Christopher Probst, Andreas Disterhöft, and Kalman Graffi:
Chunked-Swarm: Divide and Conquer for Real-time Bounds in Video Streaming
In: NEW2AN'15: In Proc. of the IEEE International Conference on Next Generation Wired/Wireless Networking, 2015.

Chunked-Swarm: Divide and Conquer for Real-time Bounds in Video Streaming

Christopher Probst, Andreas Disterhöft, and Kalman Graffi

Technology of Social Networks, University of Düsseldorf, Germany
christopher.probst@hhu.de,disterhoeft@hhu.de,graffi@hhu.de
http://www.tsn.hhu.de/en.html

Abstract. Live user-generated video streaming platforms like Twitch.tv generate a large portion of the Internet traffic. Millions of viewers daily watch user channels, although roughly 85 % of all channels have less than 200 views during one session. Due to latency, Twitch.tv provides one or more servers for each of Twitch.tv's supported countries. An alternative approach could enable peer-to-peer communication in order to utilize the capacities of the user devices. Solutions up to now, mainly offer only best effort delay guarantees on the distribution speed from initial seeders to all peers. In this paper, we present Chunked-Swarm, a swarm-based approach, which aims to offer predictable streaming delays, independently of the number of peers. Evaluation shows the various impact of the number of peers, number of video parts and chunks on the streaming delay. Being able to hold specific deadlines for up to 200 peers, predestines our solution to be suitable for the majority of Twitch.tv's channels.

Keywords: Real-time Streaming, Twitch.tv, P2P Swarm, QoS

1 Introduction

Traditional video streaming applications are mostly based on the client / server model, where the server only responds to client requests and the clients do not know each other. In case of real-time video streaming, content delivery deadlines are crucial to meet in order to provide a satisfying quality of experience. Naturally, single servers or server clusters are limited in their bandwidth and thus application providers have to upgrade their servers in order to cope with increasing bandwidth requirements leading to high operational costs. Instead of continuously upgrading servers, unutilized resources of their clients could be used to improve this imbalance.

Twitch.tv is a good example for a growing streaming application for user-generated content with hundreds of thousands of streamers and millions of spectators requiring high operational investments. Other examples are the streaming of live-lectures in universities, which are becoming increasingly important as students are used to online resources and e-learning [2]. In case of Twitch.tv, Zhang and Liu present in their measurement study [20] a model for the distribution from viewer to streamer using a Weibull and / or Gamma function. The measurements show that around 85 % of all streamers have less than 200 viewers

This work was partially supported by the German Research Foundation (DFG) Grant "OverlayMeter" (GR 4498/1-1)

The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, not withstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

during their live streaming session. Although viewers are free to start and to switch to other channels, Nascimento et al. [12] modeled the behavior of viewers in Twitch.tv and concluded that 'the content is mainly consumed by a small fraction of very assiduous streamers'.

Based on these observations of these stream characteristics, namely being user-generated, having mainly less than 200 viewers and very low churn, we propose our fully decentralized streaming solution in which the viewers use their own upload capacity to help the streamer to distribute streaming content within a given transmission deadline. In specific, we present a peer-to-peer (p2p) system which provides guarantees on the delivery time and performs best on networks with up to 200 peers. All peers (viewers) predictably receive continuous video parts, while the seeder (streamer) uploads parts only once (in case there is no churn). Our solution, in contrast to solutions from literature, emphasizes on the real-time requirements of the content distribution and meets specific distribution deadlines to achieve a certain level of quality of service (QoS).

In Section 2 we discuss related work and point out that our approach in aiming at clear content delivery timings is in contrast to the majority of solutions in literature. In Section 3, we elaborate the theory and concept of our approach. In this section, we also model the transmission time using our approach. We conduct an extensive simulation study of the proposed system. Section 4 introduces our evaluation setup and the obtained results. In specific, we highlight four scenarios that we used to analyze the impact of the number of peers, the number of chunks and number of parts on the performance of our content delivery strategy and its overhead. Evaluation shows that the desired real-time characteristic is reached. In Section 5, we conclude our work.

2 Related Work

In this paper, we propose a novel real-time streaming protocol for p2p networks, which provides delivery delay guarantees in relation to the slowest node's bandwidth. Therefore, we review related work in the field of p2p (live) video streaming and highlight first, that this desired guarantee on the delivery delay is novel. Three main solution categories have emerged for live video streaming: content delivery networks, p2p networks and IP multicast. Content delivery networks assist in the delivery of media content through a worldwide network of data centers. IP multicast is limited to a few Internet Service Providers (ISP) only and typically is not reaching beyond their networks. P2P networks offer the option to stream video to a large number of users with small operational costs, thus taking a serious impact on the media industry [9]. The content delivery is mainly operated by the end devices of the users, sharing storage, computing time and bandwidth.

