of peers still participate in a tree originating at the streaming source. The remaining peers connect to the peers in the tree or the streaming source and pull missing packets. Finally, in the last stage all peers request packets in a pure mesh fashion using pull-based scheduling. Consider the streaming systems presented in Chapter 3: each system could represent one stage in the topology transitions shown in Figure 4.2.

The system presented in this thesis is designed to support topology transitions as described above. A topology transition might imply a scheduling transition as well. A transition from a tree towards a mesh-based system, for example, would require the scheduler to switch from push-based scheduling to

a pull-based approach. The different scheduling schemes required for each of the transition stages are indicated by the line styles of connections in Figure 4.2.

For this thesis, the following requirements regarding the support for transitions are defined:

Different Topologies The system has to support transitions between pure mesh-based, pure tree/multi-tree and hybrid overlay topologies. These transitions might be executed only on a subset of all peers, if appropriate.

Different Scheduling Schemes As mentioned above, the topology transitions imply transitions of the scheduling mechanism. Therefore, the system has to support different scheduling schemes such as pull, push and any hybrid combination.

Seamless Transitions All of the aforementioned transitions are to be executed seamlessly, i.e. the requirements for live streaming defined in Section 4.1 have to hold before, during and after a transition.

Decentralized Transitions Transitions are executed by peers without any central management and control. Therefore, peers have to evaluate their environment and initialize a transition based on their local knowledge.

System Convergence The system has to converge to a stable state if the environmental conditions do not change significantly. Particularly, oscillations, i.e. too frequent transitions between two mechanisms because of slight variations in the environmental properties have to be avoided.

This thesis focuses on the transition of overlay topologies and scheduling schemes, as stated in the requirements above. We consider churn and the total amount of online peers to be the most important triggers of a transition, hereafter mostly referred to as *environmental conditions*. As the requirements are defined by now, the design of a P2P live streaming system with support for transitions is presented in the following chapter.

5 Design of a Transition-Enabled P2P Streaming System

Based on the requirements defined in the previous chapter, the design of the live video streaming system TRANSIT is presented in this section. The system has to serve two purposes: on the one hand, it has to be a fully fledged P2P live streaming system that supports seamless transitions between different scheduling mechanisms and overlay topologies. On the other hand, it has to act as a testbed for those mechanisms and topologies, such that they can be compared against each other in the same environment and under the same system assumptions. The fundamental design choices made in TRANSIT are detailed in the following section. Those design choices enable the support for transitions and heterogeneous devices, as presented afterwards. Finally, the design of TRANSIT as a testbed for transitions and the impact of different mechanisms is presented.

5.1 Fundamental Concepts and System Architecture

In a transition-enabled system a multitude of mechanisms need to coexist and communicate with each other. Due to the asynchronous nature of a distributed system, different peers in the overlay can be at different stages of a transition. Transitions are not triggered by a central entity but are based on local knowledge of each individual peer, as stated in the requirements. Thus, the state of a peer and thereby its currently used mechanisms might differ from the states of its neighbors. To allow those different mechanisms to communicate with each other, a unified protocol has to be defined. This unification helps in the realization of transitions later on.

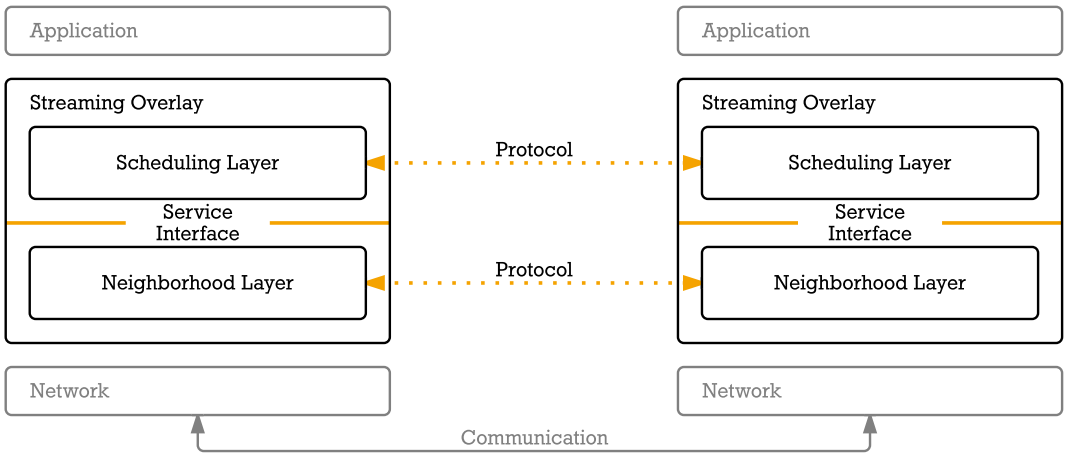


Figure 5.1.: Scheduling and neighborhood management as layers of a P2P video streaming system, together with the service interfaces and protocols.

A P2P video streaming system can be split into two functional layers as shown in Figure 5.1. On the lower layer, peers need to find neighbors and form connections to those neighbors. We refer to this

layer of the system as the *neighborhood layer* hereafter. The information gathered by the neighborhood layer is then used to schedule the transmission of payload. Accordingly, this functional layer is termed the *scheduling layer*. On each of those layers there exist a multitude of protocols and strategies. Some systems closely intertwine both layers, but in order to allow different mechanisms to run in parallel while avoiding unnecessary overhead, we decided to define a clear interface, as shown in the picture. In our proposed design, the neighborhood layer provides *connections* to the scheduling layer. How those connections are formed is up to the mechanism used in the neighborhood layer. In order to allow peers to communicate with each other regardless of the mechanisms they currently use, the protocol between two neighborhood layers was unified. How this is achieved in TRANSIT is presented in the following.

5.1.1 Neighborhood Layer and Connections

Peers in a decentralized P2P overlay need to exchange information about other peers they know in order to build and maintain a well-connected topology. The initial set of contacts is sent by the tracker together with meta information on the video, as soon as a peer joins the overlay.[1] The newly joined peer negotiates connections with some or all of the initial contacts. During the connection negotiation, peers exchange their current neighborhood, i.e. the set of contacts they have successfully established connections to. Thereby, the newly joined peer gets to know more and more contacts. The peer initiates new connections until a threshold of successfully opened connections is reached. From then on, new connections will only be formed to replace existing ones that have not been used for the transmission of payload for some time. Therefore, in TRANSIT we define *neighbors* to be the peers we negotiated a connection with. Apart from these neighbors, a peer might store additional contacts as fallbacks, for example if he has to replace a broken or unused connection. Of course, a peer might also request additional contacts from the tracker, if the initial neighborhood is not sufficient.

Peers only communicate with peers they have established a connection to. If a connection is currently not used for the transmission of video payload, it is still periodically probed with heartbeat messages. As mentioned, peers periodically piggyback a subset of their current neighborhood when sending this control messages or video payload. The neighborhood management mechanism defines which neighbors are included. A simple mechanism might just send a random subset, whereas more convenient mechanisms might select peers based on additional criteria such as their performance in the past. There exists a number of *epidemic protocols* that provide peers with new neighbors and maintain some desirable properties of the resulting topology at the same time. Prominent examples are CYCLON [VGvS05] and SCAMP [GKM01]. These protocols have proven to be very scalable and robust against churn due to their fully decentralized nature. For CYCLON, the authors show that the average path length converges to the average path length of an equally sized random graph. Furthermore, they show that the network has a low *clustering coefficient*, lowering the probability of overlay partitions.

For TRANSIT, some additional information about a peer's performance are included in the decision on which contacts are sent to a neighbor. Most important, only contacts that we have established a connection to are considered in the process. Fallback contacts are not included, as those peers might not be online anymore. Such stale contacts should not be spread throughout the overlay, as contacting them results in timeouts that slow down the overall connection process. The second information used in TRANSIT is the advertised load of a peer. Peers that established a connection between each other piggyback their load in terms of utilized upload bandwidth. As all connections are initiated by the

[1] We assume the tracker to be well-known to all participants.

receiver, this information helps in determining the probability of a successful connection setup. A peer that already advertised high load might not be able to provide additional bandwidth for a new connection.

Traverso et al. [TAB+12] conduct an experimental comparison of different neighborhood selection strategies in an unstructured P2P streaming system. They show, that the neighborhood selection and, thus, the overlay topology of the communication mesh, has high impact on the performance of the system. The authors confirm that simple mechanisms, such as selecting neighbors based on the Round Trip Time (RTT), already improves the performance significantly. According to the authors, the best performance is achieved when connections are added based on their RTT and later on dropped based on their utilization for the transmission of blocks. This strategy is also used as default mechanism in TRANSIT, albeit other mechanisms can be realized as well, based on the information that is provided with every new contact. This includes, as already mentioned, the load and the current playback position. Additionally, once a connection is negotiated, the RTT is taken into account as well. Peers are not grouped by their requested SVC layers, but instead by their quality during the scheduling process, as those connections performing good during scheduling are kept open. This design decision is further motivated by the design of CHAMELEON [NLE10], where the authors evaluated the impact of class-based versus quality-based neighborhood selection. They show, that quality-based neighborhood selection performs better than class-based selection, as it is able to adapt to the current situation of the peers in the overlay. Regardless of the mechanism that is used on a particular peer, the neighborhood layer allows us to decouple the neighborhood selection from the scheduling process. All the neighborhood mechanism has to provide is a set of contacts that we can form connections to, preferably in a way that benefits the overall system performance.

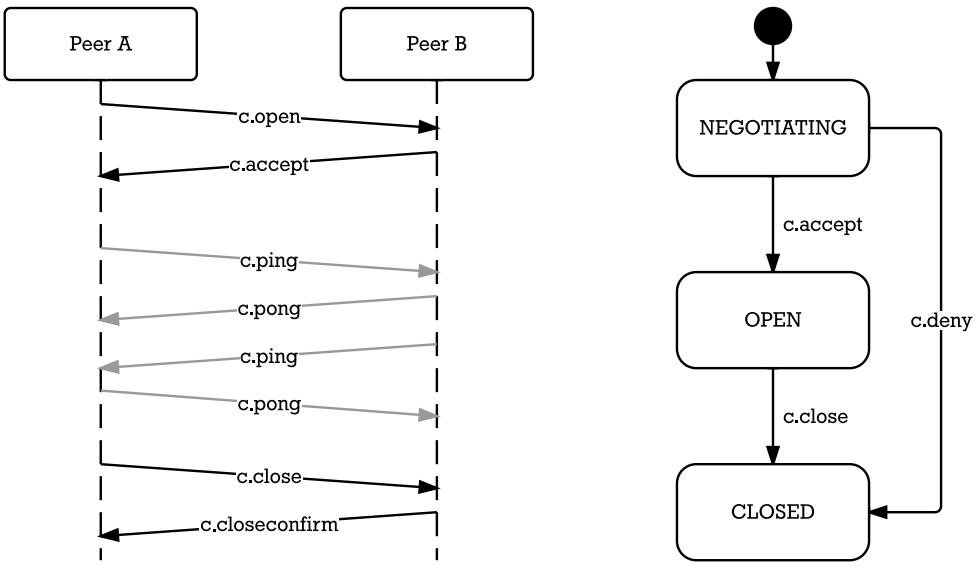


Figure 5.2.: Connection setup, maintenance and teardown protocol and resulting lifecycle of a connection.

A connection is formed using a two-way handshake, as depicted in Figure 5.2. The initiator A sends a c.open message to the intended endpoint B. If B has spare capacity for another outgoing connection, it replies with c.accept, which will signal the successful establishment of the new connection. In case of B having no spare resources, it will answer with c.deny. If the connection has been established successfully, both endpoints begin to periodically issue c.ping messages which have to be answered with c.pong

messages. These messages help in detecting dead nodes and measuring the RTTs, if the connection is currently not used for transmitting video payload. They are also used to piggyback additional information by the scheduler, as described later on.

Message	Description
c.open	Initiates the negotiation of a new incoming connection
c.accept	Accepts a connection request and opens the outgoing connection
c.deny	Denies a connection request
c.close	Closes a connection; resent, if the connection is still used afterwards
c.closeconfirm	Confirms the connection teardown initiated with a c.close, optional
c.ping	Periodical heartbeat to detect dead contacts and measure RTTs
c.pong	Reply to a c.ping

Table 5.1.: Overview of protocol messages used on the neighborhood layer.

Connections are unidirectional for the transmission of video payload in order to prevent loops, with the initiator of the connection being the sink for the video payload. If no video payload has been transmitted over a connection for a given time, or if the RTT exceeds a given threshold, the connection is closed by either one of the endpoints. To close a connection, one endpoint sends a c.close message that can optionally be confirmed by a c.closeconfirm message. If messages arrive via a connection after the connection has been closed on either of the endpoints, a new c.close message is issued. Table 5.1 summarizes the aforementioned protocol messages used by the neighborhood layer in TRANSIT. The lifecycle of a connection is depicted on the right hand side of Figure 5.2. A connection is only used by the scheduling layer if it is in the OPEN state.

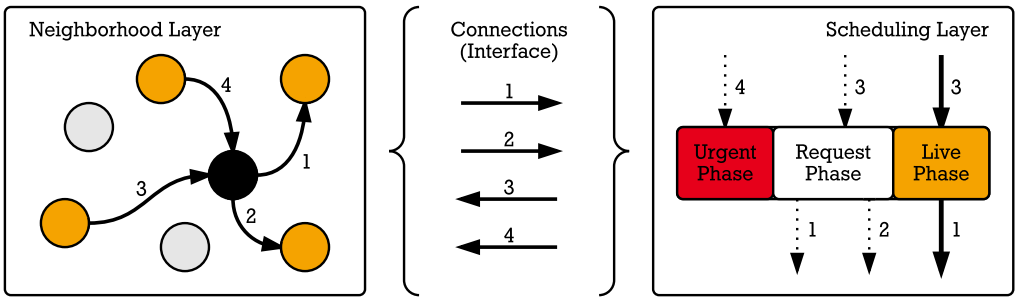


Figure 5.3.: Connections as a service interface between neighborhood layer and scheduling layer.

Connections in TRANSIT are provided by the neighborhood layer as a service for the scheduling layer. Figure 5.3 illustrates this concept. On the left hand side the neighborhood layer manages a set of contacts, with grey ones being spare contacts that were announced by other neighbors. The black peer maintains two incoming connections, which are used by the scheduler on the right hand side to fill the buffer. The scheduler furthermore distributes its buffer content via the outgoing connections. The different phases depicted for the scheduling are introduced in the next section.

Different scheduling mechanisms might be used on different peers, which is why the protocol between those scheduling mechanisms has to be unified as well. In TRANSIT, the concept of *flows* and *requests* is introduced to achieve this unification.

5.1.2 Scheduling Layer with Flows and Requests

As already shown in Chapter 2, one can distinguish between two main primitives in scheduling. Video blocks are either requested (pulled) or delivered (pushed). The latter assumes a contract between the sender and the receiver. In a single-tree-based system, for example, the contract is negotiated during the join phase: as soon as a peer found a parent for the stream, the parent agrees to deliver all blocks of the said stream. If the contract is not fulfilled anymore, maybe due to a node failure, it has to be renegotiated, possibly with a new counterpart. In a tree-based system, this is done by finding a new parent peer for the given stream. If the system is very dynamic, i.e. peers are joining and leaving at high frequencies, such contracts might be very short-lived and turn out to be inefficient.

Let us consider an extreme case, where the contract has to be renegotiated for each block of the transmission. Suddenly, what started out as a push-based system behaves just like a pull-based one, in that each block is requested separately. However, there is still one main difference: in the contract-based system, blocks are not addressed by their individual ID. Instead, they are addressed by their content, which is the layer they belong to when considering SVC encoded material. While a contract therefore would say *all blocks of layer X*, a request would only address specific blocks, for example *blocks A, B, C*. In this thesis, the contract-based scheme is termed a *flow*. This is illustrated in Figure 5.4, where a short segment of the stream and its partition into chunks and blocks is shown. Each block has its own unique ID and belongs to one specific layer of the SVC stream, here indicated by the color. As explained above, the request consists of IDs of individual blocks, whereas the flow specifies which layers are to be sent. By using this concept of *flows* and *requests*, we are able to model arbitrary scheduling schemes and, thereby, adapt to different conditions.

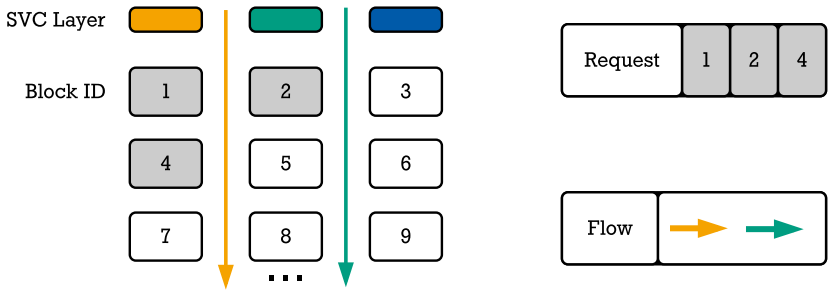


Figure 5.4.: Addressing scheme for flows and requests, with requests addressing individual blocks and flows addressing the most recent blocks of selected layers. Numbers represent the IDs of individual blocks.

An important lesson learned from the related work is the necessity to distribute new packets as fast as possible throughout the overlay. PRIME [MR09], for example, introduces a dedicated phase in the overall scheduling cycle that is used exclusively for the dissemination of recently created content. Hybrid systems such as MTREEBONE [WXL07] deploy pull-based scheduling on more stable peers, thereby distributing new packets efficiently and making them available for requests by peers that do not participate in the tree. Without dedicated mechanisms to tackle the unbalanced interest in packets that is inherent to a live video streaming system, request-based schemes cannot exploit their full potential. However, a pure push-based scheme (resulting in a tree/multi-tree system) that does not tolerate packet loss needs an additional request mechanism to cope with peer churn.

For live streaming in general and for the design of TRANSIT in particular, we define the fast dissemination of newly created packets the first and foremost goal. Thereby, it does not matter if some packets get lost on the way due to peer churn, if we also provide a request mechanism that is able to retrieve missing packets in time. This leads to the concept of flows and requests as defined above, with flows forwarding the most recent blocks of a stream, regardless of currently missing blocks in a peer's buffer or its playback position. As we use SVC encoded material, we define flows by specifying which layers of the stream are to be included in the flow. Depending on a peer's capacity, a flow might therefore deliver the whole video stream or just a subset of all layers. If a flow fails, it is replaced locally by negotiating a new flow with the remaining neighbors. This new flow will then start with the most recent packet. Missing packets due to the length of the switching phase have to be requested and are not part of the flow contract.[2]

Shen et al. [SLZV11] define the exchange of buffer maps to be one of the most important components of a pull system. In TRANSIT, buffer maps are periodically piggybacked on control or payload messages. A buffer map is a ordered set of bits, with each bit being either one or zero, depending on the availability of the respective block. In order to identify individual blocks, the ID of the first block in the set is included as well. With the knowledge about the buffer states of its neighbors, a peer has to assign requests to neighbors. How this assignment is done is up to the implementation of the scheduler; it might be based on the past performance of a neighbor or it might be completely random. It is also up to the scheduler to determine which packets have to be requested next based on the current buffer state. Especially when using SVC encoded material, blocks might be of different importance to the peer. If, e.g., a base layer packet is lost, the stream cannot be decoded at all. Therefore, the scheduler might favor base layer packets over other packets. Another strategy would be to request less widespread packets first, in order to increase their availability to all neighboring peers. This strategy is known as *rarest-first*, and is used for example in the popular BITTORRENT file sharing protocol [Coh03]. Regardless of the strategy at hand, the scheduler has to ensure that blocks are received prior to their respective playback deadline to avoid stalling.

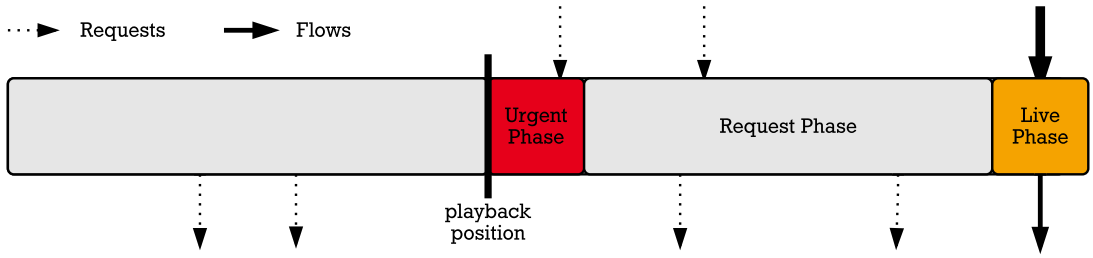


Figure 5.5.: Buffer of a peer, divided into three parts: the urgent phase contains blocks that are to be played next, the live phase contains the most recent blocks of a flow that are directly forwarded. Blocks in between those phases are requested if they are available on neighboring peers.

Therefore, we divide a peer's buffer into three segments, which are shown in Figure 5.5. The bold black line marks the playback position of the peer, and blocks that are to be played next belong to the *urgent phase*, marked in red. These blocks have to be requested immediately if they are available on a neighboring peer, in order to maintain fluid playback. The most recent blocks that are delivered by flows belong to the so-called *live phase*. These blocks are immediately forwarded to other peers, if they negotiated a flow with this peer. Incoming and outgoing flows do not need to be symmetric. In

[2] Remember, flows do not address specific packets.

our example the peer's outgoing flow consumes less bandwidth than the incoming flow, denoted by the smaller arrow. Flows do only deliver the most recent blocks, which is why only blocks in the live phase are sent via outgoing flows. The size of the buffer in between the urgent phase and the live phase depends on the playback position of the peer. Blocks that are missing in this segment of the buffer are requested with lower priority than those in the urgent phase. This part of the buffer can be compared to the swarming phase introduced in PRIME as presented in Chapter 3. Peers use this phase to request blocks that they did not receive as part of a flow. Finally, after a chunk has been played, its blocks are still stored in the buffer for a while. They might be requested by peers that experienced a larger delay in their playback.

Taking a closer look at the scheduling part of Figure 5.3, one observes that a connection can be used for a flow and a request at the same time. Consider for example the incoming connection number three: it is used for an incoming flow as well as for a request. The same holds true for outgoing connections, with connection number one being used for both an outgoing flow and to answer a request. However, only one request per connection is allowed at once, and a new request requires the previous one to be finished or canceled. This ensures that a potential source of video blocks is not overloaded due to multiple parallel requests from one single peer. A request might of course include multiple blocks rather than just a single missing block.

Message	Contents	Description
f.request	Flow schedule	Request for a new incoming flow with the given schedule
f.reply	Flow schedule	Positive answer to a f.request
f.deny	None	Deny of a f.request or source-side cancelation of an active flow
f.cancel	None	Receiver-side cancelation of an active flow
f.alter	Flow schedule	Source-side alteration of an active flow, optional
f.offer	Flow schedule	Source-side announcement of a possible flow, optional
r.offer	Buffer map	Periodically piggybacked buffer map
r.request	Request schedule	Request for one or more blocks addressed by the schedule
r.deny_capacity	Request schedule	Denial of the given request schedule because of current upload workload

Table 5.2.: Overview of protocol messages used for requests (r._) and the negotiation of flows (f._).

Table 5.2 summarizes all protocol messages that are used on the scheduling layer. A new flow is negotiated by sending an f.request message via an incoming connection that is currently not used for a flow. The request contains a schedule such as the one depicted in Figure 5.4, where blocks are addressed by the SVC layer they belong to. The receiver decides based on its current load and incoming flows, whether it can provide the full flow or only a substream. This information is then sent back to the initiator of the negotiation using an f.reply message. If the source cannot provide any flow at all, a c.deny message is sent instead. Meanwhile, the sender begins to forward blocks that arrive via its incoming flows, as long as the flow is not canceled by the receiver and the connection is still open. Blocks that are received via flows are not acknowledged, as the concept of flows does anyway not include a retransmission mechanism. In the occasion that the receiver no longer wants to receive blocks via the incoming flow, an f.cancel message is issued. If the source can no longer provide the full flow schedule, it might optionally issue an f.alter message telling the receiver which parts of the flow can no longer be delivered. If the full flow can no longer be provided, an f.deny message is sent instead of the f.alter message. Optionally, peers might advertise their ability to provide flows by sending f.offer messages to some of their neighbors.

The protocol for the request mechanism consists of three different message types. As described above, the dissemination of buffer maps is done periodically by piggybacking an r.offer message to any of the connection control messages or payload messages. A peer who receives such a buffer map might then issue an r.request message that contains a schedule addressing individual blocks. The number of concurrently requested blocks as well as their prioritization depends on the request mechanism and does not influence the protocol design. A request is accepted implicitly by sending one of the requested blocks. Other than for flows, blocks being received as result of a request are acknowledged to allow rate-controlled sending as well as failure detection on the sender side. If the receiver of an r.request message is not able to deliver the requested blocks because of its current load, it issues an r.deny_capacity message. How this message is then used to alter the request behavior is again up to the specific request mechanism.

Besides the aforementioned buffer maps and the possibility to offer flows, a peer includes status information with every message he sends. This includes his current playback position as well as the layer at which he is playing the video. This information, together with a one byte load field indicating the current upload workload, can then be used by the scheduler to assign requests and flows.

The definition of flows and requests should not be mistaken for the categorization into push- and pull-based scheduling. As described above, a flow might ultimately lead to pure pull-based scheduling if it is very short-lived, whereas a request might easily contain hundreds of blocks that are then delivered one after the other in a pure push-based fashion. To put it in a nutshell, the concept of flows and requests allows the same unification of protocols on the scheduling layer as the concept of contacts and connections did on the neighborhood layer. With the aforementioned division of the functional parts into a neighborhood and a scheduling layer and the communication over well-defined protocols, multiple different mechanisms can run in parallel and communicate with each other. This concept is an important step towards supporting transitions in the streaming overlay. During a transition, one or more mechanisms are altered or replaced, which is eased by the use of common protocols. The design choices that enable seamless transitions in a P2P live streaming overlay are presented in the following.

