By how far have recent churn-reduction techniques improved quality of experience in peer-to-peer video streaming environments?

Martin Symons

A dissertation presented for the degree of
BSc Computer Science

School of Mathematics, Computer Science and Engineering
Liverpool Hope University
United Kingdom
May 7, 2024

Abstract

The past decade has proven fruitful for P2P livestreaming research, though limited commercial uptake has kept the total performance improvement vague. To quantify this research, we aim to build three representative models under an *OverSim* simulation and compare in various QoE heuristics. Surveying recent P2P livestreaming research, we find candidate replacements for several modules in popular network *New Coolstreaming*, and select a single competitive monolithic model. In attempting an implementation, we find numerous gaps in *New Coolstreaming*'s literature that negatively impact reproducibility in the research space. A new model, *coolstreaming-spiked*, is proposed to correct these errors without compromising to meet production requirements. A novel partnership algorithm *Partnerlink* is produced to resolve various unknowns in the *New Coolstreaming* partner relationship program. These algorithms form a more valuable basis for reliable research in the field, whilst remaining compatible with prior *Coolstreaming* research.

1 Declaration of Originality

I hereby confirm that this assignment is in whole my own work, that no part of it has been copied from any other person's work, published or unpublished, and that any work paraphrased or quoted has been cited and fairly credited in the bibliography.

I have read the guidance provided by Liverpool Hope University and understand the consequences of academic misconduct, including that it may result in the termination of my studies at Liverpool Hope University.

2 Acknowledgements

Thanks to my supervisor Mark Greenwood for advising me not only through this paper, but the course as a whole. To everybody else: thanks.

Contents

List of Figures

List of Tables

Acronyms

CDN Content delivery network

DHT Distributed hash-table

IP Internet protocol (address)

P2P Peer-to-peer

QoE Quality of Experience

QoS Quality of Service

RTT Round-trip time

TCP Transmission control protocol

3 Introduction

Within the past ten years, video streaming traffic across the internet has exploded. Traditional client-server architectures place a large load on the small portion of nodes controlled by the streaming host, and lead to dramatic infrastructure costs. To combat this, peer-to-peer systems have been proposed that spread usage across the network's collective resources. Starting with *DONet/Coolstreaming* in 2004 these networks became defacto for IPTV streaming, their proven utility accelerating research in both pace and complexity. Industry, however, did not take. Since the shutdown of *Coolstreaming* and *PPLive* in the late 2000's, these newer solutions have seen little real-world usage.

As such, the actual impact of current research on network performance is vague. Still, research since has suggested that churn has a substantial and exponential impact on client quality-of-experience (QoE) beyond that which was considered in these original models. This paper attempts to investigate several marked improvements over the best-documented benchmark architecture, *New Coolstreaming*, with particular focus on methods to reduce churn throughout the network. In our proceedings we find several inaccuracies in the specification of this network interrupting our efforts, which we discuss and document. *coolstreaming-spiked* is proposed as an alternative research-oriented model with a strict specification, having these errors corrected. As part of this effort, we produce the novel *Partnerlink* partnership algorithm, forming relationship webs of any scale with minimal link breakage and including a self-healing *panic* mechanism to recover from unexpected failure.

The rest of the paper is structured as follows. Section 4 provides a background into P2P livestreaming and defines the research we intended to investigate. This research is compiled into three concrete models in Section 5. Section 6 timelines our attempts to produce these models, of which one is completed and tested for effectiveness in Section 7. Section 8 explains the problems found in *New Coolstreaming* throughout our implementation attempts. A research-oriented alternative *coolstreaming-spiked* is then formally specified in Section 9. We conclude in Section 10 and summarize further avenues of research in Section 11.