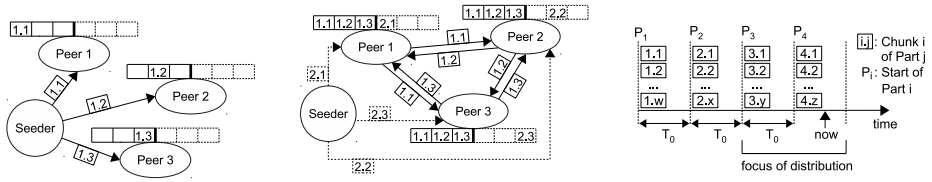
For p2p live video streaming three main solution classes have been identified: centralized schemes, clustering recursive schemes and full-distributed schemes, which are further sub-subcategorized in swarm-based schemes and tree-based schemes. Tree-based schemes can be further sub-categorized in structured, network-driven and data-driven. To ease the p2p exchange of parts, each part is split in chunks or pieces. Typical chunk sizes range from 8 KB, over 256 Kb, to 1 Mb.

In centralized approaches, a central element is coordinating the transmissions between the peers. In ALMI [15] a central element defines an explicit bi-directional fat-tree topology, which is optimized for low delays and high bandwidth. The information towards the root lists the position of the nodes in the tree and their free capacities. The information flow towards the leafs contains the video content. CoopNet [13] is a more flexible tree and uses a delay-based metric space, in which nodes are placed according to their delay distances to each other. In Graffi et al. [6], the DHT supports to match ideal transmissions between the peers based on a variety of node characteristics. Centralized approaches can reach a low delay at the costs of high load at the essential central element. Due to the tree structure of height $\mathcal{O}(\log(N))$ with N being the number of peers, the leafs have to wait $\mathcal{O}(\log(N))$ submission rounds in order to receive the full video part. A submission round is hereby the time needed to download the video part once. Here, we aim at a maximum of two submission rounds.

Clustered approaches organize nodes in multi-layered topologies. On each layer, the nodes are organized in clusters or mini-swarms in which they exchange data and elect a cluster-head which is representing them at the higher layer. While this layering scheme is similar to a tree structure, the topology is more complex as nodes on the same 'tree-level' also share data among each other. Examples for this are NICE [1] and THAG [17]. Due to the hierarchical layering, a tree of height $\mathcal{O}(\log(N))$ emerges which involves each chunk of the video to be transmitted at least $\mathcal{O}(\log(N))$ times. So, $\mathcal{O}(\log(N))$ submission rounds are needed to transfer the video part, we aim at two rounds.

The two main categories for fully distributed schemes are tree-based solutions and swarm-based solutions. Tree-based solutions aim at creating a tree structure which allows to share the video content from the seeding root to all nodes in a hierarchical manner. One option for trees is to rely on a distributed hash table, reusing the routing table or the capability to have responsible nodes for given IDs. Examples for this are SplitStream [3] and Bayeux [23]. Using a DHT involves besides the advantages also a considerable amount of maintenance overhead. Trees can be network-driven or data-driven. Network-driven trees are optimizing for having peers with high bandwidth capacities in high layers of the tree, and thus avoiding bottlenecks in the early steps of the video dissemination. Examples are mTreebone [18] and TreeClimber [21]. They support the dynamic rearrangement of nodes in the tree in order to support changes in the nodes' network conditions. Finally, data-driven tree-based solutions organize the nodes according to their current position in the video. Parent nodes are chosen in a way that their are ahead of the playback position of the considered node and thus are guaranteed to have the desired video chunks. As a results each layer in the tree is a time step behind in the video position than its parent peer. Examples are CoolStreaming [19], Substream Trading [10] and SPANC [4]. Also here, $\mathcal{O}(\log(N))$ submission rounds are needed to transfer the video part, while we aim at a maximum of two submission rounds.

Swarm-based schemes are the second large group of solution for p2p-based live video streaming. In these, peers exchange information on the chunks they



(a) Initial distribution of chunks (b) Parallel distribution of chunks in full mesh (c) Live video streaming: Distribution of max. 2 parts

Fig. 1. Chunked-Swarm Distribution

share with their neighbors and either push or pull then further chunks to complete their video part. Swarms use the network capacities of all nodes in contrast to several tree-based solutions which omit the upload potential of the leaf nodes. Examples are BTLive [16], Chainsaw [14] and LayerP2P [11]. Nodes exchange small bitmaps on their chunk offerings and use a small share set with typically four nodes to exchange individual chunks. While most large-scale p2p-based live video streaming solutions use a swarm-based scheme, they are reported [22] to have a playback delay between 5s and 20s, which allows a lost packet to be re-transmitted once or twice in case of errors. The playback delays are by this significantly longer than in tree-based schemes.

