TOPT: Supporting Flash Crowd Events in Hybrid Overlay-based Live Streaming

Julius Rückert*, Björn Richerzhagen†, Eduardo Lidanski*, Ralf Steinmetz† and David Hausheer*
* Peer-to-Peer Systems Engineering Lab (PS), E-Mail: {rueckert|eduardo|hausheer}@ps.tu-darmstadt.de
† Multimedia Communications Lab (KOM), E-Mail: {bjoern.richerzhagen|ralf.steinmetz}@kom.tu-darmstadt.de
Technische Universität Darmstadt, Germany

***Abstract*—Recent studies show that an increasing number of over-the-top live streams is delivered over the Internet. For the delivery of those streams, the dynamically changing and potentially large number of users imposes a major challenge. Flash crowds, where the number of users multiplies or significantly drops in a very small time frame, can cause serious degradations in the streaming performance. Due to the missing support for global network-layer multicast, overlay-based approaches have been broadly studied, showing that, with relaxed time constraints, they can scale well with the number of users. Yet, to support flash crowds, scaling has to happen quickly to keep up also with rapidly changing populations. Only a few approaches exist that focus on this aspect by influencing the streaming topology and, so far, it is not clear if and how these mechanisms can be applied to state-of-the-art hybrid streaming systems. Therefore, in this paper, TOPT is proposed, integrating new as well as existing mechanisms in a common framework. The evaluation shows that the streaming topology, indeed, plays a major role during flash crowds. The lightweight and decentralized tree-forming and topology optimization mechanisms of TOPT, combined with tracker extensions to attach new peers in batches, greatly help improving the streaming performance in terms of reduced playback interruptions by more than 60% and slight reduction in communication overhead at an acceptable increase in average startup delays by 24%.**

I. INTRODUCTION

Recent studies [6], [10], [23] show that video continues to be the dominating traffic class on the Internet. They also show that the delivery of live streams [7] becomes a natural part of this class, where people start shifting their so called *linear broadcast consume* to Internet-enabled devices. In addition, also a growing number of special events are delivered in a live manner over the Internet, such as product announcements or sports events. Many of these events are delivered in an over-the-top manner, where the network providers and, in particular, the Internet Service Providers (ISP) are not explicitly involved in the delivery process but are only used as transport networks. This, the missing support for global network-layer multicast, and the resulting tremendous bandwidth requirements for the unicast delivery of high-quality video streams to potentially millions of users across the world, make the delivery of live streams very challenging. Content Delivery Networks (CDNs) with more than 100k servers deployed all over the world are used today to mitigate this problem [18]. Yet, CDNs still face problems when the number of users and, thereby the streams to be delivered, exceeds their capacity. For this reason CDNs started to incorporate the upload resources of clients [33] into the delivery process and, thereby, to extend their overlay to the edge of the network. Such Peer-to-Peer (P2P) approaches have been broadly studied for live stream delivery [32], showing that they have desirable features and, in particular, can scale well with the number of users in case of relaxed time constraints.

ISBN 978-3-901882-68-5

The challenge of a P2P-based streaming approach to scale well with the number of connected users, on an abstract level, can be reduced to the problem of successfully integrating upload resources of new clients into the active delivery process. This process is mainly limited by the amount of surplus upload capacity of already active peers. Using this capacity, a new client first mostly consumes resources until it is able to serve others at a constant rate. This can quickly become a problem in face of highly dynamic and large audiences, as the time a new client requires to serve others in a stable manner and, thus, adds to the overall system capacity, heavily constrains the system's scaling behavior [4]. The results can be serious degradations in the streaming performance in terms of long startup delays for new clients and playback interruptions for already connected clients. For this reason, flash crowds, where the number of users multiplies or significantly drops in a very small time frame, are important to be considered as they are common and an integral part of live streaming use cases [11], [26]. To support flash crowds, scaling has to work under heavy time constraints and keep up with rapidly changing client populations. Only a few approaches exist that focus on this aspect by optimizing aspects of the streaming topology and, so far, it is not clear if and how the proposed mechanisms can be adapted for state-of-the-art hybrid streaming systems, relying both on mesh and multi-tree delivery structures.

Therefore, in this paper, the streaming system TOPT is proposed. It builds upon the state-of-the-art hybrid streaming approach called TRANSIT [28] and integrates newly proposed as well as existing mechanisms to support flash crowds in a common framework. Using this framework, the goal is to investigate the importance of the streaming topology for the level of achievable streaming performance during flash crowds. Therefore, a set of topology optimizations are studied to understand the effect on TOPT's multi-tree streaming topology and tracker extensions are investigated that could to stabilize the attachment process of new clients during flash crowds.

The remainder of the paper is structured as follows: Section II provides an overview on the TRANSIT streaming approach, which builds the basis for TOPT. Section III discusses existing works in the area of flash crowd support in P2P live streaming, followed by the system design of TOPT in Section IV. Subsequently, the evaluation of the approach is presented in Section V. Section VI concludes the paper.

II. Background: TRANSIT Design

In a recent work [28], the authors presented a hybrid streaming system called TRANSIT. As the mechanisms proposed in this paper extend TRANSIT, relevant core concepts of this system are briefly presented in the following. For a more detailed discussion of these concepts, the interested reader is referred to an extended version of this paper [22] and the detailed presentation of the original TRANSIT design [21].

TRANSIT is a P2P live streaming system, designed for the efficient distribution of layered video streams, using video codecs such as H.264/SVC [24] or H.265/SHVC [31]. The use of a layered video codec enables a natural division of the video stream into substreams for the delivery process. As each substream belongs to a specific video quality layer, this allows TRANSIT to support heterogeneous scenarios in which clients stream only the video quality that fits their device and network properties. However, similar to other state-of-the-art systems, TRANSIT can also be used with single-layer video codecs. In this case, a mechanism is required to split the video stream into a number of substreams.

Regarding the system structure, TRANSIT is a fully decentralized P2P approach with peers acting autonomously. The video stream is provided by a dedicated streaming source, which can consist of a single server or several servers run by a content provider. A logically centralized tracker is used for initial peer discovery, acting as node registry service. In addition, peers exchange information about available neighbors when establishing new connections. This way, peers maintain an up-to-date set of potential neighbors.