4 Literature Review

We review prior work to understand the essential components of a peer-to-peer livestreaming system, and analyze these piece-by-piece for any potential improvements. These improvements are considered against a generally-

understood set of heuristics, providing insight into the QoE for any potential user. We determine our baseline model for comparison, *New Coolstreaming*, and ways in which it could be improved on a modular basis. Finally, we introduce two contemporary monolithic network architectures to represent the state of highly optimized, production-ready designs in current-day research.

Section 4.1 describes the basic analytical model many networks are built from, and its four constituent components. Heuristics used for performance analysis across the P2P video streaming field are established in Section 4.2, alongside a discussion of our focus on churn in Section 4.3. Components of the prior analytical model are dissected to find potential churn resistance improvements in Sections 4.4, 4.5 and 4.6, investigating overlay structure, peer selection strategies and chunk schedulers respectively. We introduce *New Coolstreaming* and its older form *DONet* in Section 4.7, and pick compatible improvements from our investigations so far in Section 4.8. Section 4.9 concludes with a discussion on monolithic architectures.

4.1 Peer-to-Peer Streaming Fundamentals

Recent explosions in demand mean video streaming consumes 38% of current internet traffic (Sandvine 2024). Despite global innovations in CDN and edge computing technologies, an imbalance still arises in client and server resource availability, forming a bandwidth bottleneck (Ramzan, Park, and Izquierdo 2012). Media dissemination at scale hence remains prohibitively expensive. Peer-to-peer protocols aim to spread the bandwidth load across the complete set of peers, detaching bandwidth demands from network size whilst retaining high video quality. These protocols are generally split into VoD and livestreaming purposes, of which we focus on the latter.

Livestreaming presents a unique set of challenges compared to traditional P2P networks like BitTorrent (Cohen 2017), where QoE is measured mostly by upload speed and piece availability. These two priorities can be trivially arranged through a rarest-first tit-for-tat piece retrieval algorithm; rare pieces are disseminated across the most productive peers to ensure data availability and maximal throughput. Under a livestreaming environment, however, additional constraints apply. Blocks must arrive strictly in-order for efficient filling of client buffers, and end-to-end delay must be minimized to provide a satisfactory "real-time" experience. Different nodes may also maintain different playout positions within the stream, which must be carefully synchronized. Resolution of these factors necessitates network designs unique for this purpose (Liu et al. 2008).

Friedman, Libov, and Vigfussony 2015 surveys existing designs and expresses their structure in four key modules:

- The *player module*, which consumes the received video signal to display to the user. The current and any halting, buffering or chunk skipping mechanisms are also held here.

- The *streaming module*, which reads buffered chunks to exchange with neighbors in the overlay, and makes requests on behalf of the player ahead of playout. Depending on the chunk dissemination strategy, discussed in Section 4.6, this will also hold responsibility over gossiping the availability of chunks in the local buffers, or requesting long-term block service through the formation of parent-child relationships.

- The *overlay module*, which manages a set of short-term *known* nodes and long-term *neighbor* nodes ready for video transmission. These sets are built mainly from a list retrieved on interval from the origin node or through gossiping.

- The *network module*, abstracting the lower-level networking stack and applying any necessary encryption.

A node's life therefore begins by spinning up the network module, learning of some nodes to place in its *known* buffer, adapting these to *neighbors* based on some criteria, and exchanging initial block information. Once enough information is gathered, chunks will be requested or transmitted automatically to be stored in memory and eventually played out at the player module. We now explore the metrics used to analyze the performance of this model.

4.2 Performance Heuristics

Network heuristics can be broadly classified into two categories: service integrity, covering aspects that apply to any time-series media playback, and audiovisual, describing encoding-specific characteristics in a more subjective manner (Moltchanov 2011). As our simulation study does not support analysis of the actual encoded signal, we focus for now on the former category.

Hei, Liu, and K. W. Ross 2007 describes three key measures in an analysis of the PPLive (PPLive 2008) network.

- *Playback continuity* (also known as *playout rate* or *playout probability*) is given top priority, measuring the ratio of blocks received by the player module before their playout deadline.