In this paper, we present a p2p-based live video streaming scheme which is fully-distributed, swarm- and pull-based. Our approach aims to distribute a video stream with a fixed and predictable start-up and streaming delay for a set of roughly 200 peers in a low churn environment like 85 % of Twitch.tv's streams. For the delay we aim at a maximum of twice the transmission time of the seeder to one peer, which is not supported by previous work due to their tree-based core with a minimum delay of $\mathcal{O}(\log(N))$ or swarm-based core with arbitrary delays between 5s and 20s. In contrast to other (swarm-based) p2p solutions, our approach manages that the seeder uploads the video part only once, thus even weak peers like smartphones are perfectly suitable to be a seeder. Next, we present our approach.

3 Our Approach: Chunked-Swarm

In Chunked-Swarm we use a full mesh topology, in which all peers (viewers) in the swarm are connected to each other, thus the suggested limit of 200 peers. The seeder (streamer) splits the video stream into parts, which are then split into chunks and announced to all connected peers, see Fig. 1(a). Therefore, the chunk count should always be equal to or greater than the number of peers. Peers request a random chunk from the seeder, but are rejected when requesting a chunk, which has already been requested by another peer. Those peers remove this chunk from their interest set and start a new request. This procedure is repeated until all peers have found an unique chunk to download. Thus, the seeder delegates the responsibility for distributing chunks to the peers. If a peer goes offline, the initial chunk received from the seeder might also leave the swarm. The seeder detects those leaving peers and announces the lost chunk again. After downloading a chunk from the seeder, peers announce their own chunk to all other peers in form of a space-efficient bitmap. After this step, each peer owns

a chunk, which is of interest for the other peers. This strategy gives every peer the ability to participate equally in the distribution of each part. Therefore, the seeder does not have to participate anymore in sharing, unless chunks are lost, e.g. due to churn in the rather short period of a part. In the following, all peers download in parallel from all other peers in a pull-based manner, see Fig. 1(b).

To inform peers about available chunks, an accumulative and space-efficient bitmap-state is announced regularly to all peers. While this involves $O(\log N^2)$ messages in the network, for small and medium sized swarms the overhead is manageable and the start-up delay is below $2 * T_0$, where T_0 is the time needed to transfer an entire video part from the seeder to the slowest peer. This limit is fixed and independent of the number of peers. Therefore, our solution can theoretically distribute a video part between an arbitrary number of peers in the same time as it would take a server, to serve only two clients. If a new peer connects to the seeder, it gets an address list of all other peers in the swarm to which it connects then. The joining peer only participates for the exchange of the next part, to not induce churn in the current part exchanging clique.

3.1 Analysis of the Transmission Time of One Part

Fig. 1(a) and 1(b) show an example of the Chunked-Swarm model with one seeder and three peers distributing two video parts. Each part is of size (s) and consists of three chunks, but for now, we only concentrate on the first part, whose chunks are visualized with solid arrows. For reasoning, we assume the seeder and the peers have the same upload and download bandwidth (b). Each of the remaining peers request one unique chunk of the first part, which is a third of the whole video part ($\frac{1}{3} * s$), from the seeder in parallel. Therefore, each peer also gets $\frac{1}{3}$ of the seeder's upload bandwidth, so it takes $3 * (\frac{1}{3} * s) * \frac{1}{b} = \frac{s}{b} = T_0$ seconds to upload all chunks once to the swarm, in specific one distinct chunk to each peer in parallel.

This is the same time span it would take a server to upload the video part to one client. At this point, the entire first video part is present in the swarm, where each peer has exactly one unique chunk ($\frac{1}{3}$ of the part). Therefore, the chunks can be distributed by the peers among themselves in parallel. Since each peer has to upload one third of the first video part ($\frac{1}{3} * s$) to 2 peers, it takes each peer $2 * (\frac{1}{3} * s) * \frac{1}{b} = \frac{2}{3} * T_0$ seconds, to upload its own chunk to the two other peers. Now, every peer contains the first video part after $T = T_0 + \frac{2}{3} * T_0$ seconds. Analogously, a variable number of peers (n) needs $T(n) = T_0 + \frac{n-1}{n} * T_0$ seconds to finish a single video part. Since $0 \leq \frac{n-1}{n} \leq 1$ is always true, T never exceeds $2 * T_0$. When the chunk count (c) is doubled, the model behaves better, as the peers can start to upload their own chunks while they are downloading the next unique chunk from the seeder. In theory, following formula applies:

$$T(n, c) = T_0 + \frac{n}{c} * \frac{n-1}{n} * T_0 = (1 + \frac{n-1}{c}) * T_0 < 2 * T_0 \tag{1}$$

with $c = a * n, a = 2^i \in \mathbb{N}_0$ as chunk count and n the peer count.