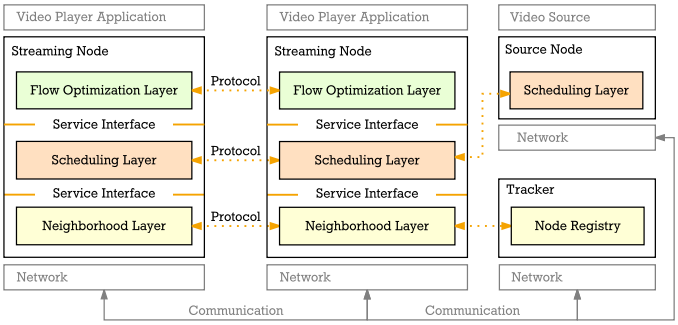


Fig. 1. Conceptual layers of TRANSIT, including the new *Flow Optimization Layer* added by TOPT and introduced in Section IV.

For a clear separation of concerns, TRANSIT is organized into three conceptual layers, as depicted in Figure 1. The individual streaming peers run all three layers, while the streaming source and the tracker only operate on one layer each. Maintaining a stable set of connected neighbors, thus, discovering new neighbors, establishing new connections, monitoring established connections, and replacing unreliable ones, is the task of the *Neighborhood Layer*. The tracker provides peers with initial contacts to other peers, thereby belonging to mechanisms in the *Neighborhood Layer*. At the peers, this layer provides the other layers with a set of active connections through its service interface.

The *Scheduling Layer* uses the connections provided by the *Neighborhood Layer* for distributing individual video blocks of a stream. Due to the hybrid nature of TRANSIT, this layer includes different mechanisms for scheduling video block transmissions. By default, data is transferred following a pull-based approach, where peers request individual blocks from their connected neighbors. For this purpose, peers periodically exchange buffermaps, indicating available video blocks. Based on their neighbors' buffermaps, peers can request locally missing blocks. Such a *Request* can ask for a single block or a set of blocks at once to keep the communication overhead low. For a healthy and efficient P2P-based delivery process of live streams, it is essential that especially new blocks are disseminated quickly to allow all peers to contribute in the distribution. Magharei and Rejaie [17], [20] propose a two-phase delivery process for this purpose, where in a *diffusion phase* video blocks are aggressively spread to a subset of the peers, followed by a *swarming phase* in which these well replicated blocks are disseminated to the remaining peers. Motivated by these observations, TRANSIT was designed to ensure a fast diffusion of new blocks. It goes beyond the approach of Magharei and Rejaie by adopting mechanisms from hybrid streaming systems, such as MTREEBONE [27], where for the diffusion of blocks stable distribution trees are established. This constitutes the second scheduling mechanisms of TRANSIT, the so called *Flows*, complementing the above mentioned *Requests*. *Flows* can be negotiated between peers to establish stable data paths to constantly deliver substreams of blocks in a push-based manner, avoiding undesired delays in the diffusion process. A single flow in TRANSIT can include multiple substreams, reducing the communication overhead for flow management. For more details on the *Neighborhood* and *Scheduling Layer*, the interested reader is referred to [21].

III. Related Work

In [14], [16], the authors show that scaling with the number of users during flash crowds is hard to achieve. As observed by Cheng et al. [4], a key aspect to achieve such scalability is to quickly include new clients in the active delivery process to assure that the growth in upload capacity overall outweighs the capacity consumption by these clients. Liu et al. [16] denote this fundamental requirement as the *scale-time* constraint of a streaming system. Especially at the beginning of a flash crowd period, new clients can easily outnumber currently active clients, leading to a situation where new peers heavily compete for the small amount of available upload capacity. However, addressing this problem is challenging due to the decentralized nature of P2P streaming systems. Peers, per definition, only have a local view on the system and, thus, the discovery and selection of the neighbors with sufficient capacity is not trivial. Current productive streaming systems, such as COOLSTREAMING or PPLIVE, rely on a large number of dedicated servers or the use of CDNs to support flash crowds [16], [29]. While this can be a valid solution in some scenarios, it contradicts the motivation of using a P2P approach in the first place. A number of different approaches have been proposed that preserve the decentralized nature, while actively supporting the system during flash crowds.

Chung et al. [5] propose a batch join tracker extension for alleviating flash crowd effects, where large groups (batches) of join requests are processed together during flash crowds. The tracker introduces the peers inside a batch to each other in a way to pre-build a tree topology, based on the upload bandwidth and waiting time of the individual peers. Once the

tree is formed, the root is connected to the active topology, enabling stream delivery for the whole batch. This greatly reduces the effects of peers competing for upload resources and allows them to quickly contribute to the delivery process.

Liu et al. [16] propose a population control mechanism to limit the arrival rate of peers during flash crowds. They show that the relative ability of a streaming system to scale can be modeled as a function of the relative average peer surplus capacity, the number of initial neighbors, and the relative peer arrival rate. This function has a maximum, the so-called *resiliency threshold*, after which the ability of the system to scale significantly drops due to the competition for resources. The authors propose to use this threshold as a control point and partially delay the joining of new peers at the tracker if this threshold is exceeded. They show that, while the approach increases the startup delay for some peers, it greatly improves the overall system scalability under flash crowds. Cheng et al. in [3] present similar mechanisms and support these findings.

Furthermore, different works focus on the role of streaming topologies in different aspects, which are considered to be closely related to a system's ability to support flash crowds. Wang et al. [27] propose MTREEBONE, a single-tree extension to a mesh-based system, where decentralized topology optimizations are used to reduce the depth of the tree structure and, thereby, the average delivery delay. A positive side effect of such topology optimizations is that undesired structures in the delivery trees are removed. This avoids that the topology becomes fragile and thus could lead to even worse resource competitions if during a flash crowd important peers, e.g. such with many children, leave the system. In this context, the area of resilient topologies can be considered closely related, which is further discussed in the extended version of this paper [22].

IV. TOPT: TOPOLOGIES FOR FLASH CROWDS

Based on the initial design of TRANSIT (cf. Section II), in this work, TOPOLOGY-OPTIMIZED TRANSIT (TOPT) is proposed, including a set of extensions to address undesired effects imposed by flash crowds. As state-of-the-art hybrid streaming system, TRANSIT is a suitable candidate to study these effects and design adequate countermeasures that could be adopted by other hybrid streaming systems as well.

The most prominent extension proposed to the initial TRANSIT design is the introduction of a third conceptual layer, the so called *Flow Optimization Layer* (cf. Figure 1). This layer uses the active flows exposed through the service interface of the *Scheduling Layer* to monitor and analyze the active flows of a peer. Based on local knowledge, this component can then plan and execute modifications to the flow topology. These modifications are carried out in a fully decentralized manner to maintain the scalability of the streaming approach.