- *Start-up latency* or *start-up delay*, measuring the interval between time of entry to first block playout. This is the first metric impacting user QoE upon network entry, ranking just below continuity in importance.

- *Playback delay* or *end-to-end delay*, measuring the interval between a block entering the network until playout at a given node. The importance of this metric depends on the interactivity of a stream: it may rank above start-up delay for real-time purposes, whilst other use cases may prove highly delay-tolerant.

These measures correlate various facets of playout quality with user QoE. Additional checks on *bandwidth utilization* and *messaging overhead* can provide insight into the effects of the overlay and streaming modules (Carta et al. 2010). In our paper, however, we focus on broader aspects of churn and its impact on topology for QoE improvement.

4.3 Churn

In a P2P system, a node's participation is not controlled or supervised; it may leave or join at any time without notification. Particularly under certain overlays, though, the QoE of any node is reliant on that a continued relationship with another - if the partner node leaves, the video stream may suffer an interruption. These dynamics, known on a macro scale as "*churn*," are a fundamental problem that must be solved by any robust P2P system (Stutzbach and Rejaie 2004).

Churn's impact on QoE is well-studied in structured DHT networks. In this environment, high periods of churn are associated mostly with a hit to startup delay (Ho et al. 2013). Under the additional constraints of livestreaming, however, this impact expands to cover all measures of performance. Nanao et al. 2012 finds that the mean churn time of a simulation noticably damages forwarding interval and end-to-end delay in topologies where the search time for a new parent is above 100ms. In even the fastest-resolving network discussed in our paper, reselection takes at least three times this interval (Sina et al. 2020). Simulation analysis in Kang, Jaramillo, and Ying 2012 finds that playout probability across all three tested chunk scheduling strategies also worsens up to 17.2% under high churn.

Understanding these direct impacts of churn on QoE, we bring particular note to the conclusions of Vassilakis and Stavrakakis 2010, drawing correlations from the playout quality and QoE at a node to its chance to churn. Hence, a network with high churn suffers poor QoE, and a network with poor QoE suffers further from worsening churn. This user-driven view of churn is reminiscent of the discovery of the Zipf distribution within P2P file-share peer lifetimes (Pouwelse et al. 2005), which has since become a research topic in its own right (Bustamante and Qiao, n.d.); whether this characterization of P2P streaming churn should be as influential in streaming networks remains

an open research question, though its compounding nature leads to its prioritization in our research. Real-world studies on *New Coolstreaming* (discussed in Section 4.7) solidify our interest in this area, where churn is described as the most impactful factor on overall performance (Bo Li et al. 2007).

We now focus on each component of the basic model we have discussed, comparing research under each to find candidate replacements for improved churn management.

4.4 Overlay Structure

Streaming systems can be mostly split into two categories, tree-based and mesh-based. Early P2P streaming networks were designed with the expectation of a rise in IP multicast uptake across the underlying internet, locating multi-host dissemination within the low-level network stack (Deering 1989). Concerns regarding error correction and scalability eventually prevented this adoption (Chu et al. 2002), bringing interest to alternative end-to-end approaches.

Designers first moved to single tree overlays, a natural analog to multicast. Tree overlays are structured, with each node forwarding the complete data stream to a greater number of children, administered by a single server at the top of the hierarchy. Goh et al. 2013 finds through comparative simulations that resulting playback delay is up to 114% lower than in a mesh, observing similar improvements to playout rate. We doubt the validity of these results, however - the paper proposes playback delays shorter than the RTT of a typical single hop in trees of up to 1000 nodes. Still, it is generally understood that tree overlays perform better in delay statistics that than their mesh counterparts Awiphan, Su, and Katto 2010, though at the expense of enormous susceptibility to peer churn, since a departing peer temporarily detaches all child nodes from the tree until the overlay heals (Ghoshal et al. 2007). The uplink bandwidth at each node in a tree also acts as a bottleneck for all children. If any one node cannot transfer the full description as necessary for playout between its children, no alternative source is available for retrieval, and QoS will suffer throughout the remaining tree (Magharei, Rejaie, and Guo 2007).