5.2 Supporting Transitions of Streaming Mechanisms

The focus of this thesis is the investigation of transitions between streaming mechanisms, such as the overlay topology and the scheduling mechanism. Such transitions become necessary due to dramatically changing environmental conditions. Examples are higher churn rates, leading to connections being dropped more frequently. As TRANSIT does not try to build and maintain a fixed overlay topology, it is by design robust to frequently changing neighborhoods. In a stable environment with little peer churn, the overlay converges towards a topology that resembles a multi-tree system. This is due to the fact that flows are negotiated with unlimited validity time, i.e. once a flow has been established, it is only canceled if the contract has not been fulfilled for some time. Consider the example depicted in Figure 5.6. Here, all peers want to retrieve all three layers of the video file. The negotiation of contracts leads to the implicit creation of distribution trees for each video layer. The characteristics of those trees are only determined by the direct negotiations between peers on one hop; there is no overall tree management involved. Therefore, the resulting subtree might as well be a chain, as is the case for the second layer (green). Other than in dedicated multi-tree systems, a peer's role is not limited by the construction scheme of the overlay. Most multi-tree based overlays would not allow a peer to act like peer D does, as D is an intermediate node for all three subtrees and, furthermore, acts as a parent that forwards all three subflows to peer B. The flow-based concept is thus much more flexible, as a peers role in the dissemination tree is

only limited by its bandwidth. If a peer is able to provide the whole stream, as is the case with D, other peers such as B will be able to retrieve the whole stream from just one parent.

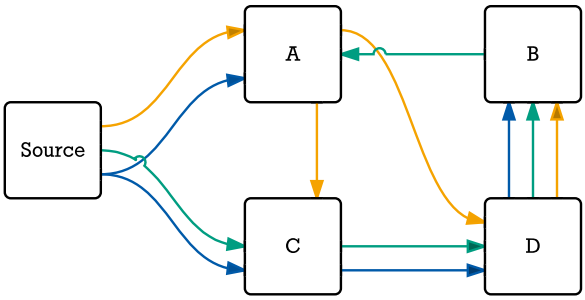


Figure 5.6.: Example of an implicit multi-tree overlay resulting from stable flows.

One very important consequence of this design is that peer D is able to act just like the source of the stream. This enables more stable peers to form a distribution system like LiveSky [YLZ+09] does with its CDN backbone, as presented in the related work. If a flow fails, peers repair it locally, i.e. by establishing a new flow with one of their neighbors. As the repair mechanism is bound to the direct neighborhood, such failures might propagate through parts of the overlay that depend on the broken flow. This would lead to renegotiations of flows and thereby to a change in the topology. In most tree-based systems, one tries to avoid such behavior by trying to maintain the complete subtree starting from the peer that detected a failure and just switching the parent of that peer. During this process, some signaling overhead is introduced as peers need to be notified of the maintenance operation in order not to start one on their own. Furthermore, latencies increase if the affected node is not able to find a new parent in time.

In TRANSIT, where the implicit trees are only used for the most recent packets, it is more beneficial to just let every affected peer repair its local incoming flow. Thereby, peers implicitly rebuild parts of the dissemination tree, which helps preventing unbalanced trees. Due to this, TRANSIT does not need to deploy maintenance protocols such as the *Up/Down* protocol introduced with Overcast [JGJ+00] or a fully centralized scheme such as the one used in CoopNet [PWC03]. An example of such a reorganization of flows is shown in Figure 5.7. Here, peer C fails which leads to the dashed flows being canceled. A possible resulting topology is shown on the right, with new flows being established by peer B as it no longer receives a complete flow. Notice that due to unidirectional connections peer B does not establish an incoming flow with peer A, as long as the connection between A and B is still open.

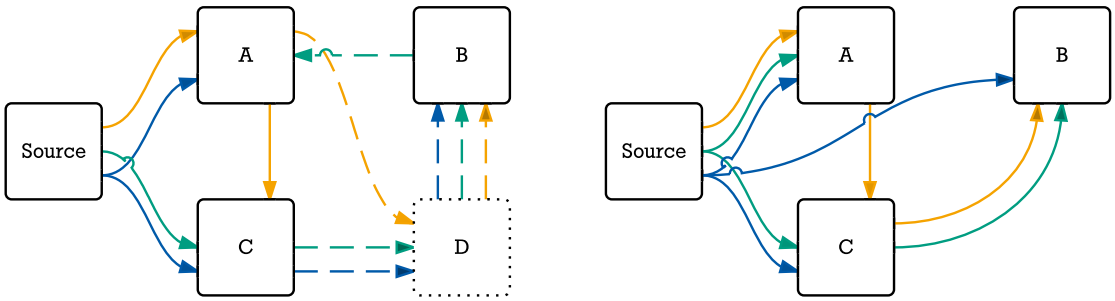


Figure 5.7.: Example of a node failure leading to new flows and an altered topology.

If peers join and leave the overlay more frequently, the lifetime of flows become much shorter. This, in turn, increases the number of requests needed to compensate for missing blocks during the flow negotiations. This shift in the ratio of flows vs. requests is considered a transition between a push-based and a pull-based scheduling mechanism. As a consequence of the shorter lifetime of flows in this environment, neighborhoods tend to change more frequently, which is also characteristic for mesh-based streaming systems. A requirement defined in Chapter 4 is the convergence of the system towards a stable state if the environmental conditions do not change significantly anymore. This is achieved by favoring flows over a pure request-based system. Once a peer is able to establish an incoming flow, the amount of requests issued by that peer will decrease significantly, as now most blocks are pushed to that peer. After some time, most peers form incoming flows again, and large parts of the system again behave like a tree-based streaming overlay. This convergence towards the flow-based scheme ensures low-latency delivery under stable conditions as well as little overhead, as flows are negotiated only once. The concepts of connections, flows, and requests as introduced in this chapter enable seamless transitions between push and pull-based scheduling and their respective topology characteristics.

5.3 Supporting Heterogeneous Peers

Another requirement stated before is the support for heterogeneous peers. We consider two types of heterogeneity: heterogeneity in the access link data rates and heterogeneity in the content that is demanded. Consider a mobile phone and an office computer that are both used to watch a video stream. The office computer is assumed to be connected to the Internet via a fast fiber link, whereas the mobile phone might use a much slower cellular network: we observe a heterogeneity in the access link data rate. However, while the office user might want to watch the stream in the best possible quality, the display of the mobile phone might not even support the highest quality. Downloading the same stream on both devices might therefore result in stalls and high computational overhead on the mobile phone. Using SVC encoded material in TRANSIT allows the mobile device to retrieve the stream at a lower bit rate and quality. However, as those lower layers requested by the mobile device are also needed to encode higher quality layers of the video, the mobile device can still contribute to the streaming overlay by sharing received video blocks. Thereby, SVC prevents the overlay from becoming segmented based on interest in different parts of the video stream, as all peers still share a common interest at least in the lowest quality layer of the stream. In TRANSIT, each peer specifies upon startup which layer it wants to retrieve. The scheduler of a peer only requests blocks that belong to layers a peer wants to retrieve, and a peer only forwards flows of layers below or at its requested quality.

Figure 5.8 depicts the well known flow structure of peers A to D. Now, we include some peers with less computational power or access link data rate, which makes them request lower quality video layers. In our example, peer M requests the lower two layers and receives them as flows from peers B and D. Peers N, O and P only request the lowest layer. Often, such peers would not forward the flow to other peers due to their constrained upload bandwidth, but in this example peer O even forwards the lowest layer to peer P. The aforementioned mechanism of neighborhood selection and flow repair leads to such peers being more often connected as leaf nodes due to their low upload capacity. Remember, peers A to D provide the same service as the streaming source and have sufficient upload capacity, making them amplifier nodes in the network. Even if a low capacity peer has no outgoing flows, he might still contribute to the overall system performance by answering requests.

In this thesis, we do not consider peers to adapt their requested video quality in terms of layers during streaming. Abboud et al. [APK+11, AZP+11] propose a system that determines an initial layer based

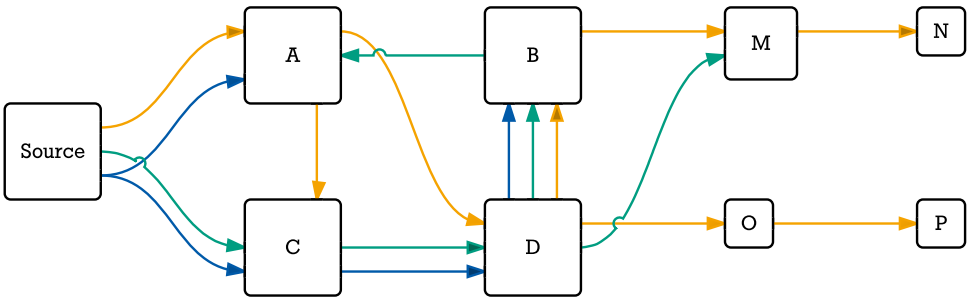


Figure 5.8.: Example of flows on heterogeneous peers. Peers with fewer resources receive only parts of the full stream.

on different strategies, such as upload bandwidth or device capabilities. The initial layer is then used as an upper bound, as peers are able to change their streamed layer based on the overlay performance or the availability of blocks in the neighborhood. The authors show that such a mechanism can significantly reduce the number of stalls, especially in bandwidth constrained scenarios. This results in a higher perceived quality for the user when compared to a non-adaptive streaming policy. These findings are further strengthened by Hu et al. [HGL11], who conduct extensive studies on the impact and benefits of SVC-aware scheduling mechanisms. The authors conclude that SVC-aware scheduling can be used as an incentive mechanism for peers to maximize the overall social welfare of peers. The design of TRANSIT does not explicitly include such mechanisms, as they do not contribute to the issue of transitions in P2P video streaming as such. SVC is an enabler for the support of heterogeneous devices, and numerous optimizations for SVC-aware scheduling exist. However, in this thesis we use a simple mechanism that only considers the initial layer selection and do not include adaptive layer selection mechanisms. Adding such mechanisms to further improve the system performance and to tackle issues such as providing incentives for peers to contribute their upload bandwidth is considered future work.

Up to now, we presented the design of TRANSIT as a streaming system with support for seamless switching of topologies and scheduling mechanisms. We therefore introduced the separation into neighborhood layer and scheduling layer and the service interface provided by the concepts of connections. Furthermore, the possibility for seamless transitions in the scheduling scheme and the resulting changes in the overlay topology have been discussed. In the following, we concentrate on TRANSIT as a testbed for different mechanisms, as the evaluation of transition possibilities and their performance is a key aspect of this thesis.

5.4 Transit as a Testbed for Transitions

Reconsider the system design as shown in Figure 5.9, this time with mechanisms being highlighted. As already discussed, TRANSIT defines the interface between neighborhood and scheduling layer, as well as the protocol between two instances of the same layer. In order to evaluate the impact of transitions between mechanisms on the behavior and performance of a streaming system, one has to compare the system with enabled transitions against its behavior with fixed mechanisms. Consider for example the aforementioned concept of flows and requests and their suitability for transitions. If TRANSIT is configured

to not negotiate flows at all and instead only rely on the request mechanism, we observe pure pull-based scheduling. On the other hand, by limiting requests to connections with a healthy flow and at the same time limiting the number of connections on the neighborhood layer, the system can only act as a multi-tree system with failure recovery. This fine-grained control over the mechanisms involved as well as the possibility to substitute mechanisms with different algorithms are the key aspects of TRANSIT as a testbed for transitions.

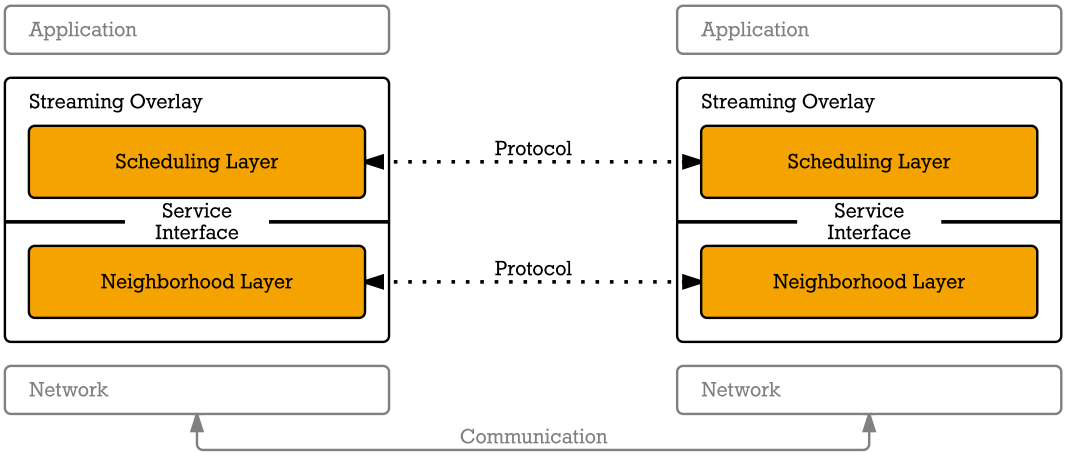
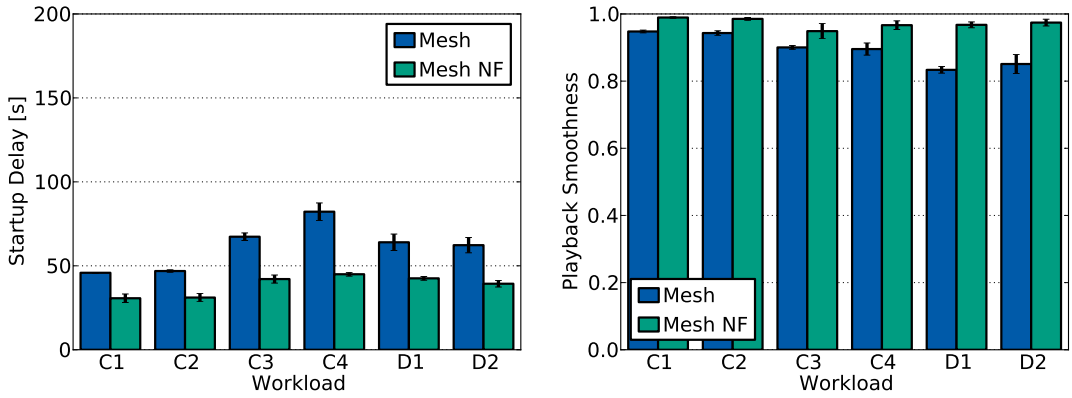


Figure 5.9.: Mechanisms on the neighborhood and scheduling layer of the streaming system.

For this thesis, two configurations of TRANSIT as a pure mesh as well as one multi-tree configuration are used in the evaluation as presented in Chapter 7. The multi-tree configuration limits the number of layers per flow to exactly one, thereby forcing different flows per layer. Furthermore, the amount of incoming connections is limited to the ones being used for active flows as well as a few backup connections used in case of a flow failure. How this translates into system parameters is explained in detail later on, when the evaluation of TRANSIT is presented. To compare TRANSIT against a pure mesh, one just needs to disable the flow mechanisms. However, flows are very important in a live streaming system, as they ensure that the most recent blocks are disseminated as fast as possible throughout the overlay. A mesh solely relying on the exchange of buffer maps and requests based on the urgency of the block for the individual peer will thus lead to content bottlenecks. In such a situation, the most recent blocks are available only on a very limited number of peers. This has severe impact on the startup delay of new peers, as they request the most recent blocks, which are not yet available in large parts of the mesh. Therefore, TRANSIT provides a second configuration of the request manager, where the most recent packets available on a neighbor are requested, even if they are not yet required for playback. This configuration is termed *newest-first*, and leads to significantly shorter startup delays of new peers as well as an overall increase in the playback smoothness.

The *newest-first* request strategy can be compared to the *rarest-first* strategy that is for example used in BITTORRENT. In a live streaming scenario, the most recent block published by the source is at the same time the rarest block, as it is only available on the source. Therefore, the rarest-first strategy translates into the aforementioned newest-first strategy in a live streaming overlay. How this affects the startup delays and playback smoothness, i.e. the ratio of playback time vs. stalling time, is presented in Figure 5.10. Here, a pure mesh and the mesh with newest-first strategy are compared against each other, using a selection of the workloads introduced later on for the evaluation (Chapter 7). Without going too



(a) Startup delay d_{su} **(b)** Playback smoothness Ψ

Figure 5.10.: Impact of the newest-first selection strategy on the performance of a pure mesh. With newest-first, peers experience significantly shorter startup delays, as their initially requested blocks are already spread throughout the overlay. The overall playback smoothness increases as well.

much into detail, the newest-first strategy helps in reducing the delay before a peer can start playback and reduces the amount and length of stalls, leading to a smoother overall playback.

Besides switching between different scheduling mechanisms, TRANSIT enables fine-grained configuration of the neighborhood layer. This includes the limitation of incoming and outgoing connections as well as their lifetime if they are not used for the transmission of payload. These parameters are described in detail in Chapter 7, where their impact on the performance of TRANSIT as a streaming system is evaluated. In the following chapter, the implementation of the proposed system for the P2P overlay simulator PeerfactSim.KOM is described.

6 Implementation

In this chapter, the implementation of TRANSIT is presented. TRANSIT is a modular P2P overlay for live video streaming. In contrast to the systems discussed in Chapter 3, it is not limited to one specific overlay topology or scheduling mechanism. Instead, it can be configured as either a tree-based or a mesh-based overlay or a hybrid combination of both. It supports seamless transitions between any of those topologies as well as any combination of push- and pull-based scheduling. TRANSIT enables a systematic evaluation of a multitude of overlay configurations under varying environmental conditions. All software has been implemented in Java as part of PeerfactSim.KOM, an event-based simulator for P2P overlay networks. Before describing the implementation of TRANSIT, we provide a brief overview of the simulator in the following section.

6.1 Overview of the P2P Overlay Simulator PeerfactSim.KOM

PeerfactSim.KOM [SGR+11] is an event-based simulator for P2P systems, written in Java. The simulator is structured as shown in Figure 6.1, providing a layered architecture based on the ISO/OSI model of communication systems [Tan81]. The layered architecture with its well-defined interfaces for each layer enables the interchangeability of implementations for a given layer. Thereby, one is able to simulate multiple overlays with the same application and network model, for example.

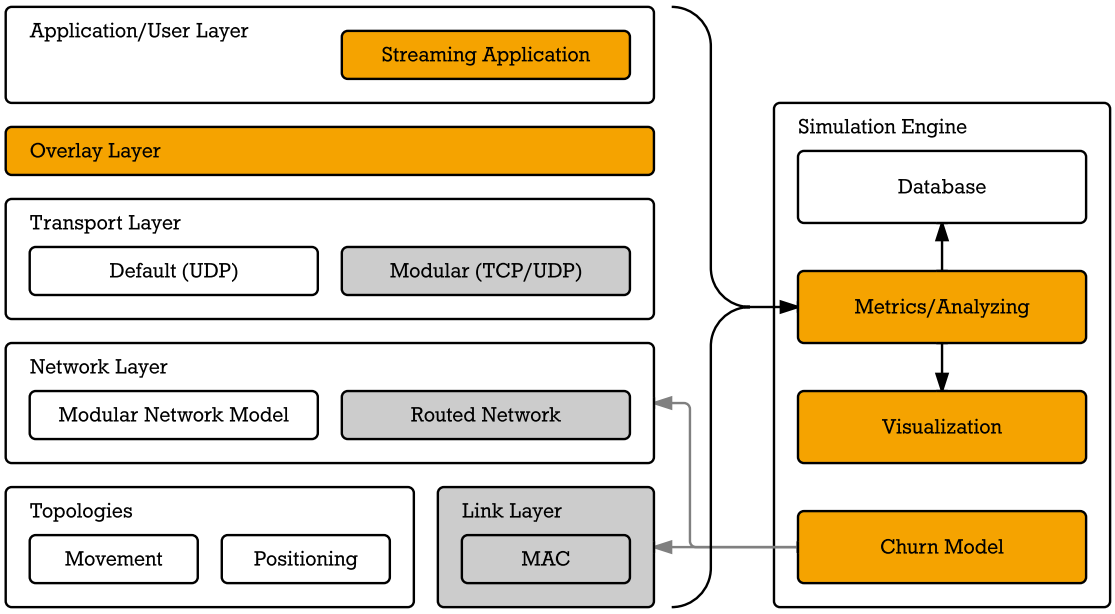


Figure 6.1.: Overview of the layers and simulation engine of PeerfactSim.KOM. Highlighted parts are extended (gray) or implemented (orange) in this thesis.

Heart of the simulator is the simulation engine, which triggers scheduled events and thereby actions in the overlay. As a discrete event-based simulator, PeerfactSim.KOM schedules events based on a discrete

clock. With each simulation step, the clock is directly increased to the time point at which the next event is scheduled. The event itself is processed by an event handler, which is usually the component that scheduled the event. Each component on any of the layers can register itself as a handler for events, schedule its own events and execute actions as reactions to those events. Consider, for example, an overlay that wants to periodically send a message to another peer. One possibility to implement this behavior is to register the overlay as an event handler and to re-schedule the event each time the previous event fired. Simplified, the code would look like the example in Listing 6.1. The method sendPeriodicalMessage() schedules a new simulation event two minutes in the future. When this event fires, it triggers the eventOccured()-method and thereby the next execution of sendPeriodicalMessage().

```
public class OverlayPeer implements SimulationEventHandler {
    public void eventOccurred(SimulationEvent se) {
        sendPeriodicalMessage();
    }
    public void sendPeriodicalMessage() {
        // [...] Code is executed every two minutes after first execution
        long nextTime = Simulator.getCurrentTime() + 2 * Simulator.MINUTE_UNIT;
        Simulator.getScheduler().scheduleEvent(nextTime, this);
    }
}
```

Listing 6.1: Scheduling and handling of a simulation event.

Scheduling events this way becomes tedious as soon as multiple different actions are to be executed. One would have to differentiate multiple events, which would inflate the eventOccured()-method. Therefore, another possibility to execute tasks is to schedule an operation rather than implementing SimulationEventHandler. An operation is a self-contained event handler and can best be compared with the concept of threads in Java. Once the operation is finished it triggers a callback, that can be used to further process the results of the operation. Thereby, the execution of code in PeerfactSim.KOM becomes asynchronous. Within the overlay layer, most actions are triggered by received messages which are in turn events being generated on the network or link-layer of the simulator.

The core goal of a simulation is of course to gather evaluation results by measuring various aspects of the overlay or application under test. Therefore, PeerfactSim.KOM provides an extensive analyzing interface. Analyzers are informed of specific events, for example the execution of an operation or a message being received at the transport layer. Therefore, general metrics such as traffic can be collected by implementing an analyzer that reacts to transport layer messages and stores the size of messages being sent or received for each node. The information collected by the analyzer is then persisted in a relational database for later processing. While analyzers are built to collect metrics on the interfaces of a specific layer or for a specific operation, they are not well-suited to collect metrics that are specific to individual overlays. For the evaluation of the streaming system presented in this thesis, a more flexible approach to collecting and evaluating such metrics has been implemented in PeerfactSim.KOM. This approach is presented in Section 6.4.

Another important component of the simulation engine is the churn model, which in the case of the streaming system presented in this thesis is directly intertwined with the user model. A churn model triggers join- and leave-events on individual peers, for example based on a statistical distribution or a measurement trace. The distinction between a churn model and a user model is motivated by the layer that the model interacts with. While the user model is interacting with the application layer, the churn model is directly interacting with the network or link-layer of a peer. A user model can thus be very

complex, in that multiple methods of a specific application can be triggered. A churn model just triggers the goOnline() and goOffline() methods of a peer. However, the application may register as a listener that is informed as soon as the connectivity changes due to a churn event. This leads to the aforementioned connection between a churn model and the user model. The application and user model used for the simulations of TRANSIT is presented in the following section.

6.2 Realization of the Streaming Application and the User Model

The application used in this thesis is based on earlier work on Video on Demand (VoD) streaming by Abboud et al. [AZP+11] and has been extended to support live streaming as well. The playback process is triggered by churn events, as depicted in Figure 6.2. As soon as a node goes online, its application is started and connects to the tracker. The tracker replies with meta information on the chosen stream and the initial neighborhood for the peer. This meta information consists of the current playback position of the source in terms of chunks as well as information about the SVC layers that are available for the given stream. Based on the meta information, the peer selects the SVC layers it wants to receive out of the full stream. In the current state of TRANSIT, this decision is fixed by the configuration of the peer. However, as already implemented for the VoD streaming overlay by Abboud et al., one might add layer selection mechanisms based on the bandwidth or computational resources of a peer. Immediately after selecting the SVC layers the peer starts to download matching blocks into the buffer. Playback starts, as soon as the buffer is filled to a given threshold.

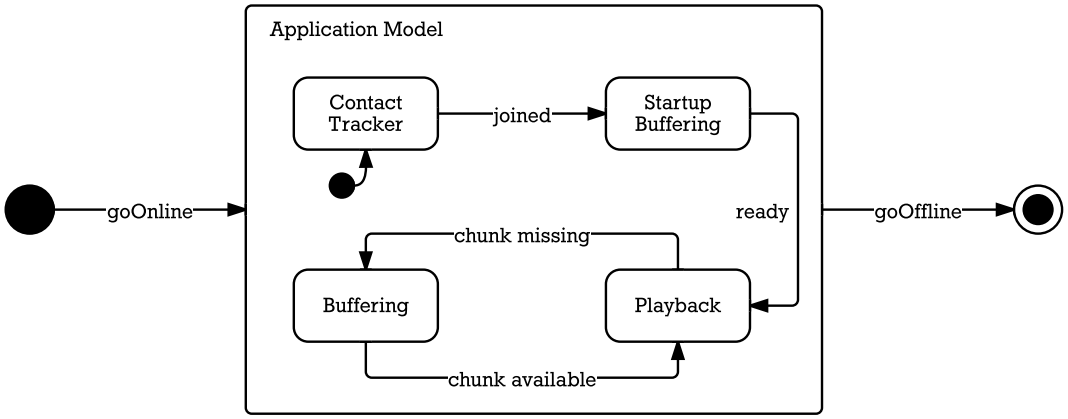


Figure 6.2.: The application model is initiated by churn events. Once a peer goes online, the application joins the overlay and starts streaming. If the node goes offline, the application is stopped.

If a chunk is not available in the buffer at the time it is supposed to be played, the player stalls until the chunk has been successfully downloaded. This process continues until the node goes offline, again triggered by a churn event. Thereby, in TRANSIT, the user model is equivalent to the churn model. This also implicates that peers do not leave the overlay due to stalls or high startup delays. For the evaluation of TRANSIT, two churn models have been implemented in order to realize the workloads defined later on in Chapter 7. A peer is only used once during one simulation, i.e. once the peer went offline, it will not join the overlay again. This helps in preventing undesired effects, such as states not being reset completely, while at the same time it leads to performance issues for large-scale simulations with thousands of peers. Currently, the simulator does not support resetting peers in order to reuse them at later points in the simulation. Even worse, all peers are fully instantiated during startup, leading to unnecessarily

high memory consumption. Future versions of the simulator might provide a resetting feature and lazy instantiation, which would enable large-scale simulations of streaming overlays.