A common goal when modifying stable parts of live streaming topologies is to optimize overall structural properties, e.g. shortening overlay paths for reduced delivery delays. Since decentralized modifications of streaming topologies inevitably impose control overhead for signaling between peers involved in a modification, the extensions proposed in this work focus on the dominating flow part of TRANSIT, leaving the concept of requests unaltered. Results from [28] show that flows build the backbone for the overall streaming process, where peers typically retrieve 80% or more of the video blocks using flows, even during highly dynamic workload scenarios. Therefore, optimizations of the flow topologies, even if imposing additional communication overhead, are expected to quickly pay off. The main reason is that, this way, optimizations have an impact on the performance of the rather long-termed flow delivery structures, in contrast to rather short-lived request structures. Figure 2 shows the individual components of TOPT, which are introduced individually in the following.

A. Topology Optimization Framework

The original concept of connections in TRANSIT was extended to support required primitives for generic topology modifications. By design, connections are unidirectional and a peer can only retrieve blocks via an incoming connection. Thus, one primitive type of modification is the ability to reverse the direction of connections. Coordinated with the individual nodes' *Flow Managers*, this allows flows, and thereby active data paths, to be reversed. This primitive is important to enable peers switching roles in the topology. A second required primitive is the ability to hand over flows to other peers. For this purpose, a number of new message types were introduced to the system's protocol. They can be used to query promising alternative peers for their ability to take part in a handover or switch of flows. To respect their autonomy, peers can always deny a request for optimization actions, e.g. if insufficient upload capacities for the required modification are available. To allow for seamless handovers, while avoiding the loss of video blocks, TOPT introduces the ability to establish duplicate flows for a short period of time. On receiving the first duplicate video block from one of the two flows, the old flow is immediately canceled, ending the handover with only a single duplicate block delivered. Using the two primitives described, the *Flow Optimization Manager* can greatly influence the role and position of a peer in the flow topologies and, thus, allows defining complex distributed topology optimizations.

To actually allow peers to decide on executing topology optimizations, they require basic knowledge on their current role and neighborhood in the topology. Therefore, a mechanism of TRANSIT is used that allows piggybacking information, such as current buffermaps, to periodically exchange control messages. This way, no additional messages are sent, greatly limiting the overhead for disseminating the local topology view. The exchanged information includes the current relative depth of a peer (as hop count from the source), the number of actively served children, its currently available upload slots, and the contact of the peer's parent in each of the active flow trees. This limited information is sufficient to allow a peer to identify potential optimizations based on its neighbors' states and plan their execution, as explained in the following.

To allow for a generic and extensible approach, where peers can evaluate potential optimizations based on the expected influence on the topology and select the one with the highest expected gain, a dedicated mechanism was designed for this purpose. Peers can locally define a set of optimization types that they consider for execution. While peers are not required to agree on the optimization types, optimizations have to be composed of the primitive modifications introduced above. Each of the locally available optimization types is evaluated for its applicability on the peer's individual active flows. Their

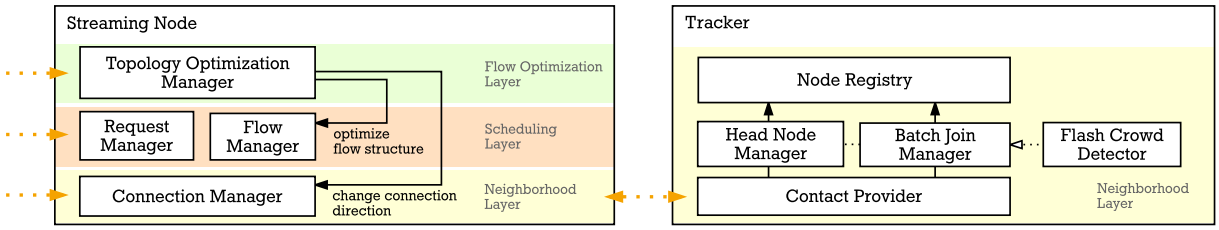


Fig. 2. Extended design, including the core components of the streaming node and tracker as well as their relations.

expected gains are calculated based on a scoring function included in the definition of the optimization types, allowing the peer to select the optimization instances with the highest score within an optimization type and apply it. Thereby, the approach goes beyond the mechanisms proposed by other state-of-the-art systems, such as MTREEBONE [27], where the first found optimization is executed and, in addition, is only defined for a single tree, instead of a more complex multi-tree topology exposed by the flows within TRANSIT and TOPT.

B. Optimizing Streaming Topologies

To understand the potential for topology optimizations in TRANSIT and in particular for helping the system to cope with flash crowds, its typical flow structures were analyzed. Figure 3 shows a simple example with a small number of peers and two flow trees. Despite its simplicity, several undesired topology properties can be observed that are characteristic also for larger scenarios. First of all, the topology appears rather sparse with most peers only serving a small number of other peers. In some cases peers only serve a single child, leading to undesired chains in the topology. Second, some of the peers deeper in the topology serve many children, although they themselves are only connected to the flows by chain-like structures. Third, individual subtrees of the same flow are highly unbalanced. Finally, different flows often use the same path, indicating combined flows that the system tends to build.

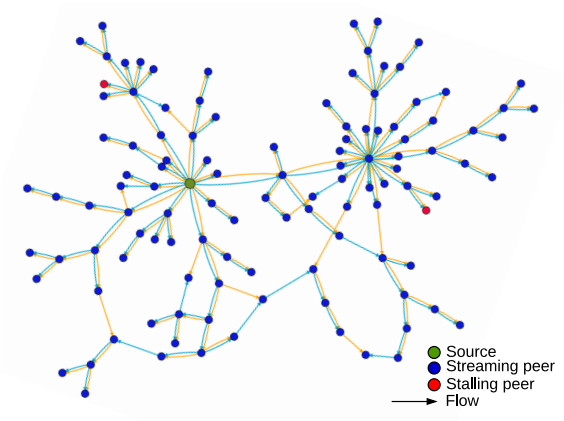


Fig. 3. Example topology for two flows as established by TRANSIT.