Multi-tree networks have been proposed to mitigate this effect. Video is split into multiple substreams, each directed through an independently managed tree. A peer can join as many trees as necessary to meet playout demand, limited only by its download bandwidth. Since a downstream node retrieves the complete encoding from multiple parents, a single failing node no longer has catastrophic impact on its children. If a layered encoding method is chosen, playout may only degrade in quality instead of buffer

(Zink and Mauthe 2004). Analysis at the time suggested the improved reliability and churn resistance came at the cost of a more complex design and intensive media decoding requirements (Ghoshal et al. 2007). The former is not a concern for a production network at scale; the latter has since been resolved by the proliferation of advanced hardware video decoding chips and the integration of layered encoding in *de facto* codecs like H.264 (Seeling and Reisslein 2012). We would now consider that a multi-tree overlay represents a guaranteed improvement over single-tree.

Still, mesh networks have become standard, forming a randomly connected unstructured overlay of parent-child relationships. Each node holds multiple parents and multiple children, resembling a multi-tree; however, nodes can now freely move anywhere in the swarm, using any other node as a source for blocks, without maintaining more than one overlay. Churn recovery is thus bolstered even further, and nodes can easily move away from any detected bottleneck on a given substream (Magharei and Rejaie 2006). Simulation results in Magharei, Rejaie, and Guo 2007 suggest bandwidth utilization in mesh networks is universally superior to a tree approach, remaining beyond 90% of the optimal in all experiments, whilst tree approaches proved less efficient and more sensitive to improperly tuned parameters. At least one additional quality level was received compared to the tree equivalent even when ideally tuned. Bandwidth utilization in a tree is also shown to crater significantly under churn, whilst staying constant in a mesh. We consider these results more reliable than those presented earlier in support of tree networks - Magharei is a respected opinion in overlay construction, being cited within a number of influential papers, and has provided unbiased insight into the field's nuances across their whole portfolio.

In our paper, we consider only mesh networks due to their well-researched performance under a variety of conditions, and their resistance to churn proving relevant to our priorities. That said, hybrid networks combining tree, multi-tree and mesh networks at once have been proposed. A sweeping analysis of these toplogies proves unhelpful due to the wide range of approaches seen in the field. As such, we discuss only the performance of relevant hybrids on a per-network basis later in section 4.9.

4.5 Peer Selection Strategies

Peer selection strategies can be split into three types: random, QoS-aware and locality-aware (Kim, Kim, and Lee 2018). Random selection methods are highly resilient to churn (Vishnumurthy and Francis 2007) and provide inherent load balancing (Wang, Fu, and Chiu 2008), though at a cost. *PPLive* was noted to suffer very high startup delay of up to 83.5 seconds in some

channels, after a failure to quickly accrue quality nodes - the first random batch usually contained nodes with insufficient bandwidth or were unreachable due to network constraints. After 150 seconds, the average node only connected to four parents Hei, Liu, and K. Ross 2008. This is not viable compared to today's near-instant CDN delivery; *PPLive*'s popularity proves that this approach can be sufficient, but alternatives should be considered.

QoS-aware methods prioritize nodes by some network heuristic before supply for connection. *OCals* tests RTT both during construction and during scheduling via TCP-Friendly Rate Control (Floyd et al. 2000), passing as many candidates for which the new node can provide good QoS as those providing good QoS to the node. Compared to established random selection strategy *SCAMP*, startup delay remains approximately constant whilst average throughput increases 103% in a 1000 node network. Gossiped heuristics can also provide system benefit. Nodes under *Chameleon* (Nguyen, Li, and Eliassen 2010) are bootstrapped with random nodes by *SCAMP*; the overlay then moves nodes matching a heuristic against the requested *quality level* closer together. Senders within these nodes are filtered again based on matching upload/download bandwidth and lowest available layer count. This strategy massively improves playout probability at scale whilst boosting quality satisfaction up to 24.7% in smaller networks, when compared to simpler bandwidth-based mechanism *FABALAM* (Liu, Dou, and Liu 2004).