In order to determine if a chunk is available for playback, the application has to communicate with the overlay itself. Therefore, TRANSIT offers an interface that exposes the current state of the buffer to the streaming application. This interface, together with the other components of TRANSIT, is detailed in the following section.

6.3 Transit – a Transition-Enabled P2P Video Streaming Overlay

TRANSIT as a modular streaming system has been implemented within the overlay layer of the simulator.[1] It exposes its functionality through well-defined interfaces, the most important one being the BufferInterface. Figure 6.3 gives an overview of the components that are part of TRANSIT, with interfaces to the application layer being highlighted. All components are part of the TransitNode, which also provides basic overlay functions such as join() and leave() through the StreamingNodeInterface. The BufferInterface provided by the scheduler is used by the application to control playback of a video and to determine, whether the next chunk of the video can be played. Other components, such as the NeighborhoodManager and the FlowManager, realize the concepts discussed in Chapter 5 and are not accessed outside of TRANSIT. Before going into detail on those components, the interface between the application layer and the streaming overlay defined in this thesis is presented.

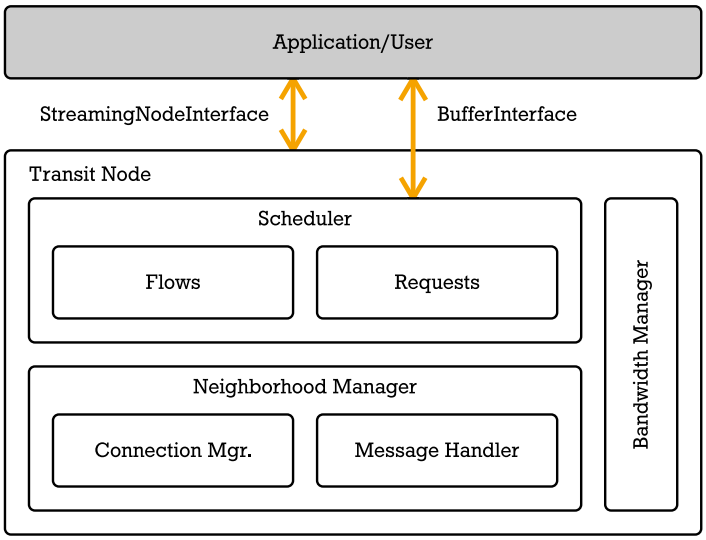


Figure 6.3.: Overview over the components of TRANSIT with interfaces to the application layer being highlighted.

The BufferInterface provides access to the state of the video buffer, as already mentioned. The interface specifies the following methods for the application layer, that allow its usage in both, a live streaming session and a VoD use case.

[1] All sources are located in the package de.tud.kom.p2psim.impl.overlay.streaming.transit. Components of the TRANSIT streaming overlay are often prefixed with Transit*, i.e. TransitFlowManager. For simplicity and readability, this prefix as well as the full class path are omitted throughout this chapter.

tryToPlay(int chunk) This method is called by the streaming application to play the given chunk of the video. It returns true if all blocks of the chunk at the requested layer are available and the playback position can be incremented. In this case, after the playback time of the chunk is exceeded, the method will be called again with the incremented chunk number. If the method returns false, the player has to stall and periodically call this method with the stalling chunk, until the playback continues.

getCurrentBufferLength() Returns the number of chunks in a row that are available at the requested video layer, starting from the current playback position. This information can be used for example during the startup phase to buffer the video up to a given threshold, before the playback is started for the first time.

Additionally, methods and callbacks for the implementation of a layer switching algorithm are provided. This allows future versions of TRANSIT to be extended with mechanisms that adapt the requested SVC layer based on the availability of blocks or the achieved throughput of a node. Abboud et al. [AZP+11, APKS09] propose such a mechanism, called Progressive Quality Adaptation (PQA). The authors show that mechanisms such as PQA reduce the number of stalls and their length, especially in bandwidth-confined scenarios. Before looking into how the scheduling layer is implemented in TRANSIT, the basic communication mesh and the realization of connections as interface between scheduling layer and neighborhood layer is presented.

6.3.1 Neighborhood Layer

The neighborhood layer is implemented according to Figure 6.4, with connections being the central element and interface to the scheduling layer. The neighborhood exchange mechanism is controlled by the NeighborhoodManager, which in turn provides the ConnectionManager with new contacts. The ConnectionManager then uses these contacts to form new incoming connections. As soon as a new connection is opened, all components that registered themselves as ConnectionListener are informed. This way, the scheduling layer is notified of a new connection and begins to use this connection to negotiate a schedule or to issue requests. The flows and requests – while being part of the scheduling layer – are directly stored on the connection objects. The objects containing flows and requests, namely FlowSchedule and RequestSchedule as well as the corresponding listener are presented in the next section.

In the following, a brief summary of the ConnectionManager and the corresponding ConnectionListener is provided, as those components are directly accessed by the scheduling layer components to retrieve connections. The ConnectionManager provides the following methods to the scheduling layer:

getConnections(Direction, State, FlowState, RequestState) This method is very important for the scheduler as it allows to retrieve a list of currently available connections matching a set of criteria. First of all, the direction of the connection can be specified as either incoming or outgoing. Second, one can filter for connections with a given state, with state being one of OPEN, NEGOTIATING, CLOSING or CLOSED as already presented in Chapter 5. Obviously, the scheduling layer will be interested mostly in connections that are open. With FlowState and RequestState, the scheduling layer is able to filter connections based on the states of the flow schedule and the request schedule, respectively. This way, one can query the ConnectionManager for example for a set of incoming, open and currently unused connections.

addConnectionListener(ConnectionListener) Adds a listener that is notified as soon as the state of a connection changes. The ConnectionListener is used to gain knowledge of newly established connec-

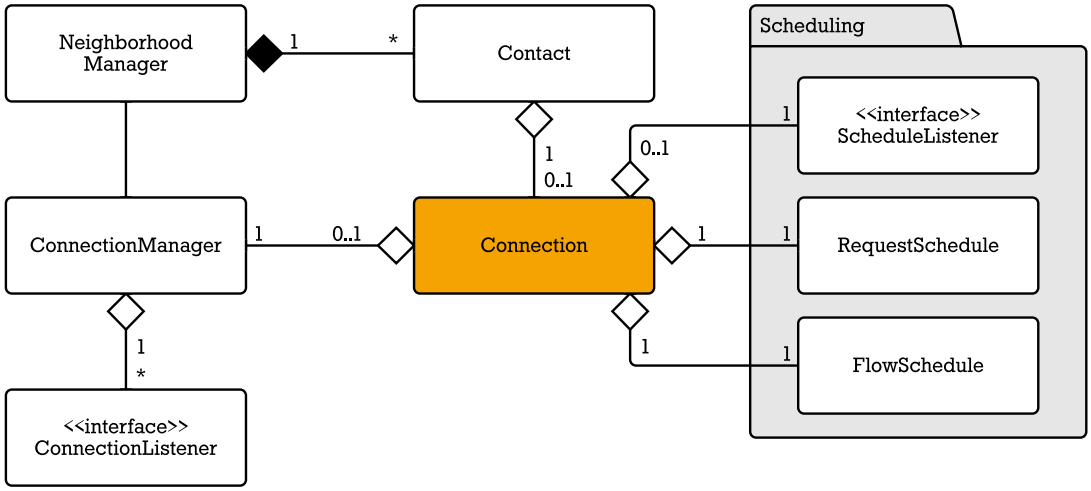


Figure 6.4.: Components of the neighborhood layer and interfaces to the scheduling layer provided by the concept of connections.

tions in the scheduling layer. It furthermore enables the scheduling layer to react to a connection teardown by canceling the corresponding requests or by establishing replacement for flows that were previously active on the connection.

The neighborhood layer does not provide methods to actively form new connections, as this task is solely up to the ConnectionManager. The initiation of connections is based on the exchange of neighborhoods and the activity of existing connections. If new contacts are provided and one of the current connections has not been used for the transmission of payload for a given time, the connection might be replaced with a new one. Thereby, connections that are *useful* in that they provide blocks are kept open, whereas unused connections are replaced periodically. Utilizing connections for requests and flows is the task of the scheduling layer, which is presented in the following section.

6.3.2 Scheduling Layer

The scheduling layer of TRANSIT is divided into three functional parts. The concept of flows and requests, as presented in Chapter 5, is directly translated into two components, namely the FlowManager and the RequestManager, as shown in Figure 6.5. The core responsibility of those components is to assign flows and requests to connections and to execute the protocols as defined in Chapter 5. As already mentioned, the resulting flows and requests are stored as FlowSchedules and RequestSchedules, respectively, which are then assigned to a connection.

To distinguish between active and non-active requests and flows, the connection furthermore stores a state for each of the schedules. A RequestSchedule can be inactive, active or canceled, with the last denoting that the request was not fulfilled. As peers exchange buffer maps and only request blocks of which they know that they are available on the neighbor, requests are only canceled by the sender in case of upload capacity shortages. A FlowSchedule has one additional state, negotiating, while it is requested but not yet confirmed by the sender.

The assignment of requests and flows is based on the current state of a peer's buffer. This buffer is managed by the Scheduler, as already mentioned when introducing the BufferInterface. In order to

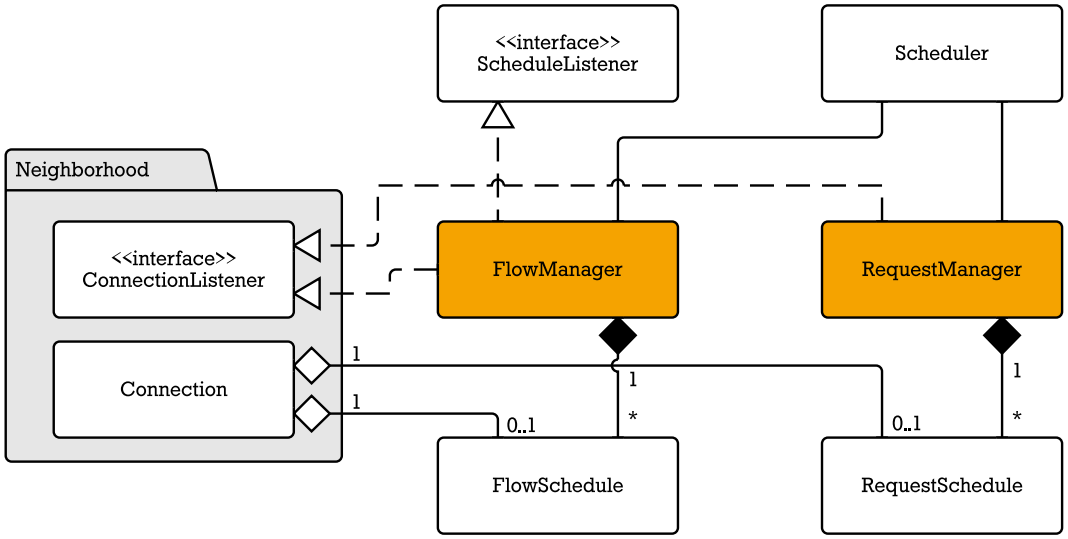


Figure 6.5.: Components of the scheduling layer in TRANSIT and the communication interfaces with the neighborhood layer.

provide said interface, the scheduler has to keep track of all blocks that were previously received by the peer. This is modeled by the StreamingDocument, which was originally implemented for the VoD streaming overlay used by Abboud et al. [AZP+11]. Internally, each block is managed as a boolean variable stored in a BitSet. This is a reasonable approach for the VoD setting, where the number of blocks is limited by the video length. However, in the live streaming scenario, the video length is not limited, but instead only determined by the length of a peer's session. Therefore, the StreamingDocument in TRANSIT uses a sliding window approach and limits the number of stored bits to the length of a peer's session.

This concept also motivated the design of the buffer maps that are exchanged as part of the request mechanism in TRANSIT. Instead of sending the peer's whole buffer, only a small subset consisting of a few chunks is sent to the neighbors. To maximize the use for the neighbor, this subset starts with the last known playback position of the respective neighbor. As peers include their current playback position with every message they send, this method leads to small but useful buffer maps, thereby reducing the overhead when compared to the static approach. Buffer maps are piggybacked periodically, in order to lower the message overhead. The mechanism that enables piggybacking in TRANSIT is briefly described in the following section.

6.3.3 Message Handler and Bandwidth Management

All messages in TRANSIT are dispatched through the MessageHandler. This class unifies the detection and handling of timeouts, as well as the estimation of RTTs. However, its most important property is the support for piggybacking of messages. TRANSIT is the first overlay in PeerfactSim.KOM that provides an interface for the piggybacking of data on sent messages, as shown in Figure 6.6. By piggybacking data such as buffer maps or neighborhoods, the amount of messages being sent is reduced. Other components can register at the MessageHandler as a PiggybackListener. The listener is queried each time a new message is sent to a peer, which in most cases will be a transmission of a video block. It may then provide additional data, which is piggybacked on top of the message. Most information needs to be exchanged

periodically, which is why there is an additional implementation of the handler that is queried only once in a specified interval.

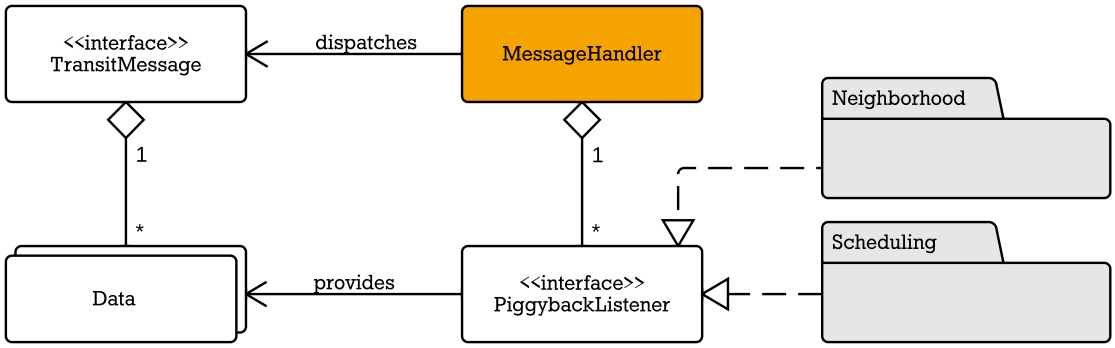


Figure 6.6.: The message handler and its piggybacking mechanism. Components of the neighborhood layer or the scheduling layer register as listeners and provide their data that is to be piggybacked.

When sending messages, the available bit rate on the access link is not to be exceeded, as otherwise packets are dropped by the network. Therefore, the sending rate of a peer is monitored in a sliding window approach. If it approaches the maximum available bit rate of the peer, new requests are no longer accepted or delayed until the bit rate needed for those requests becomes available again. The download bit rate is monitored as well, as a peer is only allowed to request new packets while its download link is not fully saturated. In addition to the bit rate, the RTTs on all connections are measured as well. Thereby, the overlay is able to detect saturated or congested peers and redirect requests accordingly.

To quickly evaluate design decisions and their impact on the metrics presented later on, a new analyzing structure was implemented for PeerfactSim.KOM. This new structure enables the collection of metrics in a unified way and adds features such as live plotting. This mechanism, together with the visualization implemented for TRANSIT, is presented in the following section.

6.4 Analyzing and Visualization in PeerfactSim.KOM

As already mentioned in the previous sections, PeerfactSim.KOM offers an extensive analyzing interface. However, this analyzing concept is mostly intended to monitor the interfaces between two layers of the simulator. It is therefore not straight forward to monitor the state of the overlay, for example the current buffer state of a peer, using this analyzing interface. For this thesis, a whole new monitoring concept for PeerfactSim.KOM was designed and implemented. Most important, this concept allows the easy observation of overlay-specific states. It thereby seamlessly integrates into the existing infrastructure, for example by using the same database abstraction layer as the original analyzing interface. Rather than implementing analyzers, with the new concept, one implements *metrics*. In this context, a metric is defined as a value that is to be monitored and should thus be accessible for later processing.

Figure 6.7 illustrates the flow of data for this concept. The overlay or any other component in the simulator provides metrics, so called *primitives*. Those metrics can then be passed to filters, which perform arbitrary calculations on their input metrics. An example of a configuration used for the evaluation of TRANSIT is provided in Appendix A. For TRANSIT, the following filters have been implemented and are used throughout the evaluation:

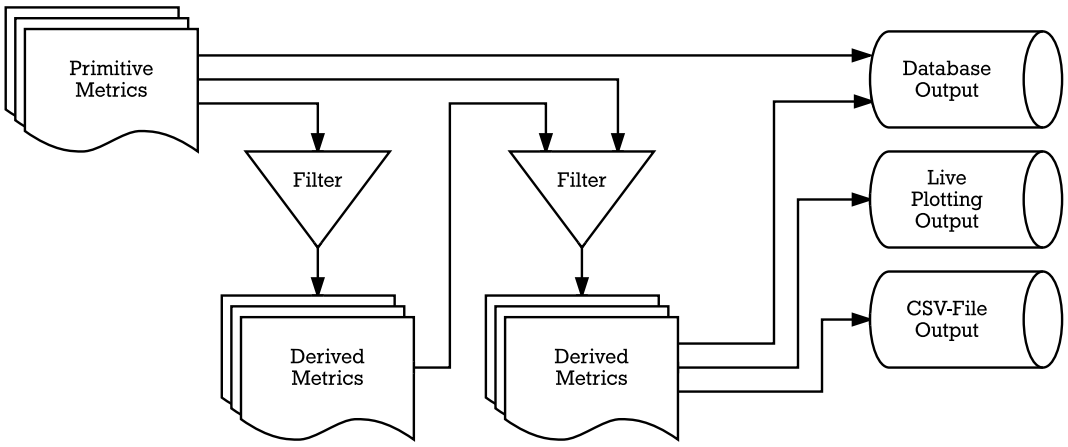


Figure 6.7.: Data flow based model of the metric analyzer, with primitive metrics being provided by, for example, the overlay node implementation. These metrics can then be passed to filters, that execute various tasks such as sampling and calculating ratios or statistical properties. Finally, metrics can be passed to outputs, that persist or present the metrics.

Periodic Sampling Filter This filter periodically samples a metric, and captures the sampled value till the next sampling period. This is, for example, used to write metrics in predefined intervals into the database, as explained later on. The sampled value is provided for further filtering as a derived metric.

Interval Delta Filter Primitive metrics often just count a given parameter, for example the number of sent messages. However, in order to calculate a derived metric for an interval, only the increase in the value since the last interval is needed. This filter therefore acts just as the periodic sampling filter, but provides the difference of the value since the last sampling interval as a derived metric.

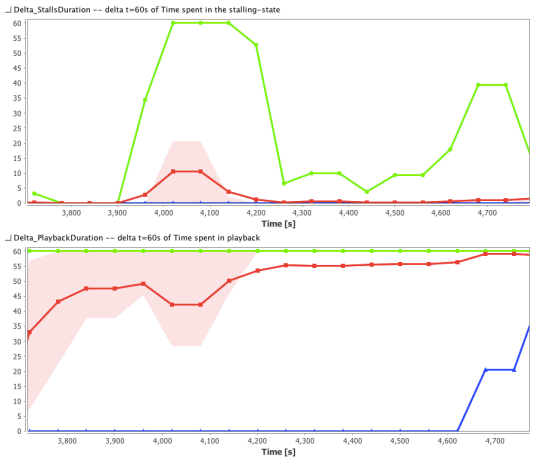
Ratio Filter This filter calculates the ratio between two input metrics and returns the result as a derived metric. It is, for example, used to calculate the workload of a peer as the fraction between current bit rate and maximum bit rate.

Statistics Filter One can perform various aggregations and computations on metrics that are sampled on a per-host basis. Therefore, this filter provides often used statistical functions that transform a per-host metric into an aggregated metric for further processing or visualization.

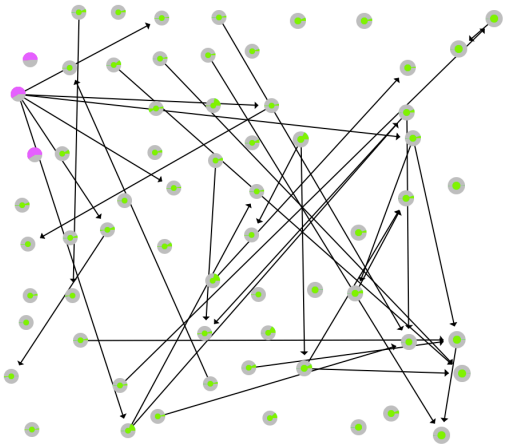
On Churn Filter Some metrics, such as the session duration of a peer, are only evaluated once in the lifetime of the peer. Therefore, this filter passes the input metric of a single host as a derived metric, as soon as the respective host leaves the system.

Metrics, that are updated or calculated based on simulator events or a periodic operation are called *active metrics*. Any output can register as a listener for updates of those metrics and, for example, write the updated value to the database. Thus, the interval after which a metric is persisted, solely depends on the configuration of the intermediate filter and may vary for different metrics. The metrics concept is very flexible, as one can easily combine existing metrics through filters to evaluate their dependencies. A commented excerpt of the metric analyzer configuration, showing this flexibility, is provided in Appendix A. Once a metric is implemented or created through filters, it can easily be persisted or visualized by using different output implementations.

An example of such a visualization is shown in Figure 6.8, where two metrics are visualized live during the simulation. The statistics filter is used to aggregate the per-host values, leading to the plots shown



(a) Live plotting of metrics



(b) Overlay visualization

Figure 6.8.: Live plotting of metrics with the respective output component. The plots can be configured to show minimum, maximum, averages and any percentiles. For small-scale simulations the visualization component shows overlay topologies as well as active flows and requests. Various other information, such as uplink utilization and playback status, are visualized as well. In this example, a mesh-based overlay is shown, where only the currently active requests are visualized.

in the Figure. Besides the visualization of metrics, TRANSIT provides a visualization component. This component visualizes the overlay topology and the scheduling mechanisms, for example by showing the flows that have been negotiated between peers. While the visualization is only helpful in small-scale simulations, it still provides a good starting point in understanding the mechanisms provided by TRANSIT.

To grasp the behavior of TRANSIT in large-scale scenarios and to understand the impact of transitions on the performance of a streaming system, extensive simulations are conducted. The results of those simulations and the evaluation of transitions in P2P streaming systems are presented in the following chapter.

7 Evaluation of Transit

TRANSIT is designed to be a testbed for transitions as well as a fully fledged video streaming system. Therefore, the evaluation is split into two parts with the first one focusing on evaluating and optimizing TRANSIT as an *adaptive* streaming system. The second part then focuses on the concept of what we defined as transitions and their impact on system performance and playback behavior. In the following, metrics are defined and the evaluation scenario is described. The scenario consists of different workload schemes, one of them being based on real-world measurement traces of the popular PPLIVE streaming system. Finally, the results of both evaluation parts are presented in Section 7.4 and Section 7.5.

7.1 Goals and Methodology of the Evaluation

As mentioned, the evaluation is divided into two parts, with the first part evaluating how well the general requirements that were stated in Chapter 4 are met. Therefore, the overlay is benchmarked under various scenarios, including real-world traces of the popular PPLIVE streaming system. The metrics for this benchmark are directly derived from the system requirements and a detailed explanation is provided in Section 7.2. Included are system-level metrics such as *protocol overhead* as well as application-level metrics such as *playback continuity*. The goal of this first part of the evaluation is to understand how different system parameters affect the overall performance of the overlay. The results from the benchmark define the default parameter settings of TRANSIT.

While the first part evaluates the system performance of TRANSIT, transitions are not explicitly considered in the evaluation. Therefore, the second part of the evaluation focuses mainly on transitions in P2P live streaming systems. We compare different scheduling schemes and overlay topologies and evaluate their performance under synthetic and real-world workload patterns. TRANSIT enables us to conduct this comparison as it can be configured as a pure mesh and pure tree system as well as any hybrid combination of both. Rather than benchmarking multiple specialized P2P systems against each other, we are thus able to benchmark the performance of the underlying mechanisms, for example pull- vs. push-based scheduling. This part of the evaluation motivates the support for transitions in a live streaming system by understanding when such a transition is beneficial for the system.

Recent work by Hu et al. [HGL11] shows the benefits of using the video codec SVC as introduced in Chapter 2. The authors evaluate different neighboring mechanisms and scheduling strategies in a mesh-pull scenario. The evaluated mechanisms specifically take the properties of the SVC encoded video material into account for example by using content of higher quality layers as incentive for other peers to contribute to the content dissemination themselves. They show that such a system can achieve a good trade-off between fairness, incentive and efficiency. In contrast to the work by Hu et al., we do not evaluate the benefits of SVC in the context of live video streaming. We use SVC solely as an enabler for the support of heterogeneous devices, by requesting only substreams of the full video stream on some peers. Of course, incentive mechanisms such as the one proposed by Hu et al. are important and interesting topics for future work, as presented in Chapter 8.

Another interesting concept of using SVC in order to provide a high and stable Quality of Experience (QoE) is proposed by Abboud et al. [APKS09, AZP+11], as already mentioned in Chapter 6. Here, the

authors adapt the layer that is requested by the scheduler during playback. The mechanism is called Progressive Quality Adaptation (PQA) and switches the playback layer of a peer depending on packet availability and past download throughput in order to prevent the peer from stalling. For their evaluation, the authors introduce a set of SVC video quality metrics. These metrics assess the frequency of layer switches as well as the quality relative to the initially chosen layer of the video. While such an progressive adaptation mechanism can easily be incorporated into TRANSIT, it would obfuscate the evaluation of transitions due to its adaptive nature. We therefore do not consider PQA throughout this evaluation, which is why there are no video quality metrics considered in the following section. However, it is planned to introduce quality adaptation concepts such as PQA in follow-on works on TRANSIT.

7.2 Evaluation Metrics

The metrics for this evaluation are directly derived from the requirements stated in Chapter 4. This motivates the breakdown of metrics into *system metrics* and *application metrics*. System metrics are measured directly at the overlay level, whereas application metrics can be measured at the application level and are thus independent of the overlay itself. Application metrics thereby enable a direct comparison of different overlays from a user's perspective. The choice of metrics presented in this section is further motivated by the work of Zhang et al. [ZH12]. The authors present a survey of P2P live video streaming schemes and define a set of evaluation metrics that are applicable to all kinds of streaming systems. These include *workload* and *overhead*, as defined in the following. However, additional metrics are considered in this thesis to study the impact of transitions and to get a better understanding of the playback quality of peers.