3.2 Live Video Streaming: Distribution of several Parts

So far, we have only discussed the distribution of a single video part, which can only be watched by viewers after completion. For live video streaming, we

use multiple video parts, each containing a consecutive time interval of the live stream. Naturally, the peers have to download and distribute the first video part completely, in order to start watching the live video stream continuously. The time needed to distribute the first video part across all peers, termed start-up streaming delay, is predictable and can be calculated using Equation 1. To provide a continuous flow of video parts, the seeder publishes the next video, while the peers distribute the last part. Therefore, the seeder and peers are never halted. Fig. 1(b) shows the collection and distribution phases, which run in parallel, after the first video part has been published.

To guarantee that multiple video parts do not interfere with each other, we evaluate the usage of 20 video parts in Scenario 4, which is presented in Section 4.4. Please note, that this scenario implements a form of video on-demand streaming, since the video parts remain in the network; late peers are able to watch the video stream from the beginning. However, as presented in Fig. 1(c), live video streaming only requires two video parts to be available at any given time. While the seeder is distributing the video part P_i, peers are busy distributing part P_{i-1} and finish their job before the seeder announces part P_{i+1}, since the $2 * T_0$ limit is respected. Older parts are not meant to be distributed, so they can be dropped by both, the seeder and peers. Therefore, the overhead caused by live video streaming based on Chunked-Swarm is very manageable. A realistic live video streaming use case, which also elaborates the relation between payload and announcements overhead, is presented in Section 4.6.

3.3 Churn

The time model in Section 3.1 disregards the influence of churn. A streaming start-up delay of $2 * T_0$, with T_0 being the time to transfer one video part, is only guaranteed if all peers participate in the chunk distribution. When peers leave during the transmission of a part, the time limit might be violated. In which extent depends on the number of concurrent leaving peers during the short time T_0. As T_0 is considered to be short, only very few peers are expected to leave within this time. Joining peers do not participate in the distribution of the current, but only of the next part. The seeder adjusts the number of chunks to the current number of peers, when a new video part is published.

4 Evaluation

An analysis of the efficiency of the Chunked-Swarm model with its live streaming option is given in Section 3.1. Here, we aim to stepwise evaluate the performance of live video streaming while looking at whether the $2 * T_0$ start-up delay limit is fulfilled in practice. The first step applies the distribution performance with only one video part and a variable number of viewers, while the second step evaluates the live video streaming with more than two parts. The second step is mandatory as two adjacent video parts may interfere with each other as mentioned in Section 3.2. We implemented and evaluated our approach in a standalone application, which is more accurate than the evaluation in a p2p simulator [5,8].

4.1 Evaluation Setup

We performed four scenarios in total using our Java implementation and compared their results and overhead caused by announcement messages. Each sce-

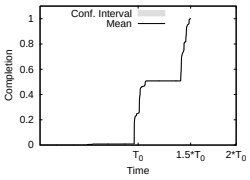
nario is run ten times. The results are merged by calculating the mean and the 95 % confidence interval. The first scenario is considered the default, in the remaining scenarios we vary one parameter at a time to measure the influence of this parameter. All scenarios ran in real-time on a server with 14 *Intel Xeon* 2.1 GHz cores and 38 GB main memory. For benchmarking, we reduced the upload bandwidth of each peer to 16 384 $\frac{byte}{sec}$, which might seem unrealistic. However, the upload bandwidth does not matter in our case, since we calculate the video part size according to the upload bandwidth and the desired start-up delay. This gives us the freedom to pick an arbitrary upload bandwidth for the evaluation, without being unrealistic.

4.2 Scenario 1: Default

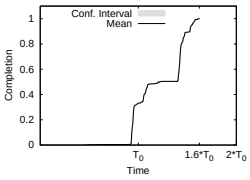
Scenario 1, also called default scenario, simulates 63 peers and 1 seeder owning only one video part, performing our first step towards live video stream. This scenario is used to evaluate the distribution of only one video part during a stream of user-generated content. The size (s) of the video part is calculated from the simulated upload bandwidth (b) of $16\,384 \frac{byte}{sec}$, such that a single transfer from the seeder to one peer takes exactly 10 minutes ($T_0 = 600\,sec$):