Most of the observations can be explained by design choices taken during the design of TRANSIT (cf. Section II). The chain-like and children dependencies are clearly a result of (1) the initial neighbors provided by the tracker, where newer peers are preferred due to the high probability of having free capacity, and (2) the fact, that established flows are kept active as long as possible. Both decisions showed to allow the system to quickly react on changing environmental conditions, such as smaller flash crowds [28]. Yet, these dependencies seem to also limit the ability to cope with more severe flash crowds. Due to the increasing length of chains and the unbalanced trees, it becomes harder with every new peer to achieve homogeneous and bounded streaming delays. Yet, they are important as they have a high impact on the user satisfaction [8], [9]. In addition, the topology becomes fragile when large fractions of peers suddenly leave the system, e.g. at the end of a flash crowd. This problem is further amplified in cases where a peer leaves that serves combined flows to others. In this case, affected children or even complete subtrees suddenly starve and quickly need to find new parents, contributing to the competition for upload resources of newly joining peers. The fact that the system does not collapse in such situations shows the strength of the hybrid design of TRANSIT. Yet, it imposes undesired overhead, playback interruptions, and dynamics in the topology that should be avoided. As shown in [2], it is important to minimize direct and indirect dependencies between nodes in the topology to increase its resilience against node failures. The deeper a tree topology is, the more indirect dependencies between nodes exist. Therefore, modifying the topology for trees with reduced heights can both help addressing this problem and reduce as well as homogenize streaming delays. Furthermore, direct dependencies, where a few nodes have significantly more children than other, can be reduced by improving the balancing of flow trees. While such modifications help improving the system structure in general, they are expected to specifically improve the system performance under flash crowds In these challenging scenarios, the system is put under extreme stress, amplifying the impact of weaknesses in the topology.

For this reason, TOPT newly introduces the previously presented framework for topology optimizations. As shown in [2], to optimize a given topology for an overall minimum streaming delay across all peers and robustness of the structure, global knowledge would be required, which is not available and not desired in a fully distributed environment. Inspired by [2], [13], [27], the topology optimizations presented here are local optimizations. For most cases, decentralization implies that it cannot be guaranteed that optimizations ultimately lead to globally optimal flow topologies. Yet, it is expected that in most cases the optimizations can lead to near optimal flow structures as long as peers can pick optimization partners not only from their local flow partners but a random set of peers within the system. Therefore, in TOPT peers can pick optimization partners from their complete set of neighbors to avoid ending up in local minima for the flow structures.

The topology optimization types adopted are inspired by mTREEBONE [27], where a single tree topology is built to improve the stream delivery between stable peers only. Non-stable peers are only connected to the tree structure as outskirts and, thus do not belong to the tree topology. While in TRANSIT the definition of tree structures and flows follows a different philosophy, the decentralized topology optimizations proposed by the authors of mTREEBONE seemed promising to be adapted for TOPT and used with the topology optimization framework. Other optimizations can be easily added using the proposed framework. Yet, it is important to note that the framework itself does not guarantee the convergence to an optimal topology. This solely depends on the optimizations themselves and their combinations.

The two applied optimizations from mTREEBONE are named *low-delay-jump* (LDJ) and *high-degree-preemption* (HDP). LDJ uses the locally available information on its neighborhood to jump to a node higher up in the tree (closer to the source), given this node has a free upload slot to accept the node as new child. This results in free slots closer to the source gradually being filled, reducing the overall height of the delivery trees. HDP checks if the local node currently serves more children than any other node in its neighborhood that is closer to the source. If this is the case, the node tries to issue a switch of positions with this neighbor. This way, nodes with more children gradually move closer to the source, leading to a denser and overall more balanced structure. These two rather simple optimizations can be easily realized using the described primitive topology modifications provided by TOPT. Even more, using the presented seamless handover mechanism, they can be used without degradations in the stream delivery process, which is an important requirement that was not considered by the authors of mTREEBONE. The optimizations were adopted and extended to be used on the multi-tree flow topology of TOPT by allowing independent optimizations of the flow trees. Optimizations of individual flows can coexist, avoiding that optimizations degrade other active flows. A side effect of this approach is that it automatically introduces a diversification of the individual flow structures, as they are optimized independently from each other.

Using the topology optimization framework, peers can freely choose optimization types to be applied and even enable or disable individual types, depending on the current situation, goal, and preferences. To compare possible optimization instances of the same type, the concept of *gain* is introduced. For the LDJ optimization, which aims at allowing a node A to jump to another node B closer to the source (assuming free capacity), the *gain* is defined in the following manner:

$$\text{gain}(\text{LDJ}_{A,B}) = d(A) - d(B), \quad (1)$$

where $d(N)$ denotes the distance in hops of node N to the source in the respective flow tree. Intuitively, the *gain* is higher if the jump towards the source is larger. Neighbors without free upload capacity are not considered for a jump. Jumps to neighbors with higher depths than the local node would result in a negative *gain* and, consequently, are also not considered.

Analogously, the *gain* for the HDP optimization is defined as follows. In this optimization, node A chooses to preempt the position of a node B. This is only done if B has fewer children than A and B is closer to the source than A. Otherwise, B is not considered for optimization. Both factors are taken into account for calculating the *gain* for this optimization:

$$\text{gain}(\text{HDP}_{A,B}) = \alpha(d(A) - d(B)) + \beta(c(A) - c(B)), \quad (2)$$

where $d(N)$ denotes the depth of node N in the tree, $c(N)$ denotes its number of children, and α, $\beta \in \mathbb{R}^+$ represent stress factors. These factors may be varied in order to influence the impact of node depths compared to the number of children. By default, both factors are considered to be equally important (α, $\beta = 1$). In future work, it is planned to extend this approach for more complex topology optimizations and *gain* definitions.

C. The Role of the Tracker

A newly joining peer first requests a list of initial neighbors from the tracker. The peer then contacts some or all of these initially provided neighbors in order to open incoming connections, establish flows, or issue requests for individual missing blocks. In addition to the above presented modifications at peer side, the tracker of TOPT includes extensions to improve the selection of adequate initial peers, take special peer roles into account, and to employ a set of special join mechanisms for improved flash crowd support. For this purpose three new components were introduced to the tracker as depicted in Figure 2: the *Head Node Manager*, the *Batch Join Manager*, and the *Flash Crowd Detector*.

The *Head Node Manager* manages only the nodes that are directly connected to the streaming source. As mentioned in [2], they are highly important as they form the first delivery hop for the diffusion process. These so called *head nodes*, short *heads*, can impose severe content bottlenecks to the delivery process if, for example, they themselves are not able to serve a sufficiently large number of other peers. Keeping track of head nodes showed to be promising to ensure a stable streaming process. To enable a fast diffusion of new blocks, the upload capacity of the heads should be fully utilized. This includes avoiding cases where a head does not forward any flow at all, as observed in the initial TRANSIT topologies. To this end, a mechanism was designed where head nodes periodically inform the tracker (i.e. the *Head Node Manager*) about available upload capacities and their flow forwarding status. If heads with free capacities or a low number of forwarded flows exist, newly joining peers are provided with an initial neighbor list that exclusively includes a set of these head nodes to enforce new peers to connect to them with a high probability. In the unlikely case that the peer cannot connect to any of those heads, it will still receive a list of the heads' neighbors during the connection setup phase, enabling it to immediately connect to other peers.