7.2.1 System Level Metrics

On the system level the following metrics are defined:

Workload

The workload W of a peer is defined as the ratio of currently used data rate R and maximum available data rate $\hat{R}$ on the access link. The workload is defined for the upload ($\uparrow$) and the download ($\downarrow$) direction of traffic as

$$W_{\uparrow} = \frac{R_{\uparrow}}{\hat{R}_{\uparrow}} \quad \text{and} \quad W_{\downarrow} = \frac{R_{\downarrow}}{\hat{R}_{\downarrow}} \quad . \tag{7.1}$$

In a P2P live video streaming system the download workload should roughly correspond to the bit rate of the video stream, at least over a longer observation time. Any increase in the download workload would indicate either duplicate packets being received or a significant increase in control traffic. The upload workload can be much higher, as one peer might simultaneously provide packets for multiple other peers. However, it is desirable that the workload is split evenly across peers. Due to the asymmetric nature of access links, the overall throughput of a P2P system is bounded by the upload rate of peers. Therefore, in a saturated system, each peer's upload workload should be roughly one.

Flow Ratio

This metric is specific to the TRANSIT overlay, as it describes the ratio of push-based scheduling versus pull-based scheduling. Incoming packets can be categorized as either received via a flow or delivered as

a result of a request. Packets that are received via a flow are collected in the set P_f, whereas packets that are delivered as a result of a request are collected in the set P_r. The flow ratio of a single peer is then defined as

$$R_f = \frac{|P_f|}{|P_f| + |P_r|} \quad , \tag{7.2}$$

with $|P_f|$ being defined as the size of the set P_f. A flow ratio of one indicates that all packets are received via push-based scheduling, whereas a flow ratio of zero means that packets are solely delivered via pull-based scheduling. This metric is especially important in the context of transitions, as it allows us to understand the current state of the topology. As explained in Chapter 5, a higher rate of flow-based packet delivery indicates longer lasting flows and thereby the formation of longer lasting distribution trees, which will lead to a convergence towards a tree-like topology. If, for example, most peers in the overlay are able to form distribution trees by negotiating flows, then the average flow ratio would be close to (or equal to) one.

Number of Connections and Frequency

The number of concurrent connections as well as the frequency of connection setups indicate how peers react to churn. We define the set of open incoming connections of a peer as $C_{\leftarrow}$ and the set of open outgoing connections as $C_{\rightarrow}$. The metric *number of connections* is given by $|C_{\leftarrow}|$ and $|C_{\rightarrow}|$, respectively. The frequency of connection setups is measured over a fixed interval Δ, referred to as *sampling interval*, and is defined for the k^{th} interval as the number of connections that were newly formed during this interval. Let $C_{\leftarrow}(k-1)$ denote the set of incoming connections during interval number $k-1$. Then, the relative complement $C_{\leftarrow}(k) \setminus C_{\leftarrow}(k-1)$ defines the set of incoming connections in interval number k that were not yet established in interval $k-1$. The frequency of connection setups for one peer in the k^{th} interval can thus be defined as

$$f_C(k) = \frac{|C_{\leftarrow}(k) \setminus C_{\leftarrow}(k-1)|}{\Delta} \quad . \tag{7.3}$$

If peers expose high churn rates, the frequency of connection setups should increase significantly, as peers try to replace dead peers. Likewise, if peers are more stable and provide value in terms of video packets, they tend to form long-lived connections and the frequency of connection setups decreases.

Protocol Overhead

In order to transmit a single packet of video payload, the overlay has to exchange an arbitrary number of control messages. This can be the request messages needed in pull-based scheduling, but it also includes messages that are needed to form and maintain connections and to exchange node information such as buffer maps and current neighborhoods. TRANSIT deploys a piggyback mechanism, where such status information can be sent as part of payload packets or other control messages. While this greatly reduces the overhead in terms of the number of messages being sent, it still has an impact on the overhead in terms of sent data. Therefore, overhead in TRANSIT is defined in two ways, with the first one being defined in terms of number of messages as

$$O_{\#} = 1 - \frac{\text{Received Video Blocks}}{\text{Total Messages Received}} \quad . \tag{7.4}$$

This metric does not consider the impact of piggybacking on the message size of transmitted video blocks. We define a second metric that takes into account the actual size of the data being transmitted as

$$O_s = 1 - \frac{\text{Received Bytes of Video Stream}}{\text{Total Bytes Received}} \quad . \tag{7.5}$$

For both metrics, duplicate video packets are counted as part of the overhead as they provide no further value to the overlay.

7.2.2 Application Level Metrics

The system level metrics are agnostic about the application. They do not provide insights into the quality of the video streaming process from a user's perspective. Therefore, the following application level metrics are considered and evaluated. Those metrics assess the playback continuity and the overall experience resulting from buffering delays and stalls.

Startup Delay

The startup delay d_{su} describes the time an individual peer has to wait after he joined, until the first chunk is played. It is determined by the time it takes a peer to connect to the overlay and to receive enough chunks for the player to start playback. This, in turn, is affected by the buffer size that is configured in the overlay. More interesting than the absolute value of this metric is its variance over different peers. In an ideal case, each peer should experience roughly the same startup delay. If the initial neighborhood of a peer is not able to provide requested packets, this can have significant impact on the startup delay. The peer would have to repeat the connection procedure until he finds neighbors that are able to provide video payload. Therefore, high variances in this metric are an indicator for problems in the initial connection procedure. A high startup delay might also indicate that the overlay is saturated, meaning that no spare bandwidth is available for the newly joining peer.

When evaluating the startup delay in terms of absolute numbers, one has to take into account the playback policy of the streaming overlay. In order to start playback, the application needs to first buffer a reasonable amount of blocks. In TRANSIT, a peer sets its initial playback position according to the information received by the tracker, i.e. at the current playback position of the source minus the delay it took the message from the tracker to the peer. Thereby, filling this initial buffer means waiting for the source to create new packets, as some of the packets needed to fill the buffer have not yet been created at all. This limits the minimal initial playback delay achievable by a peer in TRANSIT to roughly ten seconds with the current configuration of the initial buffer length. When considering user-perceived quality as introduced in Chapter 2, one might want to achieve lower startup times by arbitrarily shifting the initial playback position a few seconds into the past. Thereby, all packets needed to fill the initial buffer have already been published by the source.

Playback Smoothness

The playback smoothness describes how often a video stream of a single peer is interrupted due to missing chunks and how long those interruptions last. The metric is relative to the total length of the streaming session of that peer counted from the first successfully played chunk, i.e. the startup delay d_{su} of that peer does not influence the playback smoothness. With the session length of a peer in seconds

denoted by δ and duration(s) returning the length of a stall $s \in S$, the *playback smoothness* Ψ is defined as

$$\Psi = 1 - \frac{1}{\delta - d_{\text{su}}} \cdot \sum_{s \in S} \text{duration}(s) \quad . \tag{7.6}$$

This metric is equal to one, if the peer did not experience any stalls at all. It decreases linearly and reaches zero if the peer stalled over the whole measurement. The playback smoothness can also be calculated over a sampling interval Δ as $\Psi(\Delta)$, by dividing the time the peer spent in the playback state during this interval through the total time it has been active in the interval. Overall, this metric provides an indicator for the playback continuity of a peer, once the playback started for the first time.

Playback Experience

As the startup delay is not taken into account when calculating the playback smoothness, a second metric is defined that qualifies the whole session of a peer in terms of video playback. Therefore, the *playback experience* Υ of a peer is calculated as

$$\Upsilon = 1 - \frac{1}{\delta} \cdot \left(d_{\text{su}} + \sum_{s \in S} \text{duration}(s) \right) \quad . \tag{7.7}$$

This way, the startup delay d_{su} has an impact on the playback experience as well. Obviously, this metric can only be calculated over the whole session time of a peer. It is one, if a peer experiences absolutely no startup delay and no stalls during playback, which is not possible in a real-world scenario.

Name	Symbol	Unit	Description
Workload	$W_{\uparrow}$	none	Ratio of currently used upload bandwidth
	$W_{\downarrow}$	none	Ratio of currently used download bandwidth
Flow Ratio	R_f	none	Ratio of push vs. pull-based scheduling
Number of Connections	$\lvert C_{\leftarrow} \rvert$	none	Number of currently open incoming connections
	$\lvert C_{\rightarrow} \rvert$	none	Number of currently open outgoing connections
Connection Frequency	f_C	s^{-1}	Frequency of connection setups
Overhead	$O_{\#}$	none	Ratio of control messages vs. payload messages
	O_s	none	Ratio of control message size vs. payload size
Startup Delay	d_{su}	s	Delay until the first chunk is played
Playback Smoothness	Ψ	none	Playback continuity of a peer
Playback Experience	Υ	none	Combination of startup delay and playback smoothness

Table 7.1.: Overview of all metrics used for the evaluation of TRANSIT.

Table 7.1 summarizes the metrics presented above. The system level metrics allow us to gain understanding of the load on the system and some topology-related properties, as for example the connection frequency. They also provide a view on the cost that is represented by the overhead caused in the system, both in terms of messages and the total data being transmitted. The application level metrics provide insights into how the playback behavior is affected by different configurations or any other environmental influences, which is why they are considered to be performance metrics. The metrics introduced on the

application level focus on *objective* performance, i.e. the user-perceived quality is not taken into account. As already described in Chapter 2, one could also consider the user-perceived quality by adding QoE metrics, but this is not the focus of this thesis. Instead, the quality of the streaming process is assessed by the playback smoothness and the startup delay, or its combination as playback experience.

With the metrics presented above, the performance of TRANSIT is evaluated under various workloads, which are presented in the following section.

7.3 Simulation Setup

All evaluations are conducted by means of simulations using the simulation framework PeerfactSim.KOM, which is introduced in Section 6.1. The setup of these simulations is detailed in this section, with a focus on the workload models as well as the parameters of the underlying network model. Please consult Appendix A for excerpts of a configuration file used throughout this evaluation. Some metrics and workloads depend on a fixed *sampling interval* Δ, which is set to one minute according to Table 7.2. The workload models that are used for the evaluation of TRANSIT are presented in the following.

Name	Symbol	Unit	Description
Simulation Length	T_s	s	Length of the simulated scenario
Sampling Interval	$\Delta = 60\,s$	s	Minimal time-resolution of sampled metrics
Number of Sources	$\|S\| = 3$	none	Number of sources
Number of Peers	$\|P\|$	none	Total number of peers (excluding sources)
	$\|\tilde{P}\|$, $\tilde{P} \subseteq P$	none	Number of currently online peers
	$\|\tilde{P}\|_{max}$	none	Maximum allowed number of online peers
Access Link Bandwidth	$\hat{R}_\uparrow$	kbit/s	Maximum upload data rate of a peer
	$\hat{R}_\downarrow$	kbit/s	Maximum download data rate of a peer
Session Length	δ	s	Total length of a peer's online session
Arrival Rate	Λ	none	Number of peers arriving in the interval Δ
SVC Layer	$\|L\|$	none	Number of layers a stream provides
	$l = \{d, t, q\} \in L$	none	One layer of the SVC stream
	$l_{init} \in L$	none	A peer's initial SVC layer
Layer Bitrate	rate(l)	kbit/s	Bitrate of a video stream at the given layer

Table 7.2.: Overview of all parameters used in the simulation setup.

7.3.1 Workload Models

A workload model defines the distribution of peer arrivals and the distribution of session lengths. As soon as a peer joins the network, it starts the playback procedure and tries to stream the video starting with the current playback position of the source. Regardless of stalls and other issues, the peer will stay in the network for its full session length δ. For the evaluation of TRANSIT, we define multiple workload models to evaluate a broad range of environmental conditions. When evaluating transitions in a streaming system, the environmental conditions have to be varied and their impact on the system performance has to be evaluated. Therefore, the synthetic workload models presented in this section vary in their churn rates or simulate events such as a flash crowd with different intensities. The functionality of the system as a whole is furthermore evaluated using a trace-based model.

Workload A – No Churn

This workload model is designed to evaluate the scalability of the overlay and its functionality under ideal environmental conditions. Peers join with a constant arrival rate and stay online for the whole simulation duration T_s. Thus, the number of online peers $|\tilde{P}|$ at time t can be described as

$$|\tilde{P}|(t) = \begin{cases} 0 & \text{for} \quad t < t_{\text{j, start}} \\ |P| \cdot \frac{t - t_{\text{j, start}}}{t_{\text{j, end}} - t_{\text{j, start}}} & \text{for} \quad t_{\text{j, start}} \leq t \leq t_{\text{j, end}} \\ |P| & \text{for} \quad t > t_{\text{j, end}} \end{cases} , \tag{7.8}$$

with $t_{\text{j, start}}$ being the beginning of the join phase and $t_{\text{j, end}}$ being its end. As soon as all peers have joined the system, the number of peers remains constant. The arrival rate Λ for an interval Δ within the join phase can thus be defined as

$$\Lambda = |P| \cdot \frac{\Delta}{t_{\text{j, end}} - t_{\text{j, start}}} \quad . \tag{7.9}$$

Table 7.3 shows four configurations of this model. They differ in the total number of peers that are online at the end of the join phase. As the join phase has the same length in all configurations, the resulting arrival rate varies from five peers per minute to up to 120 peers per minute.

Parameter	Model A1	Model A2	Model A3	Model A4
T_s	3 h	3 h	3 h	3 h
$\|P\|$	300	1200	3600	7200
$t_{\text{j, start}}$	1 h	1 h	1 h	1 h
$t_{\text{j, end}}$	2 h	2 h	2 h	2 h
Λ	5	20	60	120

Table 7.3.: Variations of the scalability workload model A used in the evaluation.

During first evaluations, this workload model did not lead to significant results. As peers experience no churn, the playback performance during this workload is only affected by the joining phase of the peer and its ability to find suitable neighbors. However, investigating the joining behavior under ideal assumptions, i.e. without churn, does neither help in evaluating a suitable parameter setup for TRANSIT, nor does it provide any insights into the impact of transitions in a streaming system. To evaluate the scalability of the overlay in a massive join scenario, we therefore use the flash crowd scenario of workload C rather than workload A.

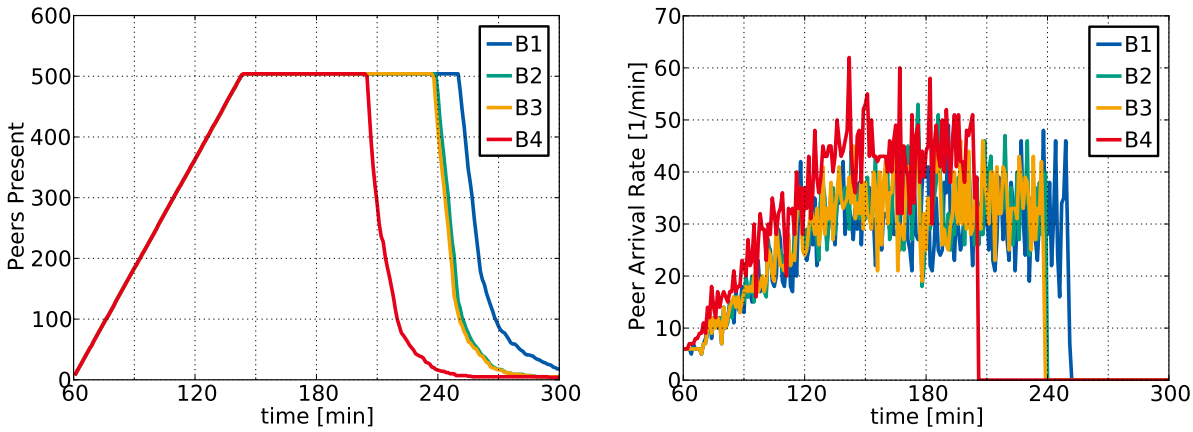
Workload B – Churn and Constant Peer Count

With this workload model, we aim at understanding the impact of churn frequency on the overlay performance. Initially, peers join the network at a constant arrival rate Λ_0. Instead of staying in the overlay for an unlimited time, they draw a random session length δ. Vu et al. [VGNL10] show through

measurements of the PPLive system that the probability p of a peer still being online after x minutes follows the exponential distribution

$$p = a \cdot e^{b \cdot x} \quad , \tag{7.10}$$

with a and b varying slightly depending on the characteristics of the monitored video channel. We use the values for a and b as determined by Vu et al.. As the measurement trace used in the study provides samples every ten minutes, the authors model the session lengths with a geometric series. The series returns integer multiples of ten minutes instead of continuous values. In the equation stated above this is neglected and the model of Vu et al. is used in its continuous variant. To prevent peers from leaving immediately, we introduce a minimum session length $\delta_{\min}$.



(a) Number of peers $\tilde{P}$

(b) Arrival rate Λ

Figure 7.1.: Number of concurrently online peers and arrival rates for workload model B over five hours simulation time.

As soon as the number of concurrently online peers $|\tilde{P}|$ reaches a threshold $|\tilde{P}|_{\max}$, a new peer joins only if a previously online peer leaves the overlay. Thus, the number of peers remains constant after the threshold is reached for the first time, allowing us to control the size of the scenario. Different configurations of this workload model are summarized in Table 7.4. For all variations we assume that peers join up to the threshold $|\tilde{P}|_{\max}$ with a rate of $\Lambda_0 = 12$, resulting in one peer joining every five seconds[1]. The maximum simulation duration is limited by T_s and by the number of peers $|P|$, as peers join the overlay only once. Thus, if all $|P|$ peers have been active, leaving peers are no longer replaced with new ones, which will ultimately lead to the overlay becoming depleted. The workload is depicted in Figure 7.1, with the workloads differing in the session lengths of peers. Variant B1 to B3 differ in the parameters for the exponential distribution. Workload B4 has a lower minimal session time, leading to higher churn rates. The number of concurrently online peers is fixed to 500, as mentioned before. Once this threshold is reached, the arrival rate is solely determined by the session lengths of peers that are already present in the system.

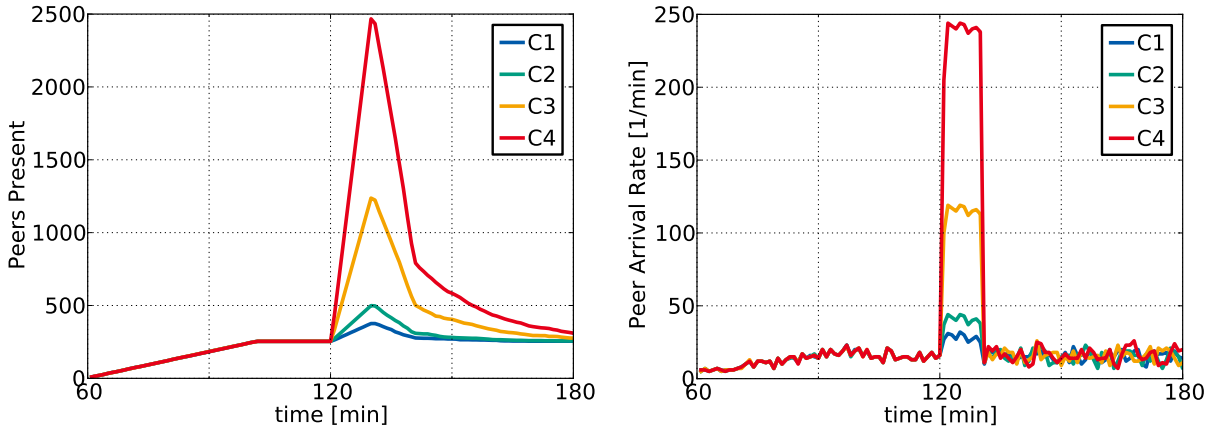
[1] The observed arrival rate is slightly higher, as peers that go offline due to their session length being expired are immediately replaced by a new peer.

Parameter	Model B1	Model B2	Model B3	Model B4
T_s	5 h	5 h	5 h	5 h
$\lvert\tilde{P}\rvert_{\max}$	500	500	500	500
a	0.6378	1.183	1.079	1.079
b	−0.05944	−0.09878	−0.09594	−0.09594
$\delta_{\min}$	10 min	10 min	10 min	2 min

Table 7.4.: Variations of the churn-based workload model B used in the evaluation. The values for a and b were extracted from [VGNL10].

Workload C – Flash Crowd

This model behaves just like workload B, in that peers join the overlay with an assigned session length until the threshold $|\tilde{P}|_{\max}$ of concurrently online users is reached. However, after time t_{fc} the number of online peers is increased linearly up to $m_{\text{fc}} \cdot |\tilde{P}|_{\max}$. This increase takes place during the *flash crowd interval* d_{fc} which is fixed to ten minutes. The flash crowd interval is fixed as we want to evaluate the system performance under varying arrival rates that can be the result of a flash crowd. Therefore, by fixing the interval length and by varying the target number of peers, we achieve different arrival rates. Peers that join during this interval are not replaced with new peers if their session length expires. Thus, the number of online peers will converge back to $|\tilde{P}|_{\max}$ as more and more sessions end. Finally, if the number of online nodes drops below $|\tilde{P}|_{\max}$, leaving nodes will again be replaced with new nodes.



(a) Number of peers $\tilde{P}$

(b) Arrival rate Λ

Figure 7.2.: Number of concurrently online peers and arrival rates for workload model C over five hours simulation time. The arrival rate as well as the number of online peers increases significantly during the flash crowd phase, starting after two hours.

The parameters used for workload C in this evaluation are summarized in Table 7.5. They are derived from the workload model B1 as described above and differ in the scaling factor m_{fc}, leading to different peer arrival rates during the flash crowd interval. In C3, for example, the number of peers during the flash crowd increases by a factor of five, leading to arrival rates of roughly two peers per second. The behavior of this model over an observation time of five hours is depicted in Figure 7.2. During the flash crowd interval starting after two hours, the arrival rate of peers is increased significantly, leading to a high number of concurrently online peers. After their session times expire, those peers leave the system and are not replaces with new peers.

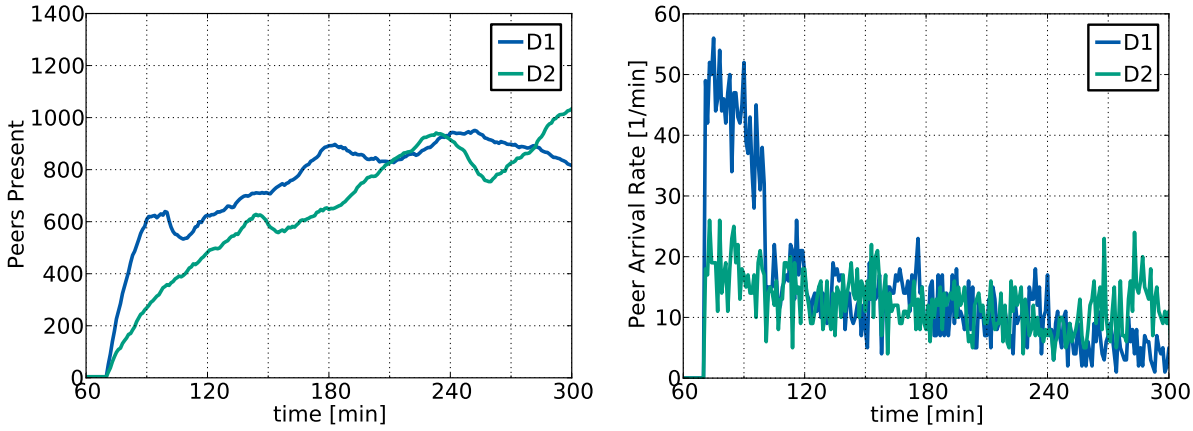
Thereby, the simulated scenario consists of a massive join of peers during the flash crowd phase and a massive leave shortly afterwards, as the session lengths are distributed exponentially. Both situations occur in real-life systems as well, for example on New Year's Eve [HLL+07] or after unforeseen events such as earthquakes [VGNL10], of course depending on the content the video stream is offering. Workload C captures these extreme, but realistic, scenarios in order to evaluate the benefits of a transition-enabled system in such a situation.

Parameter	Model C1	Model C2	Model C3	Model C4
T_s	3 h	3 h	3 h	3 h
$\lvert\tilde{P}\rvert_{\max}$	250	250	250	250
t_{fc}	2 h	2 h	2 h	2 h
m_{fc}	1.5	2	5	10
others	as in B1	as in B1	as in B1	as in B1

Table 7.5.: Variations of the flash crowd workload model C used in the evaluation.

Workload D – Measurement Trace

While the previously introduced workload models are synthetic and try to mimic real-world system characteristics, the following model is based on measurement traces of a deployed and popular P2P video streaming system. The traces used in this thesis are provided in the P2P trace archive [ZIPE10] and were conducted by Vu et al. [VGNL10] as result of a measurement study of the PPLive streaming system. There, the authors provide extensive insights into the user behavior and system dynamics of the PPLive system. The characteristics of the traces provided by Vu et al. closely resemble traces used in earlier studies by Hei et al. [HLL+07]. Those traces have also been the basis of a recent study by Hu et al. [HGL11]. The authors evaluate the benefits of developing a SVC-aware P2P streaming system in terms of incentive, fairness and efficiency. Using traces in our evaluation enables a much broader comparison with other systems and provides a good estimation of the system behavior under real-world conditions.



(a) Number of peers $\tilde{P}$ **(b)** Arrival rate Λ

Figure 7.3.: Number of concurrently online peers and arrival rates for workload model D over five hours simulation time. The arrival rates are based on measurement traces of the PPLive streaming overlay.

Figure 7.3 shows the number of concurrently online peers as well as the arrival rates for both traces. The arrival rate of trace D1 is much higher during the startup phase, leading to roughly 600 peers being online after one hour. However, after the startup phase both traces behave quite similar, as the arrival rate fluctuates around 15 peers per minute.

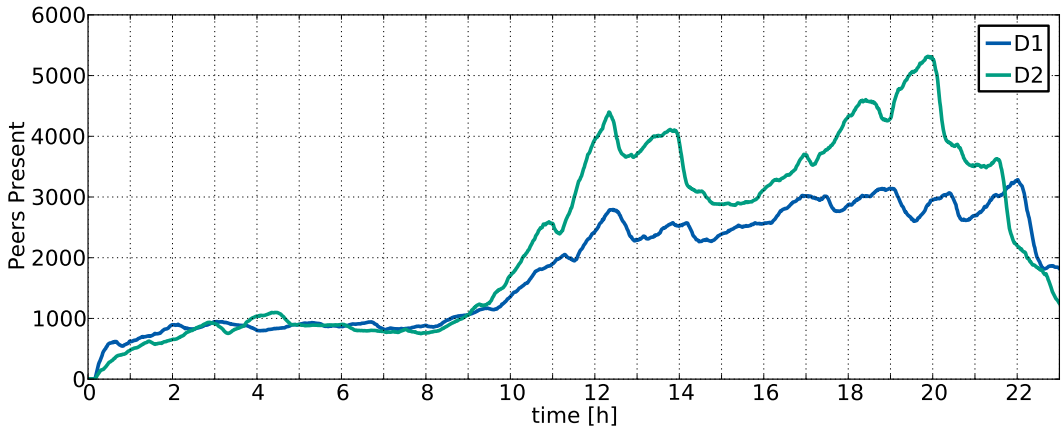


Figure 7.4.: Number of peers $\tilde{P}$ over a complete day, according to the data conducted by Vu et al. [VGNL10] in a measurement study of the PPLIVE streaming system. The diurnal usage pattern is clearly visible from the increasing number of peers at noon and during the evening.