$s = 600\,sec * 16\,384 \frac{byte}{sec} = 9\,830\,400\,byte$. This implicates a target start-up delay of 20 minutes ($2 * T_0 = 1200\,sec$). While video streams should usually start within seconds, such a long start-up delay can help to evaluate the behavior of each peer during the start-up phase in more detail. Also, a higher upload bandwidth or smaller part would certainly decrease the start-up delay, but it is important to note, that the results gathered with a large start-up delay can easily be transferred to scenarios with a smaller start-up delay. The single video part is split into twice as many chunks as there are peers except the seeder, which makes $63 * 2 = 126$ chunks. Since the video part size is adjusted to the decreased upload bandwidth, the meta data size should be proportionally reduced as well. The meta data size represents the size of an announcement packet, transferred between all peers. In reality, the meta data would require approximately 64 *bytes*, as it contains a *SHA-1* hash and a bit set or differential vectors. If we assume, that a peer with an upload bandwidth of $1\,048\,576 \frac{byte}{sec} = 1 \frac{MiB}{sec}$ is realistic, our simulated upload bandwidth of $16\,384 \frac{byte}{sec}$ would be a 64× decrease. Therefore, the meta data size should be decreased by the same amount, which results in simulating a meta data size of 1 *byte*. While this value arguably might seem too low, only the proportion is important to gain meaningful results.

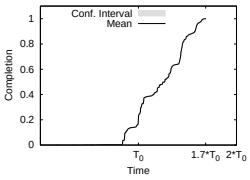
Fig. 2(a) shows the mean completion graph for each peer, where the x-axis represents the time and the y-axis the completion of the video part. After $1.5 * T_0$ seconds every peer has the entire video part available. In contrast, Equation 1 shows: $T(63, 126) = 1.49 * T_0$, which is almost equal to the measured duration. Fig. 2(b) shows the mean seeder upload bandwidth usage. For T_0 seconds, the seeder uploads at full speed after which it stops uploading, because the Chunked-Swarm model forbids uploading the same chunk twice. Figures 2(c) and 2(d) present the mean upload and download bandwidth usage of all remaining peers in average. Since there are twice as many chunks as peers, each peer can start uploading chunks after $0.5 * T_0$ seconds. It is important to note, that the seeder