The *Batch Join Manager* can influence and delay the actual attachment of peers to the active overlay in case of a flash crowd. This is done to allow for *batch joins* [5], [30] as introduced in Section III, where multiple peers are attached to the topology as a group. For this purpose, peers within a batch are provided with an initial neighborhood defined by the *Batch Join Manager*. This way, the topology of a batch can be influenced to some extend through the subset of neighbors provided to peers in the batch. By providing peers with a pre-selected set of neighbors within the batch, a balanced tree with high-capacity inner nodes and low-capacity leafs can be formed as proposed in [5]. As long as none of the peers knows

any active peer of the streaming system, no video blocks are available within the batch. As soon as the *Batch Join Manager* decides that a batch is complete, it provides at least one of the batch peers with an active contact with free capacities. The peer then contacts this active peer, establishes an incoming connection, and starts requesting video data. In best case, a set of flows is quickly established among the peers of the batch, allowing the whole batch to be served smoothly. In the simplest case, only one active peer within the overlay is involved in the attachment of the complete batch, reducing the competition between new peers for upload resources and allowing new peers to immediately contribute their upload bandwidth to the system. In its basic form, this batch joining process was proposed by Chung et al. [5] and recently extended by Wu et al. [30] to form multi-tree batches. In this work, the approach was adapted and extended for the specifics of the hybrid nature of TRANSIT and its connection handling. Thereby, a single capacity-based tree is constructed by default and the root node is connected to the active topology once the batch completes. Two variants are studied, one where the *Batch Join Manager* waits for a certain number of peers to complete a batch and another where the batch is attached after a fixed period of time. The first is expected to allow forming better batch topologies, while the second helps assuring an upper bound for the startup delays of the involved peers. The general concept is considered highly promising to also study other topology variants to be used within batches that could have desired properties for flash crowds. Therefore it is planned to further investigate this direction in future work.

The *Flash Crowd Detector*, the third newly introduced tracker component, monitors the peer arrival rate and activates the *Batch Join Manager* once a certain threshold is exceeded. It could also use other sources of information, such as expert knowledge or information about scheduled events that are expected to cause flash crowds.

V. EVALUATION

The goal of the evaluation was, first, to study the impact of the individual presented mechanisms and extensions under flash crowds when applied to a hybrid streaming system. Second, the combination of all mechanisms in form of the new integrated streaming system TOPT is compared to the state-of-the-art systems TRANSIT [28] and MTREEBONE [27]. Due to space constraints, only the second part is presented here. For the first part as well as additional evaluation results, the interested reader is referred to [22].

A. Methodology and Scenarios

Due to the necessity for large number of clients and controlling the course of the experiments to achieve the desired flash crowd characteristics, simulations showed to be the best and most feasible method for the evaluation of the proposed mechanisms. For this reason, the original implementation of TRANSIT was extended and used with the discrete event-based simulation framework PeerfactSim.KOM [25]. All simulations were executed with a total number of 5000 available clients, divided into three resource classes based on the node bandwidth distribution from [19] (cf. Table I).

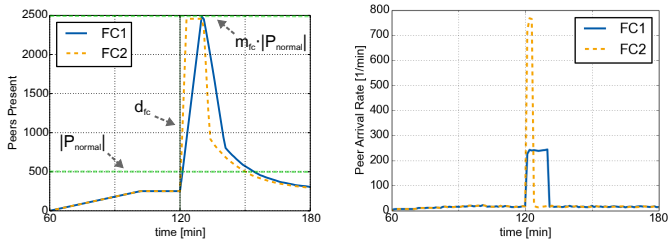
The upload bandwidth of the class *Medium* was increased to $6,150$ Kbps to allow for a fair comparison of TOPT to

TABLE I. USED PEER BANDWIDTH DISTRIBUTION BASED ON [19].

Class	Number of peers	Share	Upload bandwidth (Kbps)	Download bandwidth (Kbps)
Low	$2,800$	56%	$2,253$	$15,326$
Medium	$1,500$	30%	$6,150$ ($3,150$)	$53,665$
High	700	14%	$52,665$	$96,421$

TRANSIT [28] and MTREEBONE [27]. Simulations showed that MTREEBONE is highly unstable with the original setting. A closer investigation revealed that nodes forming the stable backbone of the overlay, the so called treebone nodes, require higher bandwidths to establish a stable tree topology. Overall, the scenario is intentionally relaxed from the upload bandwidth perspective as this work focuses on the challenge of timely integration of upload resources of arriving peers, not particularly on the efficient resource utilization of the system in more stable environments. Here, the available resources to serve new peers is limited by the streaming mechanism itself [16], justifying the choice for more realistic resource configurations.

Concerning the comparison to MTREEBONE, it is important to note that the system was proposed for single-layer video streaming, while TRANSIT and TOPT support heterogeneous scenarios by allowing peers to stream lower-quality substreams dependent on the available resources. Again, to allow for a fair comparison, peers were configured to stream the layered H.264/SVC [24] stream in its full quality, i.e. all of its quality layers, achieving a single-layer streaming behavior with the same bandwidth requirements as MTREEBONE. This way, the quality-adaptive behavior of the two other systems is intentionally disabled to purely focus on the mechanisms under study and avoid undesired cross effects. Peers are configured to temporarily pause playback (referred to as *stalling*) when video blocks required for playback were not delivered in time. The delivered video stream consists of four SVC layers with an accumulated maximum bitrate of 900 Kbps. For each layer, TRANSIT and TOPT strive to build up a maximum of four distinct flows. A single source node was used with an upload bandwidth of 12 Mbps, allowing the server to serve a maximum of 13 peers with the complete video stream. All simulations were repeated six times and 95% confidence intervals are shown for all average values. The core TRANSIT and TOPT system parameters are similar to the ones used in [28] and are not listed here due to space constraints.



(a) Number of online peers. (b) Arrival rate.

Fig. 4. Evolution of online peers and arrival rates for $FC1$ and $FC2$.