Due to scalability issues with the simulator[2], we only evaluate a part of the trace and model other aspects using the aforementioned synthetic workloads. Figure 7.4 shows the number of concurrently online users for both traces over one day. The diurnal pattern in service usage becomes clearly visible from the peaks during noon and then again during the evening. The arrival rates as well as the overall number of peers being concurrently online during such a phase motivates the design of workload C, where flash crowds are simulated.

7.3.2 Video and Network Model

After a peer joined the overlay it has to select which layer of the SVC encoded video it wants to stream. According to the definition of the video model in Chapter 2, one layer of a video out of the set of layers L is identified by $l = \{d, t, q\} \in L$ with d being the spatial level (i.e. resolution), t being the temporal level (i.e. frame rate) and q being the quality level (i.e. SNR). Each peer is assigned an initial layer based on its bandwidth, which is hereafter referred to as l_{init}. The peer always tries to retrieve all blocks necessary to play the video at layer l_{init}. If a peer misses blocks of the next chunk, it will stall until the chunk at the requested layer is completely downloaded to the buffer.

For the evaluation we configure the video model with bit rates based on the bitstream used by Hu et al. [HGL11]. For simplicity and as we do not focus on quality adaptation, we use only one dimension of the SVC stream, in this case the temporal dimension. The resulting video properties are summarized in Table 7.6. Note, that the bit rate increases by roughly 200 kbit/s with each additional layer that is added upon the base layer. The investigation of the impact of SVC videos with more quality levels and layers is part of future work and is beyond the scope of this thesis.

[2] Simulations of the whole measurement trace require roughly 40 000 hosts, as currently host objects are created once on simulator startup. Future versions might include support for on-demand instance creation, which would enable such large-scale simulations.

Layer	Bit rate (kbit/s)	Frame rate (f/s)
$\{0,0,0\}$	303.8	3.75
$\{0,1,0\}$	503.0	7.5
$\{0,2,0\}$	705.3	15
$\{0,3,0\}$	905.8	30

Table 7.6.: Video model parameters used for the evaluation. Derived from [HGL11].

We assign each peer to one of the groups specified in Table 7.7. The group determines the peer's upload and download data rate as well as the layer of the video file he is interested in. Access link data rates are based on a study conducted by the Organisation for Economic Co-operation and Development (OECD) [OEC12]. The table furthermore shows the distribution of nodes to those groups according to the OECD study. The last column contains the initial layer l_{init} for each group.

Group	$\hat{R}_{\uparrow}$ (Mbit/s)	$\hat{R}_{\downarrow}$ (Mbit/s)	Percentage	l_{init}
DSL	2.25	15.3	56%	$\{0,1,0\}$
Cable	3.15	42.0	30%	$\{0,2,0\}$
FTTH/Ethernet	52.67	96.4	14%	$\{0,3,0\}$

Table 7.7.: Access link data rates and frequency of peers according to the OECD Broadband Report [OEC12].

We consider the access link bandwidth to be the main limiting factor of a peer's performance in the overlay. The time it takes a packet to be transmitted is therefore given by

$$d_{\text{tx}} = d_{\text{q}} + \frac{\text{size(packet)}}{R_{\uparrow}} \quad , \tag{7.11}$$

where d_{q} is the time a packet has to wait in the outgoing transmission queue. The packet arrives at the receiver after an additional latency d_0, which models the routing path delay. We chose d_0 once for each end-to-end connection out of a uniformly random distribution between 50 ms and 150 ms. This value is based on measurement data of the PingER project analyzed by Matthews et al. [MC00], where the RTT between hundreds of hosts distributed all over the world is measured.[3] The length of the outgoing transmission queue is solely determined by the sending rate of a peer: if a peer saturates its outgoing link, packet delays will grow accordingly. We limit the queue of a sender to 100 packets, after which new packets will be dropped immediately. Other than that, we do not consider any intermediate packet loss. If a packet fits into the outgoing queue, it will be sent eventually. Packet loss could be an issue in more resource-constrained networks, which is why it is considered future work.

In a real-world deployment, the overlay would have to deal with peers behind Network Address Translation (NAT) boxes. Peers behind a NAT box cannot be easily reached from the outside, which makes it difficult to utilize their uploading resources. Hei et al. [HLL+07] consider missing NAT traversal capabilities one of the reasons P2P streaming systems are still not widely deployed. However, as multiple

[3] Values in the cited article are rather outdated. The homepage of the project, `http://www-iepm.slac.stanford.edu/pinger/` [accessed 08/11/2012], provides up-to-date measurements. In this thesis, we assume peers are distributed across Europe. According to the most recent data provided on the website, peers across Europe experience an average RTT of 40 ms, making our assumed delays rather pessimistic. However, as PingER measures the RTT between hosts situated at universities, an average household will most probably experience higher latencies.

approaches towards NAT traversal in P2P exist, this issue is considered to be out of the scope of this thesis. Peers in the presented network model are able to directly communicate with each other, thereby eliminating the need for NAT traversal.

The simulation setup and the metrics as defined in the previous sections are used to evaluate the impact of the system parameters of TRANSIT on its performance, as presented in the following section.

7.4 Evaluation of System Parameters

In this first part of the evaluation we study the impact of some of the system parameters of TRANSIT on its performance in various scenarios. The evaluated parameters are summarized in Table 7.8, with their default value being underlined[4]. In the following, only one parameter is varied while the others remain fixed at their default value. Each simulation is repeated five times with different seeds for the random generator to gain statistical significance. If not otherwise stated, 95% confidence intervals are reported. Due to space constraints, not all evaluation results are presented in this chapter. Please consult Appendix B for additional evaluation results.

Parameter	Symbol	Variations	Default
ConnectionMaxIncoming	$\max\|C_{\leftarrow}\|$	8, <u>16</u>, 32, 64	16
ConnectionMaxOutgoing	$\max\|C_{\rightarrow}\|$	8, 16, <u>32</u>, 64	32
ConnectionMaxIncomingInactive	$\max\|C_{\leftarrow}\|_{\text{stale}}$	2, 4, <u>8</u>, 16	8
ConnectionInactivityTimeout	d_{kill}	15 s, <u>30 s</u>, 1 min, 5 min	30 s
ConnectionInactivityKillProbability	p_{kill}	<u>0</u>, 0.01, 0.1, 0.25, 0.5, 1	0
NeighborhoodExchangeInterval	I_{n}	30 s, 1 min, <u>2 min</u>, 4 min, 10 min	2 min
RequestBufferUrgentSize	B_{u}	5, <u>15</u>, 25, 35	15 chunks
BuffermapExchangeInterval	I_{b}	1 s, <u>2 s</u>, 4 s, 8 s	2 s

Table 7.8.: Overview of the evaluated system parameters. Underlined values mark fixed settings while another parameter is varied. The resulting default parameter settings are listed in the rightmost column.

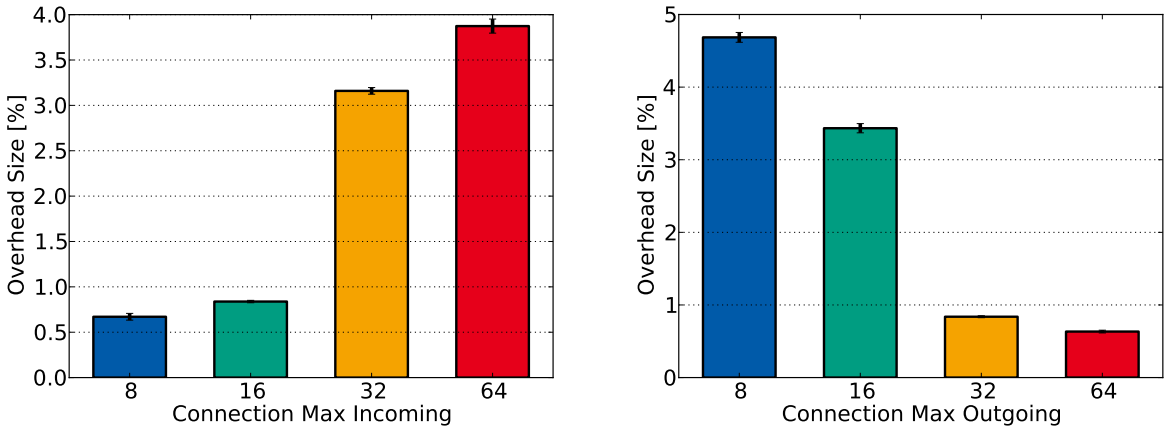
For the evaluation of the system parameters of TRANSIT we use the workload models D1 and B3 as defined in the previous section. These models are based on real-world measurement traces and therefore allow us to fine-tune the system with respect to *typical* real-world conditions. We do not want to configure TRANSIT based on, for example, a flash-crowd scenario, as later on the system has to be able to adapt to such a situation on its own. The result of this part of the evaluation should thus be considered the default configuration of TRANSIT.

7.4.1 Neighborhood Parameters

The neighborhood layer in TRANSIT is affected by different configuration parameters. The maximum number of incoming connections is limited to $\max|C_{\leftarrow}|$ and for outgoing connections to $\max|C_{\rightarrow}|$, respectively. However, there is a third parameter, $\max|C_{\leftarrow}|_{\text{stale}}$, that limits the number of concurrently open, inactive incoming connections. A connection is considered if it has not been used for requests or flows for at least d_{kill} seconds. Instead of keeping such connections open any longer, the peer establishes a new connection

[4] For easier reference, the names correspond to the identifiers used in TransitSettings.java

to replace them. For the evaluation, the aforementioned parameters are varied according to Table 7.8. The findings from these evaluations are presented below.



(a) Overhead O_s for $\max|C_{\rightarrow}| = 32$

(b) Overhead O_s for $\max|C_{\leftarrow}| = 16$

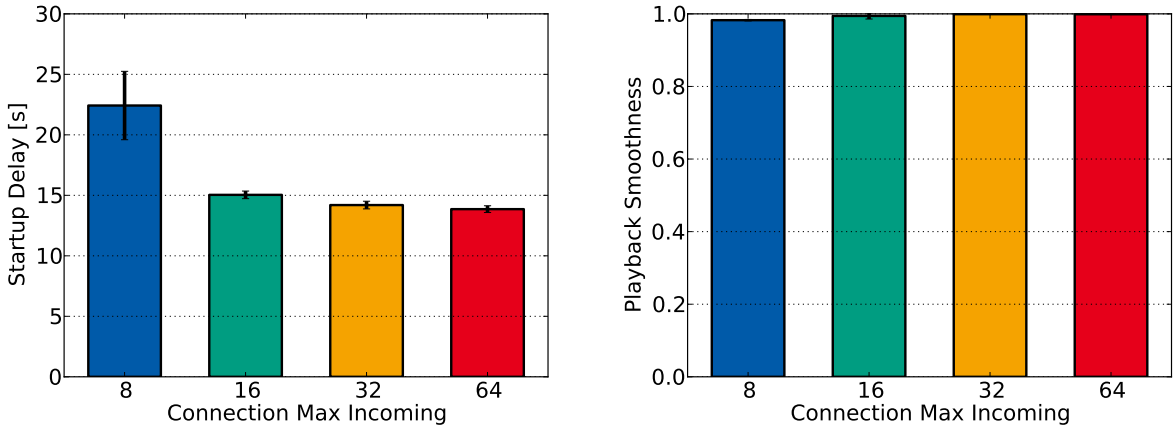
Figure 7.5.: Overhead in terms of data being transmitted, depending on the number of incoming and outgoing connections. The overhead increases significantly, if the number of outgoing connections is less than or equal to the number of incoming connections, as peers are having problems finding neighbors.

We begin by evaluating the impact of the maximum number of incoming and outgoing connections, $\max|C_{\leftarrow}|$ and $\max|C_{\rightarrow}|$, respectively. More important than the actual number of connections is the ratio of incoming versus outgoing connections, as indicated by Figure 7.5. Here, the resulting overhead depending on $\max|C_{\leftarrow}|$ and $\max|C_{\rightarrow}|$ is shown. As long, as the number of incoming connections stays at or below one half of the number of outgoing connections, the overhead stays well below 1 % of the total transmitted data. As soon as the number of incoming connections exceeds this threshold, the overhead increases to above 4 %. Low overhead indicates that peers were able to establish a useful neighborhood for the scheduler. The same effect is visible when varying the number of outgoing connections for fixed $\max|C_{\leftarrow}| = 16$. Now, as soon as the number of outgoing connections is larger than the number of incoming connections, the overhead decreases significantly.

These results are not very surprising, as peers try to maintain a well-connected neighborhood. If the number of outgoing connections is less than the number of incoming connections, peers are having problems finding enough neighbors, resulting in connection negotiations being canceled as the limit for $\max|C_{\rightarrow}|$ is already reached. This results in high connection frequencies and, thus, higher overhead.

The performance of the overlay is only slightly affected, as shown in Figure 7.6 for different values of $\max|C_{\leftarrow}|$. With an increasing number of incoming connections, peers experience shorter startup delays. This is due to the bursty transmission of the initial packets, which is more efficient when there are multiple senders. The startup delay saturates at roughly 12 seconds, as explained in Section 7.2.2. The playback smoothness is only slightly affected when $\max|C_{\leftarrow}| = 8$, but saturates at $\Psi \approx 1$ for $\max|C_{\leftarrow}| \geq 16$. Considering the overhead, as already presented in Figure 7.5, we choose $\max|C_{\leftarrow}| = 16$. The results for the number of outgoing connections are presented in Appendix B. They show similar characteristics, leading to a default value of $\max|C_{\rightarrow}| = 32$.

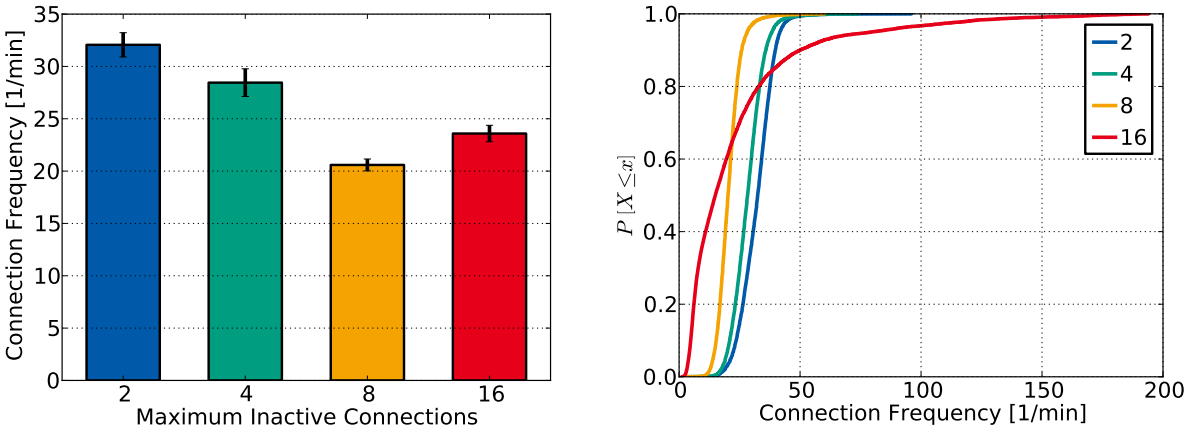
As already mentioned, the absolute numbers for $\max|C_{\leftarrow}|$ and $\max|C_{\rightarrow}|$ are not that important, as long as $\max|C_{\leftarrow}| \ll \max|C_{\rightarrow}|$ holds true. For TRANSIT, a ratio of $\max|C_{\leftarrow}| = 0.5 \cdot \max|C_{\rightarrow}|$ has proven to be a good choice. It just has to be ensured, that newly joining peers are able to establish $\max|C_{\leftarrow}|$ incoming connections as soon as possible.



(a) Startup delay d_{su} for $\max|C_{\rightarrow}| = 32$ **(b)** Playback smoothness Ψ for $\max|C_{\rightarrow}| = 32$

Figure 7.6.: Startup delay and playback smoothness for different values of $\max|C_{\leftarrow}|$. The number of outgoing connections is fixed to $\max|C_{\rightarrow}| = 32$. The startup delay decreases with more incoming connections, while the playback smoothness already reached a very high level and increases only slightly.

A connection that has not been used for the transmission of payload for the last d_{kill} seconds is considered being *inactive*. If more than $\max|C_{\leftarrow}|_{\text{stale}}$ incoming connections are inactive, those that have not been used for the longest time are closed, until the number of inactive connections again reaches $\max|C_{\leftarrow}|_{\text{stale}}$. Thereby, the neighborhood is changing frequently, even if a peer is currently able to request all packets needed for playback. By varying the timeout until a connection becomes inactive and by varying the limit of inactive connections, one can influence the frequency of neighborhood changes. This part of the evaluation studies the impact of both parameters on the performance of the streaming overlay.



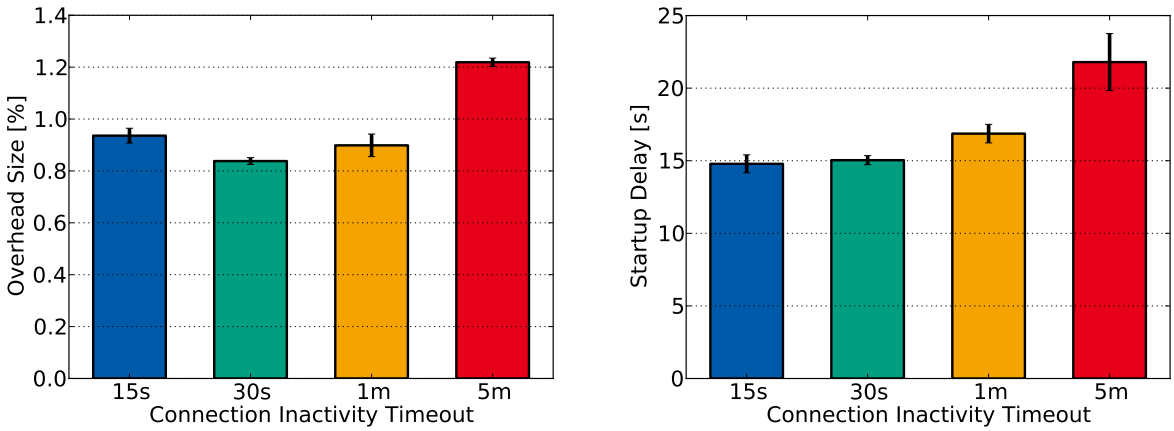
(a) Average connection frequency f_C **(b)** Distributions for different $\max|C_{\leftarrow}|_{\text{stale}}$

Figure 7.7.: Limiting the maximum number of inactive connections has direct impact on the connection frequency. If more inactive connections are kept open, the connection frequency decreases. However, if $\max|C_{\leftarrow}|_{\text{stale}} \approx \max|C_{\leftarrow}|$, peers occupy to many outgoing links of potential sources, thereby making it hard for new peers to establish connections. Thus, those peers experience very high connection frequencies as visible in the distribution for $\max|C_{\leftarrow}|_{\text{stale}} = 16$.

Figure 7.7 shows the connection frequency resulting from different configurations of $\max|C_{\leftarrow}|_{\text{stale}}$. For small values of $\max|C_{\leftarrow}|_{\text{stale}}$, the connection frequency increases, as expected. However, for $\max|C_{\leftarrow}|_{\text{stale}} = 16$, the average connection frequency again increases. Considering the distribution of av-

erage connection frequency over peers, one observes that about 10 % of the peers experience significantly higher connection frequencies of more than 50 connection attempts per minute. This is an indicator that these peers are not able to establish sufficient incoming connections to maintain fluid playback, leading to high frequencies of connection requests. As $\max|C_{\leftarrow}|_{\text{stale}} = \max|C_{\leftarrow}|$ for $\max|C_{\leftarrow}|_{\text{stale}} = 16$, peers do not close connections due to inactivity. Thereby, peers that joined early occupy connections that they never use for scheduling. If new peers join, they are not able to establish connections to a large subset of peers, as those already saturated their outgoing link with stale connections.

Frequently changing neighborhoods have shown to be beneficial to the overlay, as they increase the robustness against churn. The timeout, after which a connection is considered to be inactive and, thus, can be replaced with a new connection, is determined by d_{kill}. Figure 7.8 shows the impact of d_{kill} on the overhead and the startup delay. The playback smoothness remains largely unaffected by this parameter. We choose $d_{\text{kill}} = 30\,\text{s}$, as the trade-off between overhead and startup delay is best for this value.



(a) Overhead size O_s

(b) Startup delay d_{su}

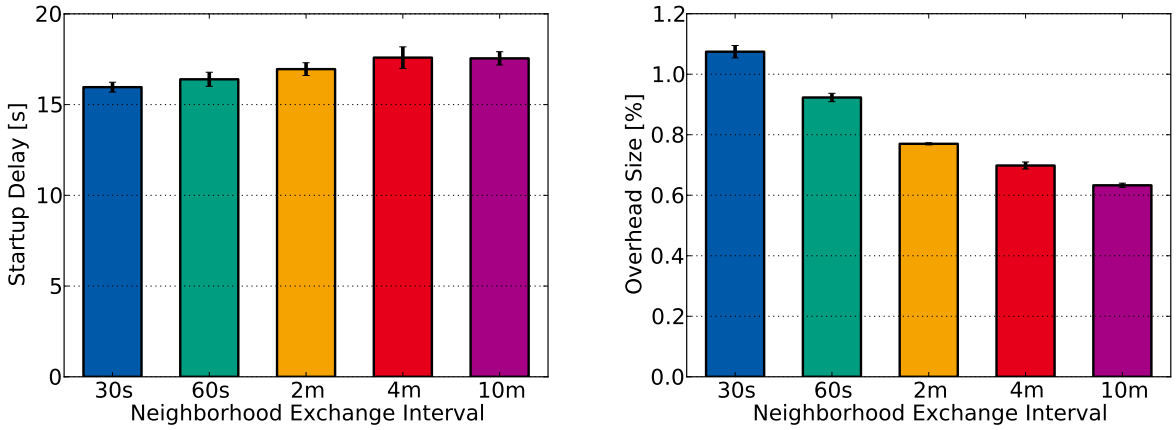
Figure 7.8.: Impact of the connection inactivity timeout d_{kill} on the startup delay and the overhead of the system. The startup delay increases with d_{kill}, as does the overhead. The best trade-off between startup delay and overhead is achieved for $d_{\text{kill}} = 30\,\text{s}$.

The neighborhood layer is reliant on the exchange of neighborhoods, in order to establish new connections and to replace broken ones. Peers gossip a subset of their current neighborhood periodically, as described in Chapter 5. The interval, after which a peer sends its current neighborhood, is defined by I_{n}. Figure 7.9 shows the resulting overhead for different values of I_{n}. As expected, the overhead decreases with larger intervals, as less messages are sent and inactive connections are less often replaced by new ones. However, this has an impact on the startup delay and the playback smoothness, which are slightly worse for longer intervals.

The default value of I_{n} is set to two minutes, as with $I_{\text{n}} = 2\,\text{min}$ a good trade-off between overhead and startup delay is achieved. Besides the configuration parameters that have direct impact on the neighborhood layer of TRANSIT, there are additional parameters used by the scheduling layer. The evaluation of those parameters is presented in the following section.

7.4.2 Requests and Buffer Maps

As presented in Chapter 5, the buffer is separated into three phases, with the *urgent phase* consisting of the packets that are closest to the peer's current playback position. Missing blocks in the urgent phase

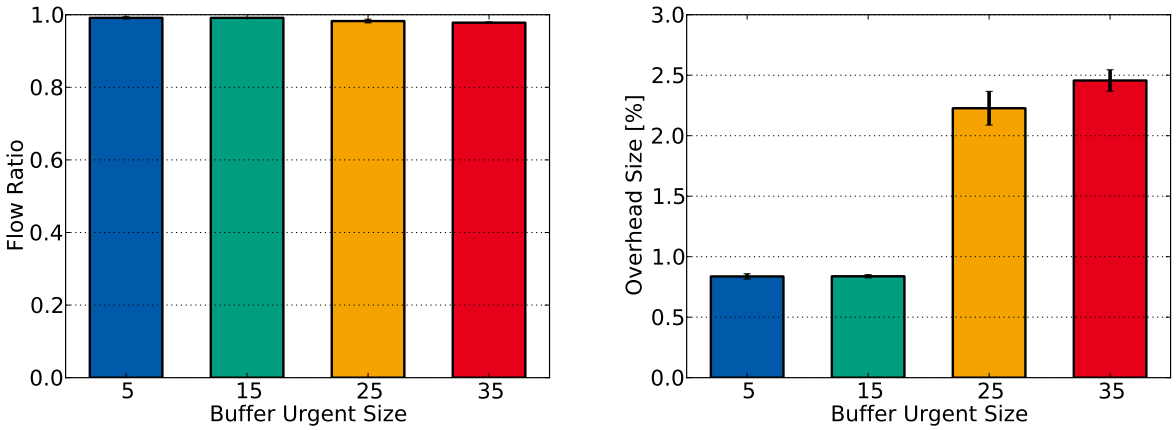


(a) Startup delay d_{su}

(b) Overhead size O_s

Figure 7.9.: Impact of the neighborhood exchange interval, I_{n}, on the startup delay of the overlay and the resulting overhead. Small values for I_{n} lead to higher overhead while at the same time providing better startup delays. A good trade-off is achieved for $I_{\mathrm{n}} = 2$ minutes.

of a buffer are requested immediately, in order to prevent the peer from stalling. Therefore, if the size of the urgent phase in terms of chunks, B_{u}, is too small, a peer experiences more stalls. If, however, the urgent phase is chosen too large, it includes blocks that are yet to be delivered via flows. Requesting those packets results in duplicates, which in turn leads to higher overhead for the peer. Figure 7.10 illustrates this problem, as the flow ratio for $B_{\mathrm{u}} > 15$ decreases slightly. The overhead increases at the same time, as more packets are duplicates. However, the impact is not too severe, as often the most recent packets are not yet advertised through buffer maps and therefore not requested by peers, even if B_{u} is large. Choosing $B_{\mathrm{u}} = 15$ provides good performance at reasonable overhead. For most of the playback time, peers are able to maintain incoming flows for all requested SVC layers, as the scenario is rather stable. This is clearly visible from the average flow ratio depicted in Figure 7.10, which stays well above 95 %. This means, that a peer is able to retrieve packets via flows for at least 95 % of its playback time.