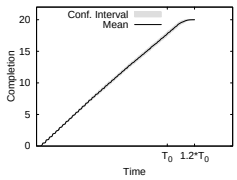
(a) Sc.1: Completion



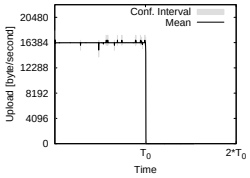
(a) Sc.2: Completion



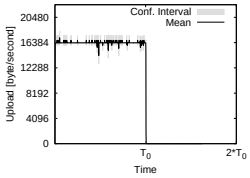
(a) Sc.3: Completion



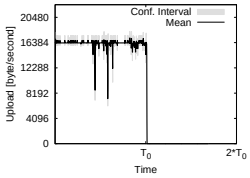
(a) Sc.4: Completion



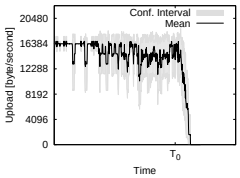
(b) Sc.1: Seeder Upload Bandwidth



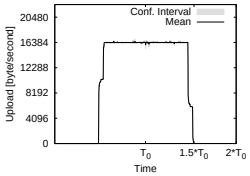
(b) Sc.2: Seeder Upload Bandwidth



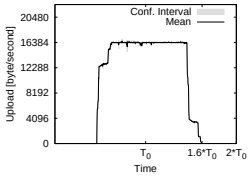
(b) Sc.3: Seeder Upload Bandwidth



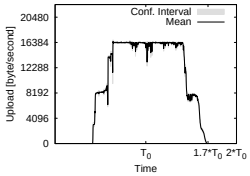
(b) Sc.4: Seeder Upload Bandwidth



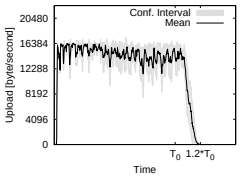
(c) Sc.1: Peer Upload Bandwidth



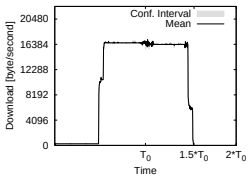
(c) Sc.2: Peer Upload Bandwidth



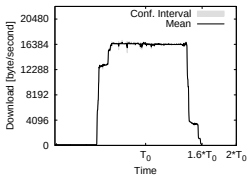
(c) Sc.3: Peer Upload Bandwidth



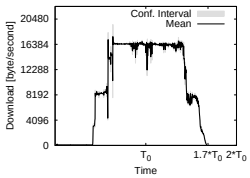
(c) Sc.4: Peer Upload Bandwidth



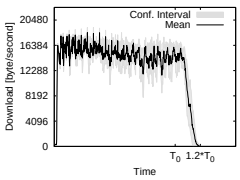
(d) Sc.1: Peer Download Bandwidth



(d) Sc.2: Peer Download Bandwidth



(d) Sc.3: Peer Download Bandwidth



(d) Sc.4: Peer Download Bandwidth

Fig. 2. Scenario 1 - 63 (+1) Peers

Fig. 3. Scenario 2 - 127 (+1) Peers

Fig. 4. Scenario 3 - 191 (+1) Peers

Fig. 5. Scenario 4 - 20 Parts

Scenario	*Payload [KiB]*	*Chunks in total*	*Size per chunk [KiB]*	*Overhead [KiB]*	*Overhead share [%]*
1 (63 peers)	9600	126	76.2	36.4	0.38
2 (127 peers)	9600	254	37.8	168.8	1.73
3 (191 peers)	9600	382	25.1	451.1	4.49
4 (63 peers)	9600	2520	3.8	1172.8	10.89
5 (31 peers)	9600	62	154.8	11.9	0.12
6 (255 peers)	9600	510	18.8	1212.4	11.2
7 (63 peers - chunk x4)	9600	252	38.1	79.1	0.8
8 (63 peers - chunk x8)	9600	504	19.0	153.6	1.6
9 (63 peers - chunk x16)	9600	1008	9.5	320.5	3.2

Table 1. Overhead tradeoff (download)

and the remaining peers upload in parallel after this time. The results show, that this model behaves as predicted and even undercuts the initial start-up delay limit of $2 * T_0$ seconds by far. The ensuing scenarios in Sections 4.3 and 4.4 modify the default scenario by using more peers and multiple video parts.

4.3 Scenario 2, 3: Higher Peer Count

These scenarios are used to observe the performance impact of using 127 and 191 peers, instead of 63 peers in a live video stream scenario with one part. There is still only one seeder. The results are shown in the vertically aligned Figures 3 and 4, which show a minor decrease in performance. While the default scenario takes about $1.5 * T_0$ seconds, these scenarios take $1.6 * T_0$ (with 127 peers) and $1.7 * T_0$ seconds (with 191 peers) respectively. Though both scenarios do not exceed $2 * T_0$ seconds, the performance drops when using more peers.

If a peer downloads its chunk too fast, it might download a second chunk, which should have been distributed by another peer. Thus it distributes two chunks instead of one, as shown in Fig. 3(c), 3(d) and 4(c), 4(d). Some peers start distributing their own chunks before $0.5 * T_0$ seconds, while others seem to upload chunks even after $1.5 * T_0$ seconds. While these small effects lead to an increased download time, the aim of $2 * T_0$ is met.

If n is the number of peers, the whole p2p network consists of n^2 connections. Simulating in real-time on one machine might induce this effect. If the peer count is doubled, the chunk count is doubled as well, so the number of announcements actually quadruples, because every peer notifies other peers about available chunks while downloading them. Theoretically, the Chunked-Swarm model works for any number of clients, in practice the overhead introduced by the growing number of chunks, and thus announcements, is just too high at some point. In reality, every announcement would also increase the latency caused by the *RTT (Round Trip Time)*. In the considered user-generated live streaming scenario, such as Twitch.tv, 85% of all streams have less than 200 users. We simulated up to 191 peers. The benchmark server actually runs in that case $191 * 192 = 36.672$ connections at once, which has a great impact on CPU and main memory usage. However, in reality the overhead is distributed evenly among all peers, so a single peer should be able to connect to more peers. We aim to prove the last statement with a setup of real peers in future work.

4.4 Scenario 4: Multiple Video Parts

In our second step towards live video streaming, which involves the distribution of two parts in parallel as shown in Fig. 1(c), we evaluate whether two adjacent video parts interfere with each other. For this reason we performed Scenario 4 which uses 20 video parts to multiply the effects of parallel part distributions. Please note, that this scenario represents more than a live video stream as the whole video, from the begin to the live time, is distributed by the peers. To maintain consistent notation, the meaning of $T_0 = 600s$ remains, now representing the time needed to distribute all video parts across all peers. Therefore, we introduce a new variable $T_{part} = 30s$, which is the time needed to transfer a single video part from the seeder to a peer and the rate at which the seeder publishes the video parts, so the streaming delay limit is now $2 * T_{part} = 60s$.