To show the effects of flash crowds, workloads were defined that in particular focus on both a steep increase and a rapid reduction of connected peers in a very small time frame

as observed for real streaming systems [11], [26]. The used workload models show similar phases of a flash crowd as observed in [1] for web traffic. At the beginning, peers enter the system with a moderate arrival rate up to a maximum number of $|P_{\text{normal}}| = 250$ connected peers. At this point, new peers only enter the system to replace peers for which sessions ended. The session lengths are modeled using the distribution derived by Vu et al. [26] (parameters: $a = 1.079$, $b = -0.09594$) from a large-scale measurement of the PPLIVE system. After a stabilization period, giving the overlay time to reach a stable state, the actual flash crowd starts at time $t_{\text{fc}} = 2$ h, with the number of active peers being linearly increased up to a threshold of $m_{\text{fc}} \cdot |P_{\text{normal}}|$ within a time frame d_{fc}. This intends to model the characteristics of a typical broadcast channel, where a small number of users is already watching the stream before the flash crowd occurs. To model different intensities of flash crowds, d_{fc} was stepwise shortened to achieve an increased arrival rate and thereby shock level of the flash crowd [1], while keeping the maximum number of concurrent users fixed with $m_{\text{fc}} = 10$. In the following, results for the two steepest studied flash crowd workloads $FC1$ ($d_{\text{fc}} = 10$ min) and $FC2$ ($d_{\text{fc}} = 3$ min) are presented. After reaching the peak of the flash crowd, the number of connected peers converges back to $|P_{\text{normal}}|$ as peers that reach their end of session are not replaced by new peers until the normal level is reached again. The evolution of the number of peers and the arrival rates for the two workloads are shown in Figure 4.

B. TOPT: *Impact of Combining the Optimizations*

To understand the impact of the different topology-related optimizations during flash crowds, they were evaluated individually as extensions to the TRANSIT system. Due to space constraints, the results for these steps are not presented here but are discussed in [22]. In the following, the most important step of the evaluation is presented, the combination of all individual optimizations and the study of their combined impact on streaming performance as well as communication overhead. This combination constitutes the newly proposed TOPT approach and includes the topology optimization framework, both described optimization types, the head optimizations, as well as the time-based batch join mechanism.

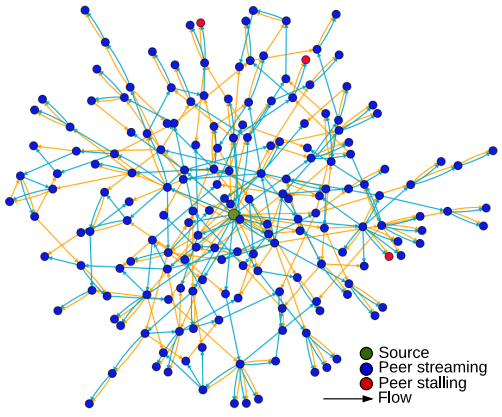
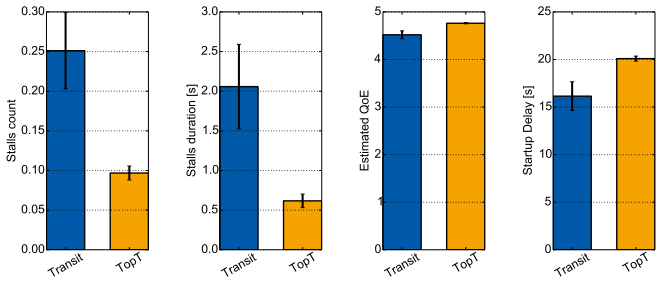


Fig. 5. Example topology for two flows as established by TOPT.

Figure 5 shows an example topology for two flows as established by TOPT. The figure shows a snapshot of the topology shortly after the start of a flash crowd for a very small test scenario. Despite its size, its structure shows major differences compared to the TRANSIT topology shown earlier. The individual flow trees are less aligned and more diverse. Besides, the individual trees are more balanced with less chain-like dependencies in the core. In the following, some of these observations are supported using quantitative results for workload $FC2$. First, TOPT is compared to the original TRANSIT [28]. Second, both are compared to MTREEBONE [27], the streaming approach that inspired the topology optimization of TOPT. The simulation model of MTREEBONE was implemented from scratch (cf. [15] for details), primarily based on the descriptions in [27]. This step was necessary as the original implementation could not be shared by the authors.



(a) Stalls count (b) Stalls duration (c) Estim. QoE (d) Startup delay

Fig. 6. Avg. streaming performance TRANSIT vs. TOPT (workload: $FC2$).

For a better understanding, in the following, key results for TOPT and TRANSIT are compared for the $FC2$ workload. Figure 6 depicts the average streaming performance results. The average stalls count per peer and minute was reduced by 60% from 0.25 to 0.1 and the average stalls duration by 70% from 2.06 s to 0.62 s. This also results in an improved average estimated Quality of Experience (QoE) from a value of 4.52 to 4.76 on the MOS scale. The latter was calculated based on the stalling metrics and using an established model by Hoßfeld et al. [12]. As a result of the used batch join mechanism, furthermore, the startup delay is increased by 24% from 16.15 s to 20.09 s, which is considered acceptable given the major improvement of the other performance metrics.

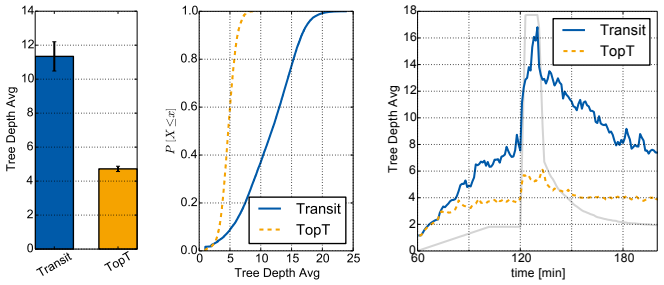
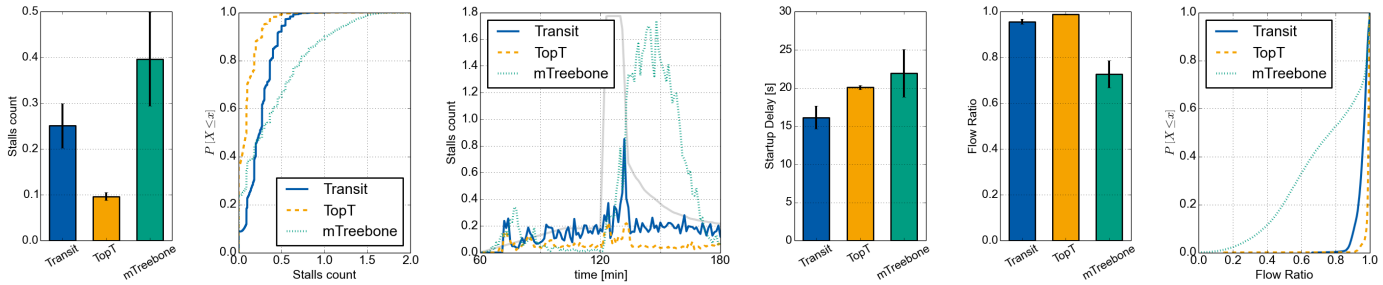


Fig. 7. Average flow tree depths: TRANSIT vs. TOPT (workload: $FC2$).