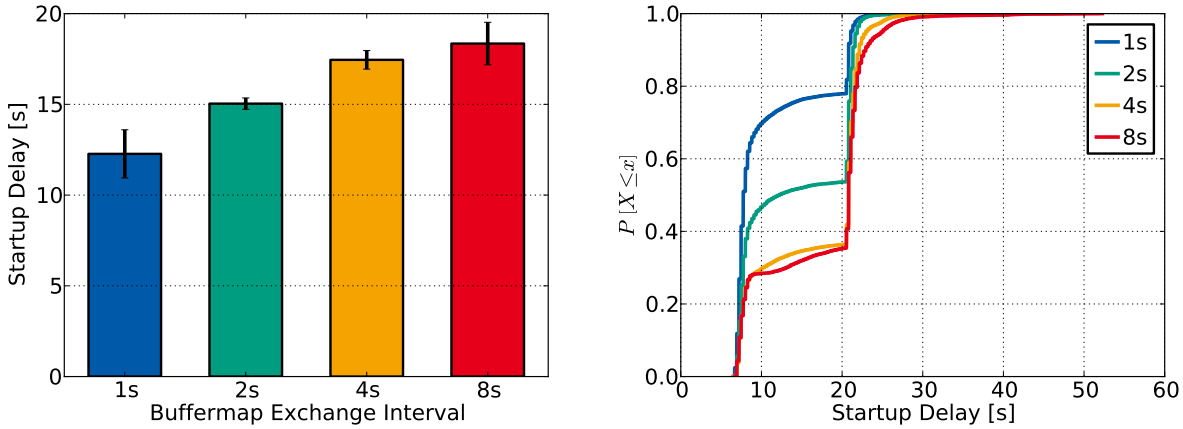
(a) Flow ratio R_f

(b) Overhead size O_s

Figure 7.10.: Impact of the urgent phase size of a peer's buffer. If the urgent phase is too large, packets are requested before their respective flow delivery, leading to duplicates and, thus, higher overhead.

The interval at which buffer maps are exchanged plays an important role for the performance of the request mechanism, which is mostly used during the startup phase and on flow failures during the trace-

driven workload. Therefore, the buffer map exchange interval, I_b, is varied and the evaluation results are presented in Figure 7.11.



(a) Average startup delay d_{su} **(b)** Distribution of startup delays d_{su}

Figure 7.11.: Impact of the buffer map exchange interval on the startup delay. With a faster exchange of buffer maps, more peers experience shorter startup delays.

As already mentioned, the trace-driven workload model is rather moderate considering churn rates, thus leading to nearly perfect playback smoothness as flows are long-lived. However, the startup delay is affected by the choice of I_b, with smaller exchange intervals leading to shorter startup delays for a larger fraction of peers. Considering the moderate increase in overhead due to smaller values of I_b, we choose two seconds as a default value for TRANSIT.

The evaluation of system parameters allows the configuration of TRANSIT with suitable settings for a real-world application scenario. In the following part of the evaluation, TRANSIT using these default settings is compared to a tree-based and two mesh-based configurations, and the impact of transitions on the performance of the system is studied.

7.5 Evaluation of Transitions

In this part of the evaluation, the impact of transitions on the system and application level metrics is studied. Therefore, TRANSIT with the configuration determined in the previous section is compared to its configuration as a pure mesh and a multi-tree system. TRANSIT as a testbed for different mechanisms was already introduced in Chapter 5. It supports two configurations as a pure mesh, with the configurations differing only in the scheduling strategy. In the simple mesh, those blocks that are closest to a peer's current playback position are requested, which results in an *earliest deadline first* scheduling strategy. With *newest-first* as scheduling strategy, peers additionally request the most recent blocks advertised by their neighbors.

In contrast to the pure pull-based content delivery, the multi-tree configuration of TRANSIT prefers push-based content delivery. However, as missing packets due to tree failures need to be retrieved, the multi-tree configuration still uses the request mechanism to some extend. Requests issued by a peer in the multi-tree configuration are only allowed on connections that are active parts of the tree. The motivation behind this configuration is to mimic the behavior of a multi-tree overlay with failure recovery: once a parent fails, it is replaced with a new one. Packets that got lost during this phase have to be requested from the new parent (or any other currently active parent). One difference compared to TRANSIT is the

limitation of flows to contain exactly one SVC layer. Thereby, we enforce a multi-tree topology, where each layer is delivered by another parent. However, there is no additional tree management involved, i.e. there is no active mechanism that tries to maintain balanced trees. This is due to the fact that the multi-tree configuration obviously relies on the same neighborhood layer as TRANSIT does. Thus, there is a mechanism in place that provides a peer with a fresh set of connections, if one of the current parents fails. In sum, the following configurations of TRANSIT are used throughout this section:

Transit The full, transition-enabled version of TRANSIT. The system is configured based on the evaluation results presented in the previous section. This configuration uses both, flows and requests. It furthermore supports seamless transitions between those mechanisms.

Tree This configuration of TRANSIT mimics the behavior of a multi-tree system by limiting the number of layers transmitted via a flow to one. Furthermore, requests are only allowed on active parents in the trees formed by the flows. Tree recovery is based on the neighborhood layer of TRANSIT, which provides new connections if a parent fails.

Mesh A mesh with *earliest deadline first* scheduling, where the blocks that are closest to a peer's current playback position are requested first. This configuration does not use the flow mechanism at all. The maximum number of incoming and outgoing connections is increased when compared to the pure TRANSIT system.

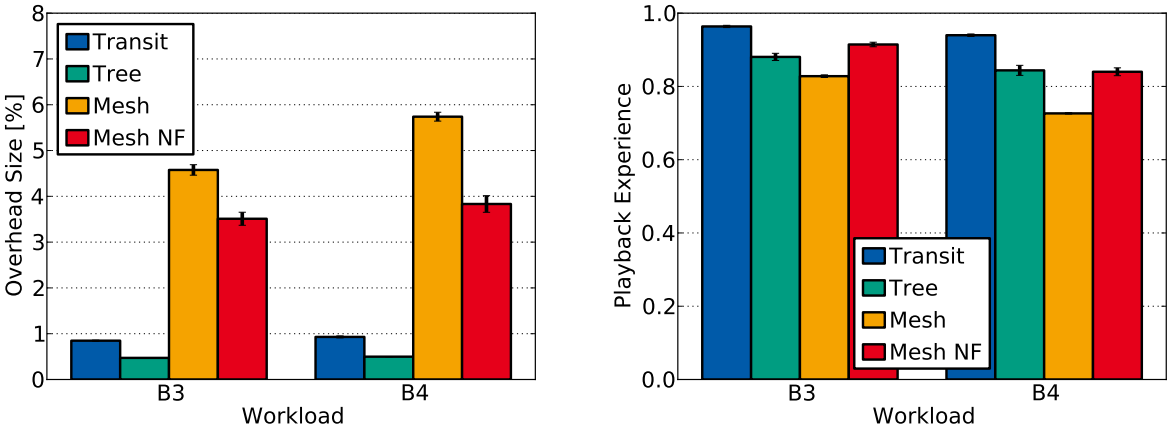
Mesh NF This configuration differs from the aforementioned mesh in that the scheduler applies a *newest-first* selection strategy. Peers do not only request the blocks that are closest to their current playback position, they furthermore request the most recent blocks advertised by their neighbors. Thereby, newly available blocks spread faster throughout the overlay.

In order to evaluate the impact of transitions on a streaming system's performance, we compare those systems under varying workloads. First, the systems' robustness towards churn is evaluated under workload B. Afterwards, the synthetic flash crowd model C is applied, as it provides both, high join rates and high leave rates. These situations are especially interesting for a transition of scheduling mechanisms. The system performance under a real-world load pattern is evaluated with the trace-based workload model D. As already mentioned in the previous section, all simulations are repeated five times with different seeds for the random generator. Evaluation results are reported with 95 % confidence intervals, if not otherwise stated.

7.5.1 Behavior under Churn

The system behavior under churn is evaluated with workload B. Due to space constraints, only the results for B3 and B4 are presented in this section as these workloads show the highest difference in churn rates. Figure 7.12 shows the average achieved playback experience as well as the overhead in terms of data being sent. TRANSIT achieves a better playback experience than the other systems. The mesh with *newest-first* scheduling performs roughly 5 % better than the tree for moderate churn, but as the churn rate increases, the mesh's performance decreases faster than the tree's performance, leading to a nearly identical playback experience of $\approx 85\,\%$. Nevertheless, all systems are affected by the increasing churn rate, as the playback experience drops slightly. Considering the overhead, the tree-based overlay and TRANSIT clearly outperform both mesh-based solutions. While TRANSIT stays well below 1 %, the overhead of the *newest-first* mesh is approximately four times higher. The mesh without *newest-first* scheduling performs even worse, with close to 6 % overhead during workload B4. This is an expected result, as the pull-based mechanism of the mesh requires at least one additional message per request. The flows

in TRANSIT and the tree-based overlay require only the overhead of the flow negotiation, as packets are pushed once a flows is negotiated successfully. The overhead of all systems increases with higher churn, but the impact is especially high for the pure mesh system. Here, more peers experience stalls due to packets not being available on any of their neighbors. This is due to the fact, that there is no dedicated mechanism to quickly distribute the most recent packets throughout the overlay. This effect is described in detail in the following section, when the flash crowd workload is considered.

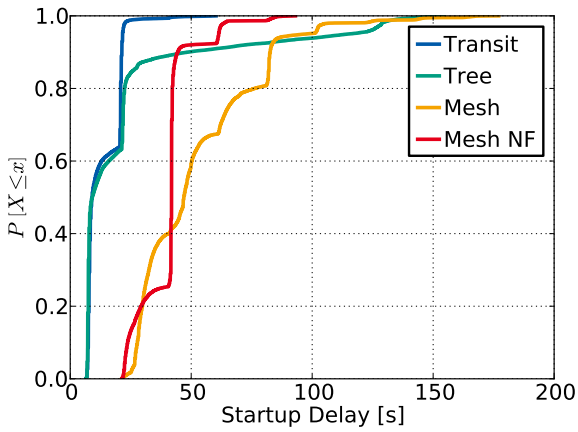


(a) Overhead size O_s **(b)** Average playback experience Υ

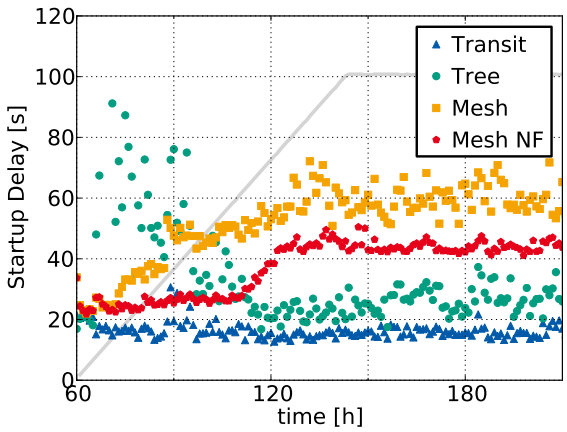
Figure 7.12.: Arrival rates for workload models B3 and B4 and the achieved average playback experience for the evaluated systems.

Figure 7.13 shows the distribution of the startup delays over peers for workload model B3. The steps that show up in the distribution are caused by two effects. First, as described in the simulation setup, we run the simulations with three different groups of peers. The groups do not only differ in the access link data rate, they also differ in the layers that are requested by the respective peer. Peers requesting higher layers than others might have difficulties retrieving the blocks of the highest layer. Those blocks are only available on the source and potentially on other peers of the same group, making them scarcer than the base layer packets, for example. Therefore, peers requesting a higher quality SVC layer tend to experience higher startup delays. With three groups of peers, one should thus observe three steps in the distribution. However, at least for the mesh-based solution there are even more steps. The additional steps in the distribution function of the mesh-based systems are caused by the slow dissemination of new packets in those systems.

As each peer requests the most recent packet after he joined, the startup delay is largely dependent on the time it takes the overlay to disseminate the most recent packets created by the source. For the tree-based approach as well as for TRANSIT, a fast dissemination is achieved through push-based scheduling of new packets. The mesh with *newest-first* scheduling is able to achieve similar results by always requesting those most recent packets, regardless of the current playback position. The mesh without such a mechanism showed to experience significant startup delays for a large fraction of peers, especially those joining late throughout the streaming process. This is clearly visible from the progression of startup delay over time, as shown in Figure 7.13(b). The more hops it takes a newly created block, the higher the delay – the curve for the mesh-based overlays follows the gray line indicating the workload progress in the number of online peers, as introduced in Section 7.3.1. The tree, in contrast, profits from more peers in the overlay. With more potential neighbors, the likelihood of being able to connect to all subtrees needed for the requested SVC layer increases. This provides an explanation for the roughly



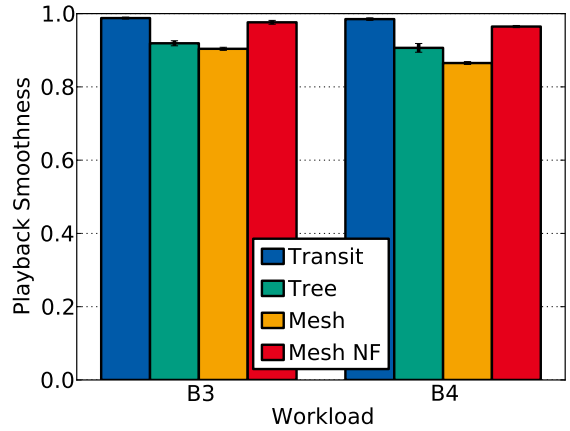
(a) Startup delay distribution



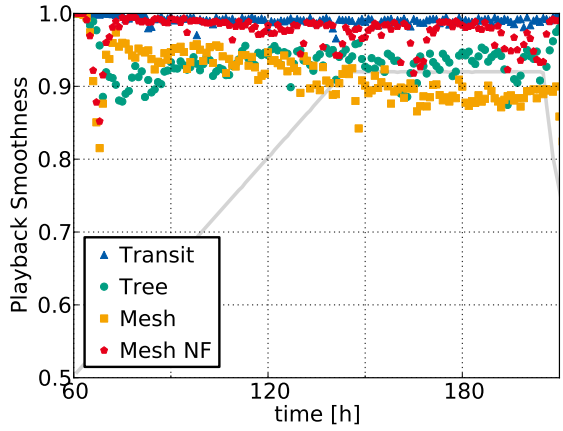
(b) Startup delay over time

Figure 7.13.: Distribution of startup delays as well as the progression over time for workload B3. Steps in the distribution function are caused by the different peer groups and their requested SVC layers, as higher layers lower the probability of neighbors being able to provide requested blocks and flows.

10% of peers that are experiencing a very high startup delay compared to the other participants of the tree-based overlay. As clearly visible from the progression over time, the startup delay of those peers that join the system early varies significantly and can become quite high, depending on the set of initial neighbors and their current SVC layer. A possible extension of the tree-based overlay should take these findings into account by providing a more sophisticated selection algorithm for the initial neighbor set.



(a) Playback smoothness Ψ



(b) Playback smoothness over time for workload B4

Figure 7.14.: Comparison of the playback smoothness for both workloads and the progression over time for workload B4, showing a decrease over time in the mesh-based overlay that does not provide a mechanism to quickly disseminate the most recent packets throughout the overlay.

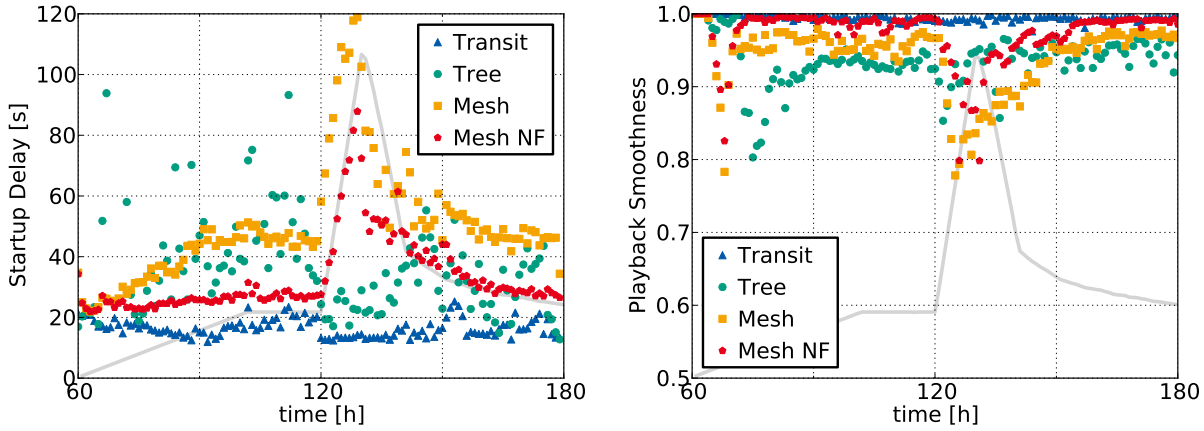
Considering the playback smoothness, shown in Figure 7.14, both TRANSIT and the *rarest-first* mesh perform well even under higher churn rates. Overall, the increasing chunk rate has only little effect on the playback smoothness. Only the mesh that does not provide a scheduling strategy for the most recent packets suffers from higher churn rates. For this system, the playback smoothness decreases over time. This is due to the fact that peers only request the packets close to their playback position. Thereby, the overall difference in the current playback position of peers increases with every stall, and at some time newly joined peers are only able to retrieve their required packets from the streaming source. The

streaming source, however, does not have enough capacity to provide packets for all those nodes, leading to even more stalls. The other systems provide rather constant playback smoothness over time, once again showing the importance of fast dissemination of new packets in a live streaming system.

One extreme case of churn in a video streaming overlay is the occurrence of a flash crowd, where the number of users increases significantly over a short period of time. The system behavior in such a scenario is presented in the following section.

7.5.2 Behavior in a Flash Crowd Scenario

As introduced in Section 7.3.1, workload C mimics flash crowds with different arrival rates, ranging to up to 250 peers per minute, leading to a total of 2500 peers being online concurrently. Right after the flash crowd interval, peers start to leave the system until the initial number of peers is reached again. Thereby, the workload requires the systems to handle the massive arrival of peers as well as a massive departure shortly afterwards. Figure 7.15 shows the startup delay and playback smoothness at different times throughout the simulation. Each depicted data point is an aggregation of all values collected over one minute. The startup delay is plotted in the interval in which the respective peer joined the overlay, while the playback smoothness is averaged in each interval over all currently online peers. The number of currently online peers is again indicated by the gray line, for exact values please consider Section 7.3.1. The flash crowd starts after two hours of simulated time, leading to a rapid and significant increase in online peers.

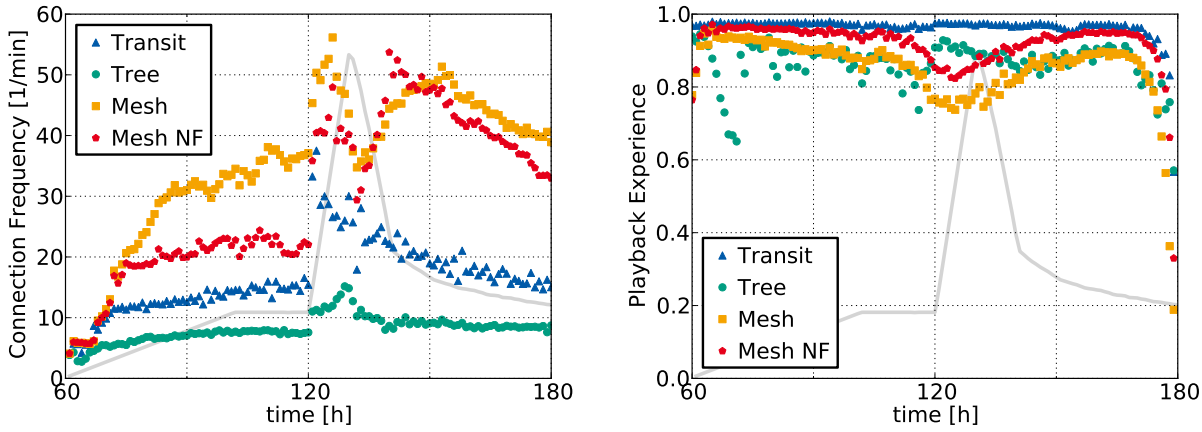


(a) Startup delay d_{su} **(b)** Playback smoothness Ψ

Figure 7.15.: System performance in the flash crowd scenario C3. The gray line indicates the number of currently online peers. During the flash crowd scenario, peers in the mesh-based systems experience higher startup delays than in the tree-based system, due to the most recent packets not yet being available on neighbors.

The high arrival rate of new peers during the flash crowd interval has severe impact on the startup delay in both mesh configurations. This is due to the fact that the larger the mesh, the more difficult it becomes to distribute new packets from the source throughout the overlay. Therefore, with each intermediate peer, the overall delay increases. As new peers start their playback at the most recent chunk in the system, they have to wait until those new packets reach one of their neighbors. While the *newest-first* approach helps in reducing this effect, it is still clearly outperformed by TRANSIT and its fast content dissemination through flows. The startup delay as well as the playback smoothness in the multi-tree system shows high

variation before and after the flash crowd interval, i.e. during a time where peers arrive at low rates. During the flash crowd interval, the multi-tree system shows rather stable values and outperforms both mesh-based approaches. This seems counterintuitive at first, given that mesh-based systems are generally considered to be more robust towards churn than tree-based systems [MRG07]. However, this effect is due to two important properties of a live streaming overlay. First, all peers are interested in roughly the same chunks of the video stream at the same time. As already mentioned, the mesh suffers due to the increased number of requests needed to distribute a new packet, even if *newest-first* is used as scheduling strategy. The multi-tree is able to distribute new packets very fast at least to a subset of all peers, as long as parts of the tree remain intact. Second, the chance of retrieving a missing block increases with the number of neighbors. Each time a peer in the multi-tree overlay leaves the system, missing flows on its children have to be re-established and missing blocks have to be requested from neighbors. During the flash crowd, the performance of the request mechanism of the multi-tree overlay increases, as the connection frequency and, thereby the number of potential sources for missing packets, increase as well. This is observable in Figure 7.16, where the connection frequency of all systems increases during the flash crowd phase.



(a) Connection frequency f_C

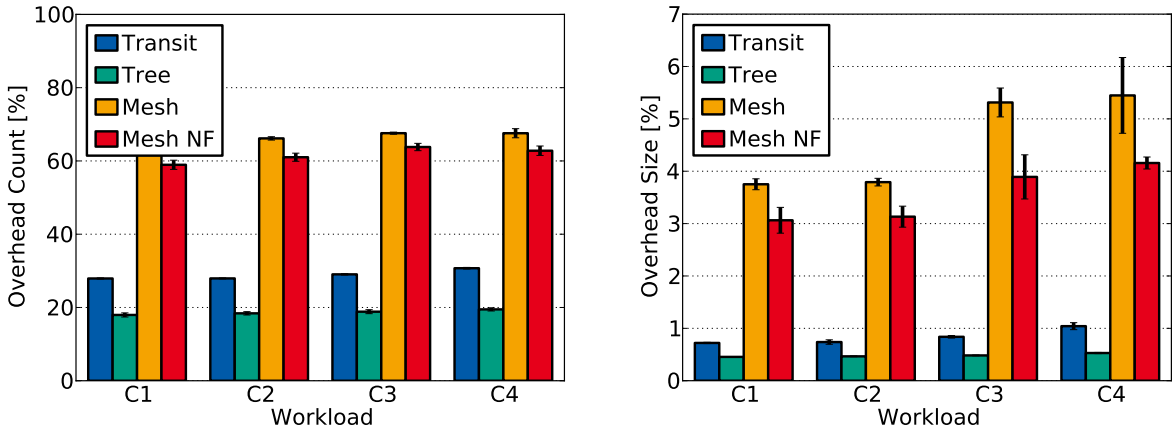
(b) Playback experience Υ

Figure 7.16.: Connection frequency over time, with an increase during the flash crowd phase. Higher connection frequencies benefit the performance of the tree, and the playback experience for peers that joined during this phase is higher than in the mesh-based system.

One could argue that the mesh should perform better as well, given that it consists solely of a request mechanism that should profit from the increased connection frequency. Nevertheless, compared to the multi-tree overlay, the mesh is not able to distribute new packets fast enough to a sufficiently large fraction of peers to support the high number of new arrivals. This results in higher startup delays as well as a significantly decreased playback smoothness of both mesh approaches during the flash crowd phase.

TRANSIT is able to maintain both high playback smoothness and low startup delay, even during the flash crowd phase. Figure 7.16 shows that the connection frequency of TRANSIT follows the pattern of the multi-tree overlay, with a much higher increase during the massive join phase. Thus, it is able to maintain ideal properties for the request mechanism: a high number of neighbors and frequent changes of those neighbors. At the same time, the request mechanism benefits from the flow mechanism used in TRANSIT, as new packets are distributed to a large fraction of the overlay very fast. This becomes even clearer when combining playback smoothness and startup delay of a peer over its full session time into a single metric, termed *playback experience*, as shown in Figure 7.16(b). Each marker in the plot shows

the aggregated playback experience for all peers that joined during the respective one minute interval. Therefore, a marker during the flash crowd is the average over a large number of peers that joined during this phase, whereas the markers before and after the flash crowd phase stand for less peers. As shown in Figure 7.16, the mesh with *newest-first* request strategy exhibits a very high and, more important, *consistent* playback experience for peers that joined before and after the flash crowd. The tree does not perform much worse, but its performance is far more dependent on the initial neighborhood selection and the failures of other peers, leading to high variations. However, during the flash crowd phase, the conditions for the tree benefit the playback smoothness and the startup delay, resulting in superior – and consistent – performance for peers that join during this phase. TRANSIT combines the advantages of both mechanisms and achieves consistently high performance, regardless of the time a peer joins the system. The sharp decrease in playback experience towards the end of the simulation is due to the fact that the peers left the overlay after three hours. Peers that join during this last phase of the simulation experience a startup delay as illustrated in Figure 7.15, but their session time is much shorter, resulting in shorter playback times. As the playback experience combines startup delay and playback phase weighted with their respective lengths, the startup delay has high impact on the calculation on this peers. From the results for the individual metrics, as presented in Figure 7.15, it becomes clear that those peers experience the same high performance as the others do.