Fig. 5(a) shows the average video part completion graph of all peers. Interestingly, this graph almost represents a bisector, which indicates, that all peers finish each video part roughly at the same time. The reason is, that the download and the distribution phases of each video part ran in parallel. So the seeder starts uploading the next part while peers are still distributing the last part but they do not interfere with each other, because every peer is uploading one chunk to $n-1$ peers and downloading one chunk from $n-1$ peers plus one new chunk from the seeder. After an improved $1.2 * T_0$ seconds, each peer has all 20 parts available, although each video part still only has a chunk count factor of two as in the previous scenarios. This means that our delay limit of $2 * T_{part} = 60s$ has been undercut, thus the average delay for each video part is $\frac{1}{20} * 1.2 * T_0 = 36s$. The higher confidence intervals in Figures 5(b), 5(c) and 5(d) are caused by the large amount of chunks, which arise from using multiple video parts. When simulating ten runs, it is very unlikely, that all results are equal. As the confidence intervals show, it is still very likely that a further run will be similar. Therefore, we can conclude that a live video stream can be achieved by distributing and announcing only the latest two parts of the video stream. Next, we discuss the overhead in Section 4.5 and in Section 4.6 our results with a real live video streaming application in mind: Twitch.tv.

4.5 Overhead

Compared to other approaches, where the overhead partially depends on the actual payload size, the overhead of our Chunked-Swarm approach only depends on the number of chunks and peers. To fulfill the $2 * T_0$ start-up delay limit reliably, we recommend to use twice as many chunks as peers, so an increase of peers always causes an increase of chunks. The overhead of using more peers grows quadratic, while more chunks cause only a linear growth of overhead.

To reduce protocol overhead we use accumulated announcements, but we still encounter an asymptotically quadratic relationship. To determine the influence of using more peers and chunks more reliably, further scenarios were taken into account. The performance in these scenarios was not considered during the evaluation, as their results did not deliver more insight into the performance characteristics than the other four scenarios did.

Table 1 shows the overhead for these scenarios. When comparing Scenario 1 to 7, 8 and 9, where the chunk count was modified, the overhead seems to grow linearly, as shown in the *Overhead* column, which is what the model predicted. Scenarios 2, 3, 5 and 6 increase the peer count comparing to Scenario 1, thus the overhead should grow quadratically. Since the chunk count is increased as well due to the coupling with the peer count, additionally the overhead should be further increased. Interestingly, the results certainly show an exponentially growth, but it is not quite quadratic. This is due to the internal optimizations, so the quadratic growth can be seen as an upper limit, it is far less in practice.

4.6 Realistic live video streaming use case: Twitch.tv

To conclude, we transfer our work's results to the well-known live video streaming platform Twitch.tv and calculate the needed bandwidth of a streamer for a

certain start-up delay. To provide similar delays, as occurring in Twitch.tv [20], we choose a delay limit of 10s, so $2 * T_{part} = 10s$ and $T_{part} = 5s$. In a swarm of 191 nodes with one video part, 382 announcements (see Table 1) has been sent, resulting in 764 packets for two video parts distributed at the same time. The size of the 764 announcements (each 64 byte) with network headers (20+20+18 bytes for TCP+IP+Ethernet header) is $764 * (64 + 58)$ byte = 93.2 kByte.

The maximum supported bitrate streamers are allowed to stream via Twitch.tv is $3500\frac{kBit}{s} = 438\frac{kByte}{s}$ currently – as of 26th February 2015. Thus, viewers send each 5s traffic with $438 * 5 = 2190$ kByte video payload and 93.2 kByte announcement overhead, which results to 2283.2 kByte, thus our announcements introduce a 4.3 % overhead. Finally, all participants need an upload bandwidth of $\frac{2283.2*8}{5}\frac{kByte}{s} = 3.65\frac{MBit}{s}$, which is below the global broadband upload bandwidth with $10\frac{MBit}{s}$ (see Ookla – http://www.netindex.com/upload/). Thus the Twitch.tv scenario is well supported and our solution would allow private users to stream their content to up to 200 viewers with a small local bandwidth consumption and with reliable delay guarantees.

5 Conclusion

In this paper, we presented our approach for live video streaming, termed Chunked-Swarm. It uses a full mesh topology and a p2p chunk distribution algorithm that guarantees the distribution of the video to all connected peers under $2*T_0$, where T_0 is the time needed to transfer the desired data from the seeder to the slowest peer. If the number of chunks is chosen wisely, which is coupled with the number of peers, we are able to undercut the $2 * T_0$ mark, which is novel as related work introduces a delay of at least $\mathcal{O}(\log(N) * T_0)$. With a small modification, this model is also capable for streaming applications with a predictable start-up delay. In the evaluation we analyzed the implementation of our model, confirmed our expectations on the analysis of the distribution time mentioned in Equation 1. We evaluated the overhead and calculated a realistic use case. The injected overhead, caused by announcement messages, depends on the chunk count and also peer count, which are fixed for some distribution scenarios but may vary during a live video streaming. If the number of peers is doubled, the chunk count is also doubled and leads to an quadratic upper bound for the overhead. Nevertheless, the overhead share is still reasonable for file sharing and live video streaming applications for small and medium sized scenarios up to hundreds of nodes. As long as the transmission time aim is reached and the seeder has to upload the video only once, the overhead for the peers is considered bearable. Thus, we summarize that our solution is suitable for 85 % of Twitch.tv's streamers and would decrease the operational costs of Twitch.tv by utilizing unused resources of the spectators.