The explanation for these major improvements in the streaming performance can be found in the flow topology structures of TOPT. Figure 7 exemplary shows the average flow tree depths as one of the studied topology metrics. It is calculated per flow and averaged over all four flows. For the analysis also other topology metrics were used, such as



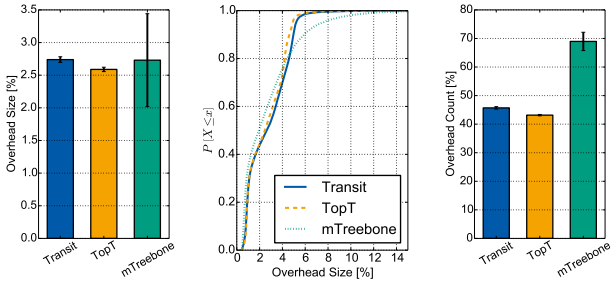
(a) Avg. and distribution of stalls count. (b) Avg. startup delay. (c) Avg. and distribution of flow ratio.

Fig. 8. Streaming performance for TRANSIT, TOPT, and MTREEBONE (workload: $FC2$).

minimum and maximum flow depths. They are not shown here due to space constraints but support the effects observable for the average depths, which was reduced by 58% from 11.34 to 4.72 with also a lower variance as visible in the distribution of the average tree depth. While TRANSIT shows a drastic increase of tree depths during the flash crowd to more than 16, TOPT manages to keep it below a value of 6 even under heavy stress. In addition, it can be observed that, for all four metrics, the confidence intervals are smaller for TOPT. This shows that the new system is able to provide more stable and reliable average results over the different simulation runs. Random effects that seem to greatly influence the structure of the topologies are reduced with the proposed extensions.

Figure 8 shows the key results of the streaming performance comparison of all three systems. The average stall count (0.397) and stalls duration (11.16 s, not shown here) of MTREEBONE are significantly higher than the values for the two other systems already presented earlier in this section. The time plot reveals the reason for this. While the topology of MTREEBONE very well supports the sudden increase in the number of peers (workload shown as grey curve in background), it performs badly when a large number of peers leaves the system. Similar observations can be made for the stalling duration (not shown here). Figure 8c shows the average flow ratio, which describes the share of video blocks delivered over flow structures. As all three systems are hybrid systems, broken tree structures lead to the remaining blocks being streamed using a less efficient mesh/pull mechanisms. While the two other systems quickly rebuild broken flow structures, MTREEBONE needs more time for this as leaving tree peers first have to be replaced by newly promoted stable peers. For fairness, the threshold after which a peer is promoted to be stable was configured in a manner to allow MTREEBONE to successfully build up a stable tree structure before the flash crowd period starts. Yet, the system in its version described in [27] seems not to be able to deal with the sudden leave of peers. Besides, MTREEBONE shows an average startup delay that is not significantly different to the one of TOPT.

Finally, Figure 9 compares the communication overhead as imposed by the three different approaches. It is defined as the share of traffic used for control messages in relation to the overall streaming traffic of a peer. Here, two alternative views are possible: one based on the number of messages (cf. Figure 9a) and the other based on the size of messages (cf. Figure 9b). This distinction is done to account for the fact that control messages usually make up for only a small fraction of the traffic volume due to the large size of the video streams. The average relative overhead size for all three systems is on a similar level (TRANSIT: 2.74%, TOPT: 2.59%, and MTREEBONE: 2.73%). For the average relative overhead count, MTREEBONE (68.97%) shows a 34% higher count than TRANSIT (45.69%), with TOPT (43.16%) again showing an even lower overhead than both. Interesting is the fact that TOPT in both cases actually shows a significantly lower overhead than TRANSIT. This was not expected as TOPT introduced additional communication for topology optimizations as described in Section IV. It seems that the more stable topologies achieved, compensate for the additional overhead as flows after optimization exist longer and thus overall less control communication between peers is required.



(a) Avg. and distribution of bandwidth overhead. (b) Message overhead.

Fig. 9. Overhead TRANSIT, TOPT, and MTREEBONE (workload: $FC2$).

VI. CONCLUSION

To sum up the results of the presented study, it can be said that in the field of flash crowd support for P2P live streaming systems, a number of different approaches have been proposed in the last years. Yet, the question remained if and how these approaches can be adopted for hybrid streaming systems. Therefore, in this paper and an extended version of this work [22], a number of different approaches have been studied, adapted and extended for this purpose. Their individual adoption showed promising results that motivated the proposal of an extended streaming approach called TOPT.

By combining the different presented optimizations, the hope was to achieve an even higher performance improvement as for the individual parts alone. Indeed, the combination

achieved a 60-70% improved streaming performance, translating to an average estimated QoE improvement to a value of 4.76 (excellent) on the MOS scale. These improvements are seen as a direct result of the significantly reduced flow tree depths by almost 60%, compared to the original TRANSIT system. On the downside, the startup delay increased by 24% or about 4 s as a result of the applied batch join mechanisms. Yet, this is considered acceptable, given the major improvements of the other performance metrics. A comparison to MTREEBONE showed that TOPT outperforms both reference systems in all aspects, where MTREEBONE shows a rather stable performance at the beginning of the flash crowds but collapses when a large number of peers leave the system. A surprising result showed to be the observations for the communication overhead of the three systems. While for the size all three performed at an equally low level below 2.8% of the video traffic, for the overhead count, MTREEBONE performed worst, while TRANSIT and TOPT are at an almost similar level. Here, TOPT performed slightly but significantly better than TRANSIT, which was not expected because of the additional communication it introduces. It is assumed that the overall more stable topologies lead to a more efficient streaming process compensating the additional communication over time. Overall, the results show great potential to further study topology optimization mechanisms to improve the support for flash crowd scenarios in hybrid P2P live streaming systems.