(a) Overhead count $O_{\#}$

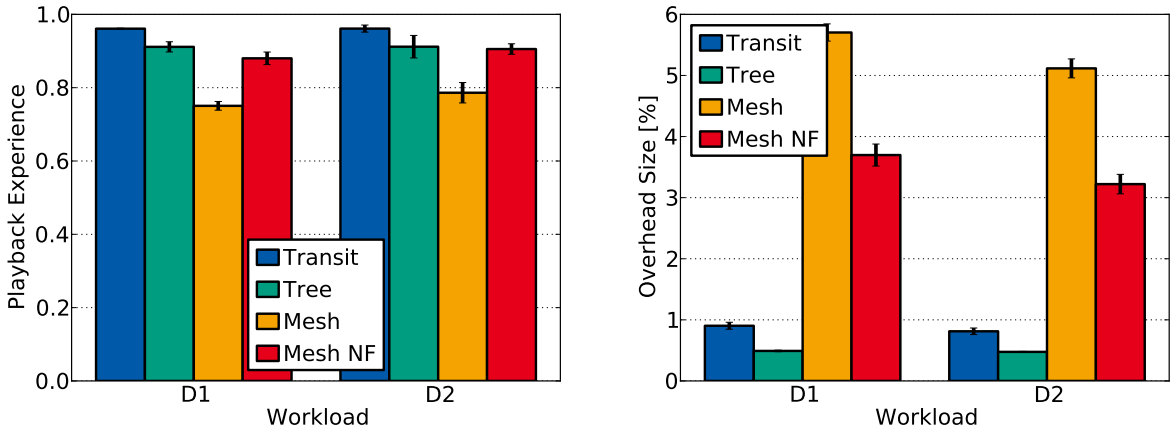
(b) Overhead size O_s

Figure 7.17.: Average per-peer overhead of the evaluated systems, in terms of messages being sent and in terms of data being transferred. Mesh-based systems need to request packets and thereby show significantly higher overhead in terms of messages being sent.

Often, additional robustness comes at the cost of increased overhead. Figure 7.17 shows the overhead in terms of messages being sent as well as the overhead in terms of actual bytes being transmitted on average as a fraction of the total data being sent. Both mesh-based systems show significantly higher overhead due to the pull-based packet scheduling. The overhead of all systems increases slightly with the intenseness of the flash crowd, indicating additional control traffic being exchanged due to connection maintenance. When considering the overhead in terms of actual data being transmitted, this effect becomes far more significant for the mesh-based systems. Here, overhead accounts for roughly 4% of all transmitted data, whereas it stays below one percent for TRANSIT and the tree-based overlay. Still, with only a fourth of the overhead introduced by the mesh-based overlay, TRANSIT is able to provide superior playback performance.

7.5.3 Behavior in a Real World Scenario

To simulate the systems under real world conditions, we use the trace-based workload model D, as described above. The simulations run for five hours, reaching up to 1000 simultaneously active peers. However, the churn rates in these traces are lower than the ones achieved with the synthetic workload models above. Therefore, the performance of the overlays does not change significantly over time, as the conditions are rather stable. Figure 7.18 shows the aggregated playback experience for both traces and all systems, as well as the overhead that is necessary to reach this performance. TRANSIT reaches the best performance of more than 95 % at an overhead of below 1 % of the total data being sent. The multi-tree overlay performs comparably well, and requires even less data overhead than TRANSIT. Considering the mesh-based overlays, the *newest-first* selection strategy exhibits superior performance over the *earliest-deadline-first* strategy, resulting in an increase of roughly 15 % considering the playback experience. At the same time, due to the more efficient scheduling process, connections are not changed as frequently, leading to a decrease in data overhead.

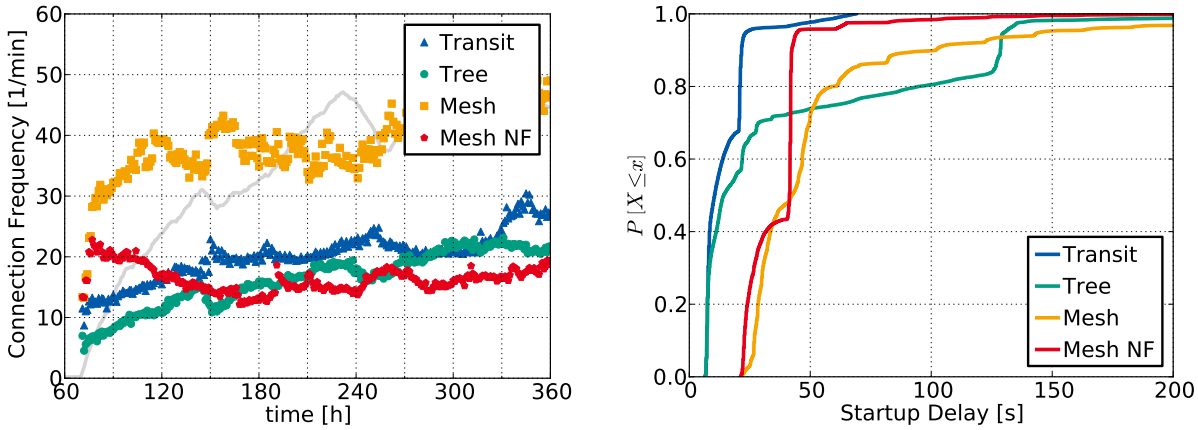


(a) Playback experience Υ

(b) Overhead size O_s

Figure 7.18.: The playback experience for the trace-based workload model does not differ significantly between the tree and the mesh with *newest-first* strategy. The simple mesh is not able to efficiently disseminate the most recent packets throughout the overlay, leading to a decreased playback experience. TRANSIT and the tree-based overlay exhibit low overhead compared to both mesh-based solution. Still, the *newest-first* mesh causes less overhead than the simple mesh.

The connection frequency over time is shown in Figure 7.19, again showing a significantly lower connection frequency for the *newest-first* mesh when compared to the other mesh configuration. The gray line, again, indicates the number of concurrently active peers in the overlay due to workload D. The connection frequency for the tree-based system follows this line in that the frequency increases with the number of online peers. The average connection frequency of TRANSIT rises slightly during phases where peers leave the overlay. This is an indicator of the efforts of the neighborhood layer, trying to maintain a set of connections for the negotiation of flows and requests. Due to the *newest-first* selection strategy, the respective mesh overlay profits from nearly every connection. Connections are not closed as long as they are used for the transmission of payload, leading to low connection frequencies. In contrast to that, the pure mesh changes neighborhoods very frequently, as especially peers that just joined the overlay have very little chance of finding a neighbor that can provide the newest blocks. Once again, the importance of the packet scheduling strategy in a live streaming overlay becomes clear.



(a) Connection frequency f_C over time

(b) Distribution of startup delays d_{su}

Figure 7.19.: The connection frequency over time for the trace-based workload model D2, as well as the distribution of startup delays over all peers. The tree-based solution and TRANSIT provide low startup delays for most of the peers. The tree has problems finding suitable contacts for peers that request the video at a higher layer, leading to a significant increase in startup delay for some of those peers. The startup delay in the mesh-based solutions depends largely on the availability of new packets and, thus, the number of hops a packet had to take from the source to the newly joined peer.

The distribution of startup delays as shown in Figure 7.19 shows the advantage of TRANSIT over the mesh-based solutions, as TRANSIT is able to achieve a much lower startup delay. The tree performs comparably well, but has problems with a fraction of the peers that request the video at a higher SVC quality layer. Those peers need significant amounts of time to find subtrees for each of their requested layers, leading to high startup delays.

In sum, the transition-enabled system exhibits superior performance over the mesh-based and the tree-based overlay. Especially in highly dynamic environments, for example during a flash-crowd, TRANSIT is able to maintain high playback smoothness and low startup delay, resulting in a high playback experience. The combination and seamless adaptation of the different mechanisms in TRANSIT results in a system that is able to operate under a broad range of environmental conditions. Depending on the conditions, the combination of mechanisms is altered in a fine-granular way to achieve consistently high performance. The evaluation results strongly motivate transitions as a built-in feature of future streaming systems. Furthermore, they qualify the assertion that mesh-based streaming systems are superior to tree-based systems in highly dynamic environments. As shown above, the most important design requirement for a live streaming system is the fast dissemination of the newest packets, be it by means of push-based content delivery or by specialized request strategies.

8 Conclusion and Future Work

Video streaming is one of the fastest growing applications in today's Internet. Due to the bandwidth requirements arising from streaming high quality video to thousands of users, most large-scale services utilize a Content Distribution Network (CDN). This, however, leads to significant operational cost for the service provider. Letting users contribute their resources to the service by deploying a P2P system is a promising alternative. Thus, in recent years, a multitude of streaming systems have been proposed, each under specific assumptions regarding the operational conditions. In Chapter 3 we showed that these assumptions influence the system design, with some mechanisms being more efficient under certain conditions than others. However, the conditions during a streaming session change significantly, as the number of users follows a diurnal pattern. Based on the content of the channel, the time of day, and the occurrence of events in the real world, the number of users can change significantly over short periods of time. These characteristics pose major challenges for a real-world streaming system.

Based on the idea presented in [RH12], we propose the concept of transitions in a P2P video streaming system to address these challenges. A transition enables seamless switching between mechanisms, for example from pull-based packet delivery in a mesh towards push-based packet delivery in a tree. Thereby, if the environmental conditions change, the system switches to a mechanism that is better suited for the new conditions and optimizes the video delivery. As a first system in this direction, the design and implementation of TRANSIT, a transition-enabled streaming system, is presented in this thesis. A layered architecture and well-defined interfaces and protocols enable multiple mechanisms on different peers to communicate with each other. To allow any combination of push- and pull-based packet scheduling, the concept of flows and requests is introduced. Flows, on the one hand, enable the fast, push-based dissemination of the most recent packets created by the source. This fast dissemination of new and, thus, rare packets in a P2P streaming overlay is crucial for the performance of the system, as shown during the evaluation. Requests, on the other hand, allow peers to request missing packets from their neighbors, based on the exchange of buffer maps. Thereby, strategies such as *earliest-deadline-first* and *newest-first* can be realized. The decoupling of those mechanisms allows the configuration of TRANSIT as a pure mesh-based overlay and a pure multi-tree overlay, thereby allowing an in-depth evaluation of the impact of transitions between mechanisms.

In all evaluated scenarios, TRANSIT outperforms pure tree- and mesh-based configurations in terms of playback continuity and startup delay. Peers in TRANSIT experience consistent playback performance even during flash crowds and scenarios with extreme churn. The evaluation results clearly show the benefits of a transition-enabled system over a single mechanism. In the following section, open research questions and thoughts on the future of transition-enabled streaming systems are presented.

8.1 Future Work on Transit as a Streaming System

TRANSIT is a modular streaming system, allowing the systematic evaluation and comparison of different mechanisms for scheduling and neighborhood selection. In terms of neighborhood selection, adaptive strategies such as the one presented by Lobb et al. [LdSL+09] promise better performance under heterogeneous conditions. It remains unclear, if such an algorithm can outperform basic random selection

strategies. As already mentioned in Chapter 5, Traverso et al. [TAB+12] recently conducted large-scale evaluations on a real-world streaming application, studying the impact of different neighborhood selection strategies. In their studies, a simple strategy based on the Round Trip Time (RTT) leads to the the best performance. However, in the context of transitions, other techniques might expose even better results for a limited set of conditions. Therefore, the results presented by Traverso et al. are to be carefully re-examined for transition-enabled systems, where the system is not limited to one mechanism. The performance of the scheduling algorithm is closely tied to that of the neighborhood selection, in that the neighborhood layer has to provide *useful* connections. It is up to the scheduler to assign flows and requests to those connections. The way this is done has a direct impact on the playback performance. Currently, the assignment is solely based on the RTT and the advertised load of the connection. However, future work should investigate the impact of more sophisticated assignment strategies on the overlay performance, especially if fairness is to be considered. In its current state, the system tries to maximize the overall performance, thereby distributing load across peers in an unfair manner. More advanced strategies could, for example, balance the workload more equally across peers. Especially for mobile devices, this load balancing mechanism might take the energy consumption of a device into account – we do not want the battery to drain immediately, even if it would be fair considering only the workload. TRANSIT, with its modular design and sophisticated evaluation framework motivates further research into this direction.

TRANSIT in its current state has been evaluated against its configurations as a pure tree and a pure mesh. Future work could include the implementation of other streaming systems for the simulator PeerfactSim.KOM. Especially the performance of hybrid streaming systems presented in the related work would be of interest, as those systems are designed to operate under a broad range of conditions. The workload models and metrics presented in this thesis allow a systematic comparison between those systems and TRANSIT, especially considering their reaction to a rapid change of conditions.

Security and Incentives

Not focus of this work, but important for every P2P streaming system are means to protect the system against attacks and misbehaving peers. In TRANSIT, peers are trusted entities, i.e. it is assumed that every peer behaves well. However, in real life scenarios, peers might intentionally manipulate packets in order to corrupt the stream. How to detect such peers and how to defend against this kind of attack is a topic of current research. Vieira et al. [VCA09] present a decentralized reputation mechanism to detect and isolate such peers. According to the authors, the overhead is small compared to the retransmission of data needed in case of a malicious peer intercepting the stream. Therefore, future work on TRANSIT could include the addition of a reputation mechanism such as the one proposed by Vieira et al..

Another common problem in P2P systems are *free riders*, i.e. peers that do not contribute any resources. In the context of live streaming, such peers might not forward blocks and they might advertise only empty buffer maps. In order to tackle the problem of free riding, an incentive mechanism to reward proper behavior is required. A popular mechanism to discourage free riders is *tit-for-tat*, as introduced for BITTORRENT [Coh03]. With *tit-for-tat*, peers experience higher QoS if they contribute more resources to the system. However, in live streaming, peers often do not have any content that is of interest for others, as interest is centered around a peer's current playback position. Therefore, once a peer experiences playback stalls, its playback position is behind most other peers' position. Thus, the peer is not able to provide any packets that are of interest for those peers in a *tit-for-tat* scenario. The design of an incentive mechanism for P2P live video streaming remains an open problem, according to Liu et al. [LRL08].

Nevertheless, Pianese et al. [PPK07] present a streaming system that uses a local pairwise incentive mechanism bearing similarities to *tit-for-tat*. The authors claim that the system efficiently rewards peers that perform well with a higher quality of service. According to the authors, this even leads to the starvation of peers that cannot contribute resources, finally forcing them to reconnect to the network. With SVC encoded content, an incentive mechanism could, for example, be centered around the video quality that is delivered to a peer. Piatek et al. [PKV+10] propose a system that provides better quality of service in terms of higher quality SVC layers to peers that contribute more of their resources. This idea is further taken on by Hu et al. [HGL11], with the authors proposing the use of SVC as a mean to provide incentive while at the same time achieving fair resource allocation. Future versions of TRANSIT could incorporate similar mechanisms. One could even think about possible transitions between incentive mechanisms, thereby adapting the rate between social welfare and a peer's individual welfare based on the current environmental conditions. For example, during a news broadcast after an important event, the social welfare might be more important as we want all peers to receive the stream at acceptable quality. How transitions could be triggered based on events in the real world could be part of future work on transitions in P2P.

8.2 Future Work on Transitions in Peer-to-Peer

Currently, the system adapts based on local information of a peer, i.e. transitions are executed locally, without any central coordination. An interesting question is the utilization of the streaming source for controlled transitions. Consider the following scenario: a streaming service is offered as a pure client/server system during times with moderate load. Thereby, peers experience shorter startup delays and high playback continuity. Now, if the load on the source exceeds a given threshold, the streaming system automatically executes a transition from a pure client/server system to a P2P system. This peer-assisted system model would allow scalability comparable to today's cloud offerings, but at lower operation costs for the service provider. With TRANSIT, one can easily add centralized control over any configuration parameter, thereby evaluating the impact of centralized decisions regarding transitions. This, of course, requires a monitoring solution in order to collect relevant information at the source. Providing such a monitoring service on the neighborhood layer is an interesting topic for future work, as monitoring would also allow more differentiated scheduling decisions on peers.

TRANSIT and the results presented in this thesis are a first step towards the support of transitions in P2P systems. The deployment of TRANSIT on a testbed such as GLab or PlanetLab is an inevitable next step in order to evaluate the performance of a transition-enabled system under real-world conditions. From the promising results shown in our evaluation of TRANSIT, we expect the transition-enabled system to perform similarly well *in the wild*.

A Configuration Files for PeerfactSim.KOM

The simulator PeerfactSim.KOM is configured through XML-files. Excerpts of those configuration files are shown below, with Listing A.1 showing the configuration of the metrics analyzer as introduced in Chapter 6. Listing A.1 gives an overview over the configuration structure used for the evaluation of TRANSIT. Due to space constraints, some parts of the configuration file are omitted.

```
<Monitor class="common.DefaultMonitor" start="0m" stop="6h" experimentDescription="Transit">
  <Analyzer class="analyzer.metric.MetricAnalyzer">
    <!-- Raw-Metrics counting incoming packets for flows and all others -->
    <Metric class="overlay.streaming.transit.metric.MVideoPackets$CountBlocks"
        packetType="Noduplicate" reason="Receive"/>
    <Metric class="overlay.streaming.transit.metric.MVideoPackets$CountBlocks" packetType="Live"
        reason="Receive"/>

    <!-- Delta-Filter that periodically calculates the change of both metrics over one interval -->
    <Filter class="analyzer.metric.filter.IntervalDeltaFilter" intervalLength="1m" prefix="Delta"
        whitelist="CountBlocksReceiveNoduplicate;CountBlocksReceiveLive"/>

    <!-- The flow ratio is calculated from the sampled delta-metrics -->
    <Filter class="analyzer.metric.filter.RatioFilter$SimpleRatio" metricName="MFlowRatio"
        nominator="Delta_CountBlocksReceiveLive" denominator="Delta_CountBlocksReceiveNoduplicate"
        unit="NONE" description="Flow Ratio of a peer (ratio of flow-packets vs. total-packets)"/>

    <!-- Per-Peer output is averaged and plotted live during the simulation -->
    <Filter class="analyzer.metric.filter.StatisticsFilter$Avg" whitelist="MFlowRatio"/>
    <Filter class="analyzer.metric.filter.StatisticsFilter$Percentile" percentile="75"
        whitelist="MFlowRatio"/>
    <Filter class="analyzer.metric.filter.StatisticsFilter$Percentile" percentile="25"
        whitelist="MFlowRatio"/>
    <Output class="analyzer.metric.output.MetricOutputLivePlot" whitelist="MFlowRatio"
        upperPercentile="75" lowerPercentile="25" enableCDF="true" samplingInterval="1m"/>

    <!-- Per-Peer output is written to the database, each time a new delta is calculated -->
    <Output class="analyzer.metric.output.MetricOutputDAO" table="transit" whitelist="MFlowRatio"/>
  </Analyzer>
</Monitor>
```

Listing A.1: Configuration of the metrics analyzer for measuring the flow ratio on peers. The flow ratio is calculated by passing primitive metrics through filters. The resulting metric can then be plotted live during the simulation and at the same time be written to the database. This concept helps during the debugging and implementation process, as design decisions can be evaluated instantaneously.

```
<Configuration xmlns:xi="http://www.w3.org/2001/XInclude">
  <SimulatorCore class="de.tud.kom.p2psim.impl.simengine.Simulator" static="getInstance" seed="1"
      finishAt="6h"/>

  <!-- Cloud topology, assuming end-to-end connectivity and latencies from 50ms to 150ms -->
  <Topology class="topology.TopologyFactory">
    <View class="topology.views.CloudTopologyView" phy="ETHERNET" linkDrop="0.0">
      <Latency class="topology.views.StaticLatency" latency="100ms" variance="50ms"/>
    </View>
  </Topology>

  <!-- Link layer of a DSL-connected peer -->
  <LinkLayer class="linklayer.LinkLayerFactory">
    <Mac class="linklayer.mac.configs.SimpleMac" phy="ETHERNET" trafficQueueSize="100"
        downBandwidth="15326Kbps" upBandwidth="2253Kbps"/>
  </LinkLayer>

  <!-- NetLayer with global knowledge routing, one hop for the message delivery -->
  <NetLayer class="network.routed.RoutedNetLayerFactory" enableFragmenting="false">
    <Routing class="network.routed.config.Routing" phy="ETHERNET" algorithm="GLOBAL_KNOWLEDGE"
        protocol="IPv4"/>
  </NetLayer>

  <TransLayer class="transport.modular.ModularTransLayerFactory"/>

  <!-- Streaming overlay with default configuration, normal peer (NODE) -->
  <Overlay class="overlay.streaming.transit.TransitFactory" factoryType="NODE" />

  <!-- Application used for live streaming -->
  <Application class="application.streaming.StreamingAppFactory" propertyFile="streaming.properties"
      liveStreaming="true"/>

  <!-- User and trace-based churn model -->
  <User class="user.streaming.StreamingUserFactory"/>
  <ChurnGenerator class="churn.DefaultChurnGenerator" start="1h">
    <ChurnModel class="churn.TraceBasedChurnModel" file="data/traces/pplive_ch2"/>
  </ChurnGenerator>

  <!-- Host builder, defining the number of peers and their configuration -->
  <HostBuilder class="scenario.DefaultHostBuilder" experimentSize="1">
    <!-- [...] -->
  </HostBuilder>
</Configuration>
```

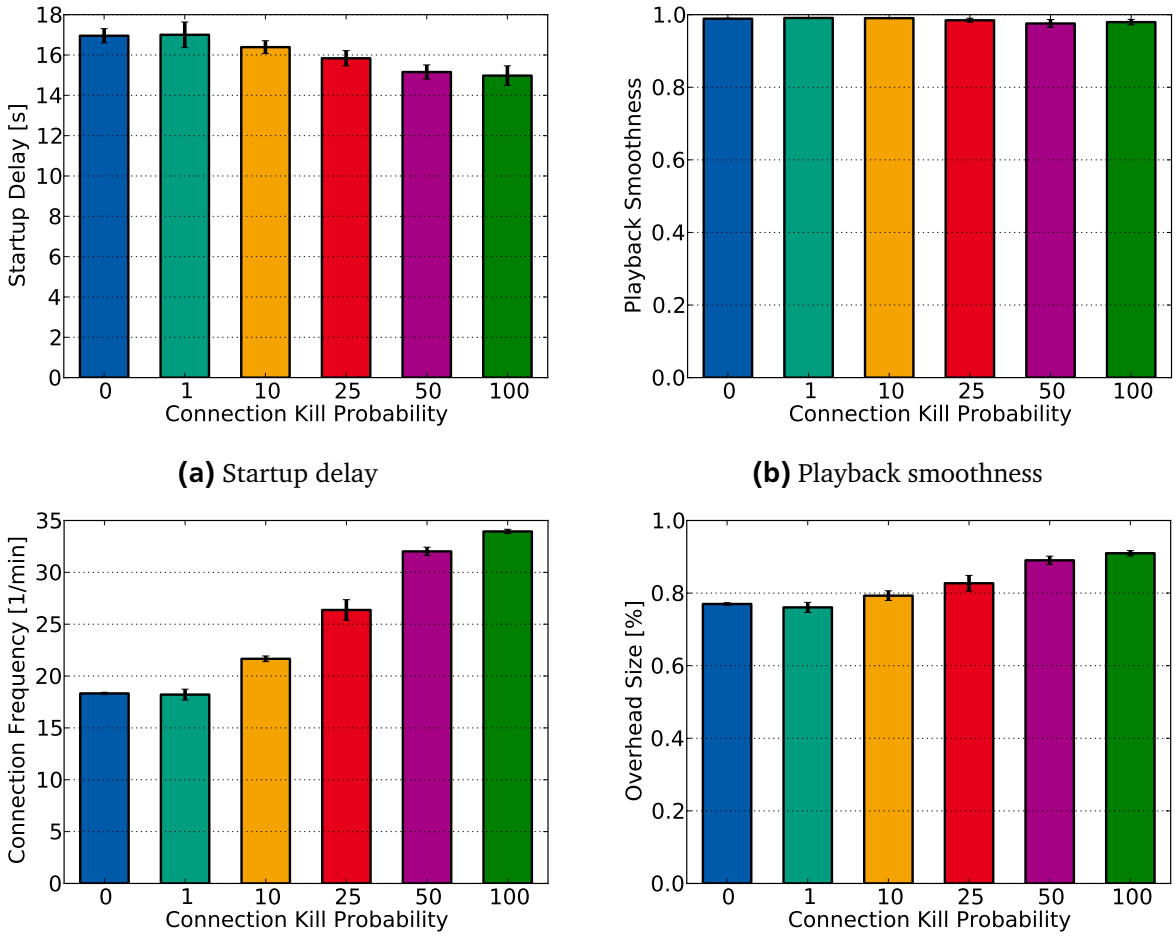
Listing A.2: Simplified configuration file containing all functional layers of the simulator. The underlay consists of topology, link layer, network layer and transport layer. The application is triggered by the user model, which is in turn directly determined by the churn model. In between the application and the transport layer, the overlay manages the dissemination of the video stream. All components are the configured for groups of hosts in the HostBuilder, which is not shown due to space constraints.

B Additional Evaluation Results

Additional results from the evaluation presented in Chapter 7 are presented in this chapter. These include the parameter evaluation under workload model B3, as well as additional data from the evaluation of transitions that did not make it into Chapter 7 due to space constraints.

B.1 Parameter Evaluation

In this section, additional results for the parameter evaluation of TRANSIT under workload model B3 are briefly presented and discussed.