References

1. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *ACM SIGCOMM'02*, 2002.
2. S. Cardall, E. Krupat, and M. Ulrich. Live Lecture versus Video-recorded Lecture: Are Students Voting with Their Feet? *Academic Medicine*, 83(12), 2008.

3. M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Multicast in Cooperative Environments. In *ACM SOSP'03*.
4. T. K. Chan, S. G. Chan, and A. C. Begen. SPANC: Optimizing Scheduling Delay for Peer-to-Peer Live Streaming. *IEEE Transactions on Multimedia*, 12(7), 2010.
5. K. Graffi. PeerfactSim.KOM: A P2P System Simulator - Experiences and Lessons Learned. In *IEEE P2P'11*, 2011.
6. K. Graffi, S. Kaune, K. Pussep, A. Kovacevic, and R. Steinmetz. Load Balancing for Multimedia Streaming in Heterogeneous Peer-to-Peer Systems. In *ACM NOSSDAV'08*, 2008.
7. A. Kovacevic, K. Graffi, S. Kaune, C. Leng, and R. Steinmetz. Towards Benchmarking of Structured Peer-to-Peer Overlays for Network Virtual Environments. In *IEEE ICPADS'08*, 2008.
8. A. Kovacevic, S. Kaune, H. Heckel, A. Mink, K. Graffi, O. Heckmann, and R. Steinmetz. PeerfactSim.KOM - A Simulator for Large-Scale Peer-to-Peer Networks. Technical Report Tr-2006-06, TU Darmstadt, 2006.
9. N. Liebau, K. Pussep, K. Graffi, S. Kaune, E. Jahn, A. Beyer, and R. Steinmetz. The Impact Of The P2P Paradigm on the New Media Industries. In *AMCIS*, 2007.
10. Z. Liu, Y. Shen, K. Ross, S. Panwar, and Y. Wang. Substream Trading: Towards an open P2P Live Streaming System. In *IEEE ICNP'08*, 2008.
11. Z. Liu, Y. Shen, K. Ross, S. Panwar, and Y. Wang. LayerP2P: Using Layered Video Chunks in P2P Live Streaming. *IEEE Trans. on Multimedia*, 11(7), 2009.
12. G. Nascimento, M. Ribeiro, L. Cerf, N. Cesario, M. Kaytoue, C. Raissi, T. Vasconcelos, and W. Meira. Modeling and Analyzing the Video Game Live-Streaming Community. In *LA-WEB'14*, 2014.
13. V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing Streaming Media Content using Cooperative Networking. In *ACM NOSSDAV'12*, 2002.
14. V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *IPTPS'05*, 2005.
15. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An Application Level Multicast Infrastructure. In *USENIX USITS'01*, 2001.
16. J. Rückert, T. Knierim, and D. Hausheer. Clubbing with the Peers: A Measurement Study of BitTorrent Live. In *IEEE P2P'14*, 2014.
17. R. Tian, Y. Xiong, Q. Zhang, B. Li, B. Y. Zhao, and X. Li. Hybrid Overlay Structure Based on Random Walks. In *IPTPS'05*, 2005.
18. 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*, 21(3), 2010.
19. S. Xie, B. Li, G. Y. Keung, and X. Zhang. Coolstreaming: Design, Theory, and Practice. *IEEE Trans. on Multimedia*, 9(8), 2007.
20. C. Zhang and J. Liu. On Crowdsourced Interactive Live Streaming: A Twitch. TV-Based Measurement Study. *arXiv preprint arXiv:1502.04666*, 2015.
21. X. Zhang and H. S. Hassanein. TreeClimber: A Network-driven Push-Pull hybrid Scheme for Peer-to-Peer Video Live Streaming. In *IEEE LCN'10*, 2010.
22. X. Zhang and H. S. Hassanein. A Survey of Peer-to-Peer Live Video Streaming Schemes - An Algorithmic Perspective. *Comp. Networks*, 56(15), 2012.
23. S. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. In *ACM NOSSDAV'01*, 2001.