ACKNOWLEDGMENT

This work has been funded in parts by the European Union (FP7/#317846, SmartenIT and FP7/#318398, eCOUSIN) and the German Research Foundation (DFG) as part of projects C02 and C03 within the Collaborative Research Center (CRC) 1053 – MAKI. The authors would like to thank Matthias Wichtlhuber for his valuable input and feedback.

REFERENCES

[1] I. Ari, B. Hong, E. Miller, S. Brandt, and D. Long, "Managing Flash Crowds on the Internet," in *IEEE/ACM MASCOTS*, 2003.

[2] M. Brinkmeier, G. Schafer, and T. Strufe, "Optimally DoS Resistant P2P Topologies for Live Multimedia Streaming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, 2009.

[3] Y. Chen, B. Zhang, and C. Chen, "Modeling and Performance Analysis of P2P Live Streaming Systems under Flash Crowds," in *IEEE ICC*, 2011.

[4] Z. Chen, B. Li, G. Y. Keung, H. Yin, C. Lin, and Y. Wang, "How Scalable Could P2P Live Media Streaming System Be with the Stringent Time Constraint?" in *IEEE ICC*, 2009.

[5] T.-Y. Chung and O. Lin, "A Batch Join Scheme for Flash Crowd Reduction in IPTV Systems," in *IEEE ICPADS*, 2011.

[6] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2013 – 2018," Tech. Rep., 2014.

[7] Die Medienanstalten, "Digitisation 2013 - Broadcasting and the Internet - Thesis, Antithesis, Synthesis?" Tech. Rep., 2013.

[8] R. Dongni, Y.-T. Li, and S.-H. Chan, "On Reducing Mesh Delay for Peer-to-Peer Live Streaming," in *IEEE INFOCOM*, 2008.

[9] ——, "Fast-Mesh: A Low-Delay High-Bandwidth Mesh for Peer-to-Peer Live Streaming," *IEEE Transactions on Multimedia*, vol. 11, no. 8, 2009.

[10] Ericsson ConsumerLab, "TV and Media 2014 - Changing Consumer Needs are Creating a New Media Landscape," Tech. Rep., 2014.

[11] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, vol. 9, no. 8, 2007.

[12] T. Hoßfeld, R. Schatz, E. Biersack, and L. Plissonneau, "Internet Video Delivery in YouTube: From Traffic Measurements to Quality of Experience," in *Data Traffic Monitoring and Analysis*. Springer, 2013.

[13] L. Krumov, A. Andreeva, and T. Strufe, "Resilient Peer-to-Peer Live Streaming using Motifs," in *IEEE WoWMoM*, 2010.

[14] B. Li, G. Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin, "An Empirical Study of Flash Crowd Dynamics in a P2P-Based Live Video Streaming System," in *IEEE GLOBECOM*, 2008.

[15] E. Lidanski and J. Rückert, "Implementing a P2P Live Streaming Overlay for PeerfactSim.KOM," Peer-to-Peer Systems Engineering Lab, TU Darmstadt, Germany, Tech. Rep., 2014, http://www.ps.tu-darmstadt.de/publications/PS-TR-2014-01.pdf.

[16] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao, "Flash Crowd in P2P Live Streaming Systems: Fundamental Characteristics and Design Implications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 7, 2012.

[17] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-driven Mesh-based Streaming," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, 2009.

[18] E. Nygren, R. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-performance Internet Applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, 2010.

[19] Organisation for Economic Co-operation and Development, "OECD Broadband Report," Tech. Rep., 2012.

[20] R. Rejaie and N. Magharei, "On Performance Evaluation of Swarm-based Live Peer-to-Peer Streaming Applications," *Multimedia Systems*, vol. 20, no. 4, 2014.

[21] B. Richerzhagen, "Supporting Transitions in Peer-to-Peer Video Streaming," November 2012, Master's Thesis, Peer-to-Peer Systems Engineering Lab, TU Darmstadt, Germany. [Online]. Available: http://www.ps.tu-darmstadt.de/fileadmin/publications/Ric12.pdf

[22] J. Rückert, B. Richerzhagen, E. Lidanski, R. Steinmetz, and D. Hausheer, "Optimizing Flow Topologies for Flash Crowd Support in Hybrid Peer-to-Peer Live Streaming," Peer-to-Peer Systems Engineering Lab, TU Darmstadt, Germany, Tech. Rep., 2014, http://www.ps.tu-darmstadt.de/publications/PS-TR-2014-03.pdf.

[23] Sandvine, "Fall 2014 Global Internet Phenomena Report," 2014.

[24] H. Schwarz and M. Wien, "The Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Signal Processing Magazine*, vol. 25, no. 2, 2008.

[25] D. Stingl, C. Gross, J. Rückert, L. Nobach, A. Kovacevic, and R. Steinmetz, "PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems," in *IEEE HPCS*, 2011.

[26] L. Vu, I. Gupta, K. Nahrstedt, and J. Liang, "Understanding Overlay Characteristics of a Large-Scale Peer-to-Peer IPTV System," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 6, no. 4, 2010.

[27] F. Wang, Y. Xiong, and J. Liu, "mTreebone: A Collaborative Tree-Mesh Overlay Network for Multicast Video Streaming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 3, March 2010.

[28] M. Wichtlhuber, B. Richerzhagen, J. Rückert, and D. Hausheer, "TRANSIT: Supporting Transitions in Peer-to-Peer Live Video Streaming," in *IFIP NETWORKING*, 2014.

[29] H. Wu, J. Liu, H. Jiang, Y. Sun, J. Li, and Z. Li, "Bandwidth-aware Peer Selection for P2P Live Streaming Systems under Flash Crowds," in *IEEE IPCCC*, 2012.

[30] H. Wu, K. Xu, M. Zhou, A. Wong, J. Li, and Z. Li, "Multiple-tree Topology Construction Scheme for P2P Live Streaming Systems under Flash Crowds," in *IEEE WCNC*, 2013.

[31] Y. Ye and P. Andrivon, "The Scalable Extensions of HEVC for Ultra-High-Definition Video Delivery," *IEEE MultiMedia*, vol. 21, no. 3, 2014.

[32] X. Zhang and H. Hassanein, "A Survey of Peer-to-Peer Live Video Streaming Schemes - An Algorithmic Perspective," *Computer Networks*, vol. 56, no. 15, 2012.

[33] M. Zhao, A. Chen, Y. Lin, and A. Haeberlen, "Peer-Assisted Content Distribution in Akamai NetSession," in *ACM IMC*, 2013.