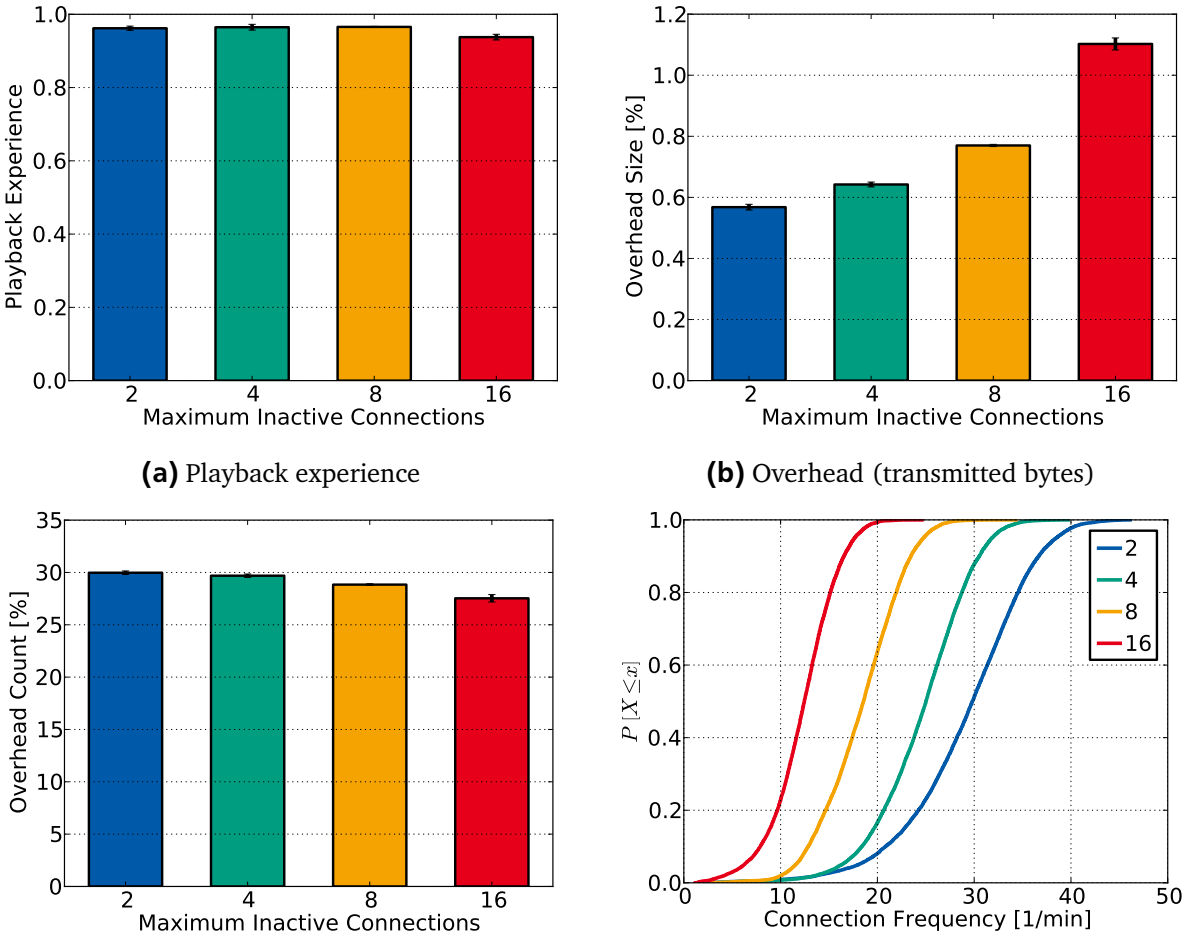
(a) Startup delay

(b) Playback smoothness

(c) Connection frequency

(d) Overhead

Figure B.1.: Impact of actively dropping connections with a given probability each time a new neighborhood is received by a peer. The startup delay is slightly better with a higher value for p_{kill}, while at the same time the playback smoothness decreases. Considering the additional overhead due to an increased connection frequency and the negative impact on the playback smoothness, the mechanism is not used in TRANSIT.



(a) Playback experience

(b) Overhead (transmitted bytes)

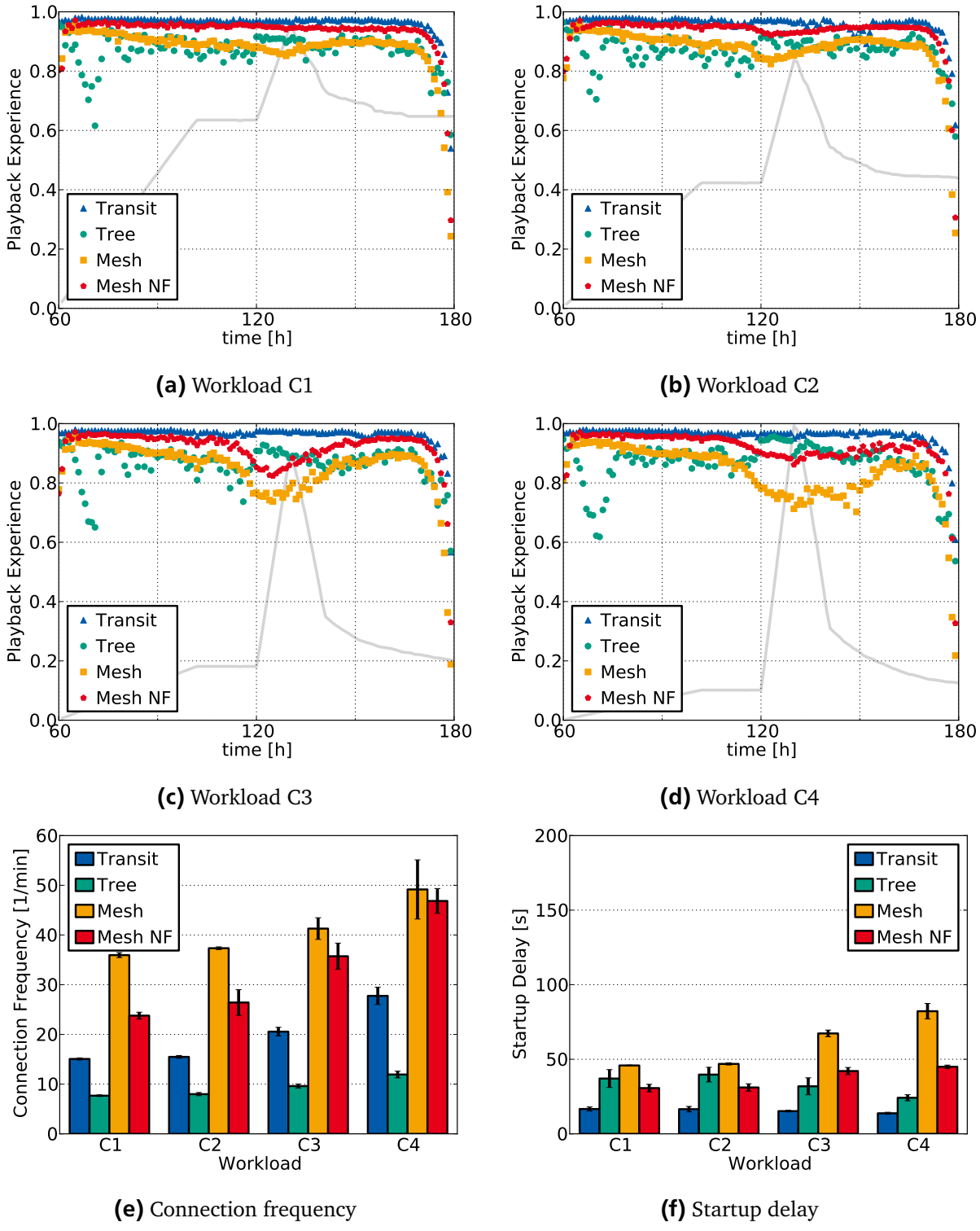
(c) Overhead (number of messages)

(d) Connection frequency

Figure B.2.: In order to decrease the frequency of neighborhood changes, a given number ($\max|C_{\leftarrow}|_{\text{stale}}$) of inactive connections is tolerated. The playback experience is best for $\max|C_{\leftarrow}|_{\text{stale}} = 0,5\max|C_{\leftarrow}|$. The decrease in connection frequency for higher values of $\max|C_{\leftarrow}|_{\text{stale}}$ and, thus, a small decrease in the overhead in terms of messages being sent, is compensated by an increase in the overhead considering the transmitted data.

B.2 Evaluation of Transitions

In this section, additional results from the evaluation of TRANSIT in comparison to its mesh-based and tree-based configurations are presented.



(a) Workload C1

(b) Workload C2

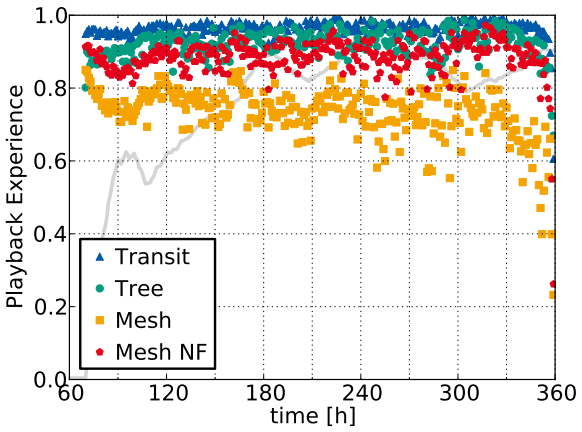
(c) Workload C3

(d) Workload C4

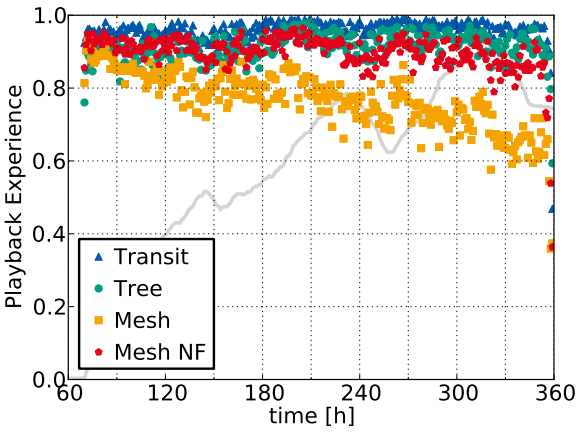
(e) Connection frequency

(f) Startup delay

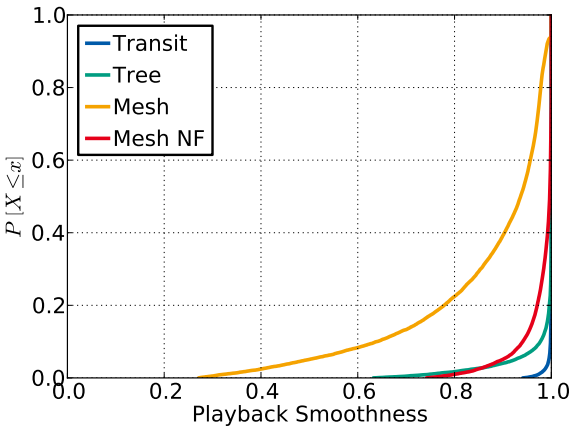
Figure B.3.: Playback experience for peers depending on their arrival time during different flash crowd intensities. The average connection frequency of peers in TRANSIT adapts to the intensity of the flash crowd. The startup delay of the tree-based configuration decreases during more intense flash crowds, as new blocks are faster disseminated throughout parts of the overlay than in the mesh-based configurations.



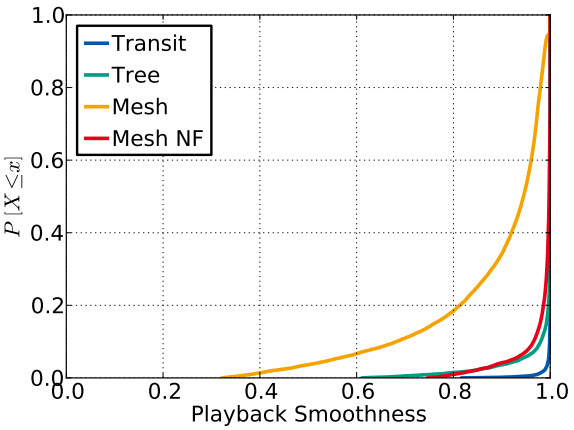
(a) Playback experience for workload D1



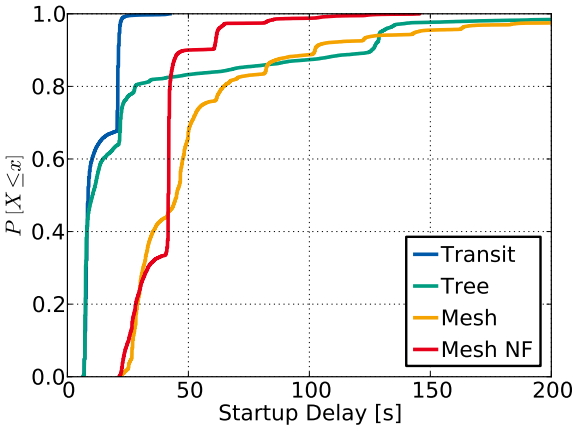
(b) Playback experience for workload D2



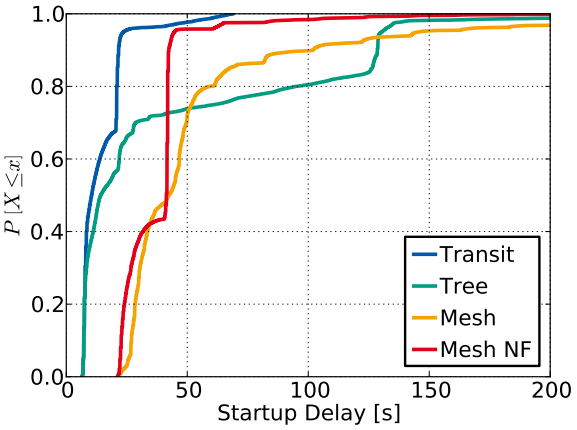
(c) Playback smoothness for workload D1



(d) Playback smoothness for workload D2



(e) Startup delay for workload D1



(f) Startup delay for workload D2

Figure B.4.: The playback experience of the pure mesh decreases over time, whereas the other configurations and TRANSIT achieve a consistent performance. Peers in TRANSIT experience a higher playback smoothness and lower startup delays than peers in the other configurations. The impact of heterogeneous peers is most severe for the tree-based configuration, where 20 % of all peers experience a startup delay of more than two minutes.

Acronyms

ALM Application Layer Multicast.
AS Autonomous System.
AU Access Unit.
AVC Advanced Video Coding.

CDN Content Distribution Network.

DHT Distributed Hash Table.
DNS Domain Name System.

ESM End System Multicast.

HTTP Hypertext Transfer Protocol.

IP Internet Protocol.
IPTV Internet Protocol Television.
ISP Internet Service Provider.

MDC Multiple Description Coding.

NAT Network Address Translation.

OECD Organisation for Economic Co-operation and Development.

P2P Peer-to-Peer.
PQA Progressive Quality Adaptation.

QoE Quality of Experience.
QoS Quality of Service.

RTT Round Trip Time.

SNR Signal-to-Noise Ratio.
SVC Scalable Video Codec.

VoD Video on Demand.
VQM Video Quality Metric.

Bibliography

[AA96] Kevin C. Almeroth and Mostafa H. Ammar. Collecting and Modeling the Join/Leave Behavior of Multicast Group Members in the MBone. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 209–216, 1996.

[APK+11] Osama Abboud, Konstantin Pussep, Aleksandra Kovacevic, Katharina Mohr, Sebastian Kaune, and Ralf Steinmetz. Enabling Resilient P2P Video Streaming: Survey and Analysis. *Multimedia Systems*, 17(3):177–197, 2011.

[APKS09] Osama Abboud, Konstantin Pussep, Aleksandra Kovacevic, and Ralf Steinmetz. Quality Adaptive Peer-to-Peer Streaming Using Scalable Video Coding. In *Wired-Wireless Multimedia Networks and Services Management*, volume 5842 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 2009.

[ASK10] Suphakit Awiphan, Zhou Su, and Jiro Katto. ToMo: A Two-Layer Mesh/Tree Structure for Live Streaming in P2P Overlay Network. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, pages 1–5, 2010.

[ATC07] Emrah Akyol, A. Murat Tekalp, and M. Reha Civanlar. A Flexible Multiple Description Coding Framework for Adaptive Peer-to-Peer Video Streaming. *IEEE Journal of Selected Topics in Signal Processing*, 1(2):231–245, 2007.

[AZP+11] Osama Abboud, Thomas Zinner, Konstantin Pussep, Sabah Al-Sabea, and Ralf Steinmetz. On the Impact of Quality Adaptation in SVC-based P2P Video-on-Demand Systems. In *Proceedings of the ACM Conference on Multimedia Systems (MMSys)*, pages 223–232, 2011.

[BNSG07] Pierpaolo Baccichet, Jeonghun Noh, Eric Setton, and Bernd Girod. Content-aware P2P Video Streaming with low Latency. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 400–403, 2007.

[BSWG07] Pierpaolo Baccichet, Thomas Schierl, Thomas Wiegand, and Bernd Girod. Low-delay Peer-to-Peer Streaming using Scalable Video Coding. In *Proceedings of the IEEE International Packet Video Workshop (PV)*, pages 173–181, 2007.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proceedings of the ACM Symposium on Operating Systems Principles (SIGOPS)*, pages 298–313, 2003.

[CDKR02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[Cis11] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2010–2015. Technical report, 2011.

[CJK+03] Miguel. Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman. An Evaluation of Scalable Application-Level Multicast Built Using Peer-to-Peer Overlays. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, volume 2, pages 1510 – 1520 vol.2, 2003.

[CN03] Yi Cui and Klara Nahrstedt. Layered Peer-to-Peer Streaming. In *Proceedings of the ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, page 162, 2003.

[Coh03] Bram Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer systems (P2PECON)*, volume 6, pages 68–72, 2003.

[DLL+00] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, 14(1):78–88, 2000.

[GKM01] Ayalvadi Ganesh, Anne-Marie Kermarrec, and Laurent Massoulie. SCAMP: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication. In *Networked Group Communication*, volume 2233 of *Lecture Notes in Computer Science*, pages 44–55. Springer, 2001.

[Goy01] Vivek K. Goyal. Multiple Description Coding: Compression Meets the Network. *IEEE Signal Processing Magazine*, 18(5):74–93, 2001.

[GYL+08] Yang Guo, Shengchao Yu, Hang Liu, Saurabh Mathur, and Kumar Ramaswamy. Supporting VCR Operation in a Mesh-Based P2P VoD System. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, pages 452–457, 2008.

[HGL11] Hao Hu, Yang Guo, and Yong Liu. Peer-to-peer streaming of layered video: Efficiency, fairness and incentive. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(8):1013–1026, 2011.

[HLL+07] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and Keith W. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, 9(8):1672–1687, 2007.

[ITU10] ITU. ITU-T H.264 - Advanced Video Coding for Generic Audiovisual Services. Technical report, 2010.

[JGJ+00] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the ACM Symposium on Operating System Design and Implementation (OSDI)*, volume 4, pages 14–29, 2000.

[JMB09] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-Man: Gossip-based fast Overlay Topology Construction. *Computer Networks*, 53(13):2321–2339, 2009.

[KGK+08] Aleksandra Kovacevic, Kalman Graffi, Sebastian Kaune, Christof Leng, and Ralf Steinmetz. Towards Benchmarking of Structured Peer-to-Peer Overlays for Network Virtual Environments. In *Proceedings of the IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 799–804, 2008.

[LdSL+09] Richard John Lobb, Ana Paula Couto da Silva, Emilio Leonardi, Marco Mellia, and Michela Meo. Adaptive Overlay Topology for Mesh-Based P2P-TV Systems. In *Proceedings of the ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 31–36, 2009.

[LGL08a] Chao Liang, Yang Guo, and Yong Liu. Is Random Scheduling Sufficient in P2P Video Streaming? In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 53–60, 2008.

[LGL08b] Yong Liu, Yang Guo, and Chao Liang. A Survey on Peer-to-Peer Video Streaming Systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, 2008.

[LRL08] Jiangchuan Liu, S.G. Rao, and Bo Li. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. *Proceedings of the IEEE*, 96(1):11–24, 2008.

[LWLZ09] Zimu Liu, Chuan Wu, Baochun Li, and Shuqiao Zhao. Distilling Superior Peers in Large-Scale P2P Streaming Systems. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 82–90, 2009.

[LXK+07] Bo Li, Susu Xie, Gabriel Y. Keung, Jiangchuan Liu, Ion Stoica, Hui Zhang, and Xinyan Zhang. An Empirical Study of the Coolstreaming+ System. *IEEE Journal on Selected Areas in Communications*, 25(9):1627–1639, 2007.

[LXQ+08] Bo Li, Susu Xie, Yang Qu, Gabriel Y. Keung, Chuang Lin, Jiangchuan Liu, and Xinyan Zhang. Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 1031–1039, 2008.

[MC00] Warren Matthews and Les Cottrell. The PingER Project: Active Internet Performance Monitoring for the HENP Community. *IEEE Communications Magazine*, 38(5):130–136, 2000.

[MR09] Nazanin Magharei and Reza Rejaie. PRIME: Peer-to-Peer Receiver-driven Mesh-based Streaming. *IEEE/ACM Transactions on Networking (TON)*, 17(4):1052–1065, 2009.

[MRG07] Nazanin Magharei, Reza Rejaie, and Yang Guo. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 1424–1432, 2007.

[NBD09] Animesh Nandi, Bobby Bhattacharjee, and Peter Druschel. What a Mesh: Understanding the Design Tradeoffs for Streaming Multicast. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):85–86, 2009.

[NBH+09] Jeonghun Noh, Pierpaolo Baccichet, Frank Hartung, Aditya Mavlankar, and Bernd Girod. Stanford Peer-to-Peer Multicast (SPPM) - Overview and Recent Extensions. In *Proceedings of the IEEE Picture Coding Symposium (PCS)*, pages 1–4, 2009.

[NLE10] Anh Tuan Nguyen, Baochun Li, and Frank Eliassen. Chameleon: Adaptive Peer-to-Peer Streaming with Network Coding. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9, 2010.

[OEC12] Organisation for Economic Co-operation and Development. *OECD Broadband Report*, 2012. `http://www.oecd.org/internet/broadbandandtelecom/oecdbroadbandportal.htm` Accessed: 03/11/2012.

[PKV+10] Michael Piatek, Arvind Krishnamurthy, Arun Venkataramani, Richard Yang, David Zhang, and Alexander Jaffe. Contracts: Practical Contribution Incentives for P2P Live Streaming. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (OSDI)*, pages 6–16, 2010.

[PPK07] Fabio Pianese, Diego Perino, and Joaquín Keller. PULSE: An Adaptive, Incentive-Based, Unstructured P2P Live Streaming System. *IEEE Transactions on Multimedia*, 9(8):1645–1660, 2007.

[PW04] Margaret H. Pinson and Stephen Wolf. A New Standardized Method for Objectively Measuring Video Quality. *IEEE Transactions on Broadcasting*, 50(3):312–322, 2004.

[PWC03] Venkata N. Padmanabhan, Helen J. Wang, and Philip A. Chou. Resilient Peer-to-Peer Streaming. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, pages 16–27, 2003.

[PWCS02] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou, and Kunwadee Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. *Proceedings of the ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 177–186, 2002.

[RAZ+12] Julius Rückert, Osama Abboud, Thomas Zinner, Ralf Steinmetz, and David Hausheer. Quality Adaptation in P2P Video Streaming Based on Objective QoE Metrics. In *NETWORKING 2012*, volume 7290 of *Lecture Notes in Computer Science*, pages 1–14. 2012.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware 2001*, pages 329–350, 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172, 2001.

[RH12] Julius Rückert and David Hausheer. Bridging the Gap: Towards an Adaptive Video Streaming Approach Supporting Transitions. *Dependable Networks and Services*, pages 38–41, 2012.

[San11] Sandvine. Global Internet Phenomena Spotlight - Netflix Rising. Technical report, 2011.

[San12] Sandvine. Spring 2012 Global Internet Phenomena Report. Technical report, 2012.

[SGR+11] Dominik Stingl, Christian Gross, Julius Ruckert, Leonhard Nobach, Aleksandra Kovacevic, and Ralf Steinmetz. PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems. In *Proceedings of the IEEE International Conference on High Performance Computing and Simulation (HPCS)*, pages 577–584, 2011.

[SLZV11] Zhijie Shen, Jun Luo, Roger Zimmermann, and Athanasios V. Vasilakos. Peer-to-Peer Media Streaming: Insights and New Developments. *Proceedings of the IEEE*, 99(12), 2011.

[SM10] Nicolas Staelens and Stefaan Moens. Assessing Quality of Experience of IPTV and Video on Demand Services in Real-Life Environments. *IEEE Transactions on Broadcasting*, 56(4):458–466, 2010.

[STL04] Gary J. Sullivan, Pankaj Topiwala, and Ajay Luthra. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In *Proceedings of the SPIE Conference on Applications of Digital Image Processing*, pages 1–21, 2004.

[SW08] Heiko Schwarz and Mathias Wien. The Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Signal Processing Magazine*, 25(2):135–141, 2008.

[SZFNR08] Jeff Seibert, David Zage, Sonia Fahmy, and Cristina Nita-Rotaru. Experimental Comparison of Peer-to-Peer Streaming Overlays: An Application Perspective. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN)*, pages 20–27, 2008.

[TAB+12] Stefano Traverso, Luca Abeni, Robert Birke, Csaba Kiraly, Emilio Leonardi, Renato Lo Cigno, and Marco Mellia. Experimental Comparison of Neighborhood Filtering Strategies in Unstructured P2P-TV Systems. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 13–24, 2012.

[Tan81] Andrew S. Tanenbaum. Network protocols. *ACM Computing Surveys*, 13(4):453–489, 1981.

[VCA09] Alex Borges Vieira, Sergio Campos, and Jussara Almeida. Fighting Attacks in P2P Live Streaming: Simpler is Better. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 355–356, 2009.

[VGNL10] Long Vu, Indranil Gupta, Klara Nahrstedt, and Jin Liang. Understanding Overlay Characteristics of a Large-Scale Peer-to-Peer IPTV System. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 6(4):1–24, 2010.

[VGvS05] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[Win07] Stefan Winkler. Video Quality and Beyond. In *Proceedings of the EURASIP European Signal Processing Conference (EUSIPCO)*, pages 150–153, 2007.

[WLX08] Feng Wang, Jiangchuan Liu, and Yongqiang Xiong. Stable Peers: Existence, Importance, and Application in Peer-to-Peer Live Video Streaming. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 1364–1372, 2008.

[WRL05] Yao Wang, Amy R. Reibman, and Shunan Lin. Multiple Description Coding for Video Delivery. *Proceedings of the IEEE*, 93(1), 2005.

[WXL07] Feng Wang, Yongqiang Xiong, and Jiangchuan Liu. mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2007.

[YLZ+09] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky. In *Proceedings of the ACM International Conference on Multimedia (MM)*, pages 25–34, 2009.

[ZH12] Xiangyang Zhang and Hossam Hassanein. A Survey of Peer-to-Peer Live Video Streaming Schemes – An Algorithmic Perspective. *Computer Networks*, 56(15):3548–3579, 2012.

[ZIPE10] Boxun Zhang, Alexandru Iosup, Johan Pouwelse, and Dick Epema. The Peer-to-Peer Trace Archive: Design and Comparative Trace Analysis. In *Proceedings of the ACM CoNEXT Student Workshop*, pages 2–3, 2010.

[ZLLY05] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. CoolStreaming/DONet: a Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 2102–2111, 2005.

[ZZSY07] Meng Zhang, Qian Zhang, Lifeng Sun, and Shiqiang Yang. Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? *IEEE Journal on Selected Areas in Communications*, 25(9):1678–1694, 2007.