Locality-aware methods are the black sheep of the family, primarily aiding underlay efficiency instead of overlay performance. They reduce end-to-end delays to a minimum, though lack the churn-resilience and throughput utilization better provided by QoS or even random approaches (Zhao and Wu 2012). Fluid representations of the selection strategy categories proved that locality-aware methods can push better performance in networks with universally high RTT (Couto da Silva et al. 2011), though this is a rarity in today's highly capable networks. Magharei et al. 2014 proposes *OLIVE*, explicitly aiming to reduce demand on the underlying routing hardware and support the best-effort internet. Though this is a valiant effort, compromises to playout quality for the sake of such optimizations are unlikely to see uptake until a minimum QoS barrier is crossed.

Improvements originating in selection strategy can be amplified by exercising the strategy wherever possible across the network. Budhkar and Tamarapalli 2017 proposes three unique sorting algorithms across the network. At entry, the origin node prioritizes peers with high upload bandwidth and a full buffer. Entering nodes filter this list further according to locally perceived upload, propagation delay, lag and buffer before initiating connections. Throughout a node's lifetime, the overlay module scores all ancestors once more to slowly drive performance as playout continues. This thorough

application of principles reduces startup delay by 10% in all cases, and playback delay and parent search time by up to 20% under churn. This may incur additional performance costs local to each node - however, modern hardware should be capable of these calculations without meaningful strain.

4.6 Chunk Schedulers

Chunk schedulers under a mesh topology are generally split into pull, push and hybrid-push-pull approaches, depending on the node taking responsibility of chunk selection. Whilst most aspects of P2P live streaming have been thoroughly surveyed, insights into the ideal scheduler approach appear limited to suppositions made in the proposal of new overlays, and a complete survey remains an open research question.

Pull schemes run the scheduler on the child node, whom determines blocks to request from its parents based on the status of their buffer maps. Overhead under this scheme is high: the sequential processes of neighbor selection, buffer map exchange, chunk selection, request and finally delivery add up, resulting in high end-to-end delay (Hei, Liu, and K. W. Ross 2008). Such a strategy is therefore rarely proposed for low-latency interactive use (M. Zhang et al. 2007). Under mathematical modelling, it is proven that pull schemes provide near-optimal playout probability as long as a node is allowed to push at least three blocks per timestep (J. Zhang et al. 2014). Duplication is not an issue, as nodes control exactly when and how each block is received (Lo Cigno, Russo, and Carra 2008). Generally speaking, pull schemes are considered "good enough" - even when effort is spent to determine the ideal scheme, Liang, Guo, and Liu 2009 finds through real-world internet experiments that the chunk scheduler only becomes relevant when both the streaming rate approaches maximum throughput for a network and minimal end-to-end delay is desired.

Still, we consider the alternatives. Push schemes have seen some minor uptake. Under this arrangement, the scheduler is run on a parent node automatically pushing received blocks to its children, who attach or leave the parent at will based on their own heuristics. The children, however, have no say in the blocks that they receive; parents instead run an algorithm to determine which block should be pushed to which peer, for which many solutions have been proposed. Massoulie et al. 2007 finds through fluid modelling that the *most deprived peer, random useful chunk* algorithm provides ideal throughput, whilst Sanghavi, Hajek, and Massoulié 2006 theoretically proves the optimal delay of *random peer, latest blind chunk* in pursuit of a hybrid approach. Bonald et al. 2008 notes, however, that the delay performance of the former and throughput of the latter are suboptimal for use in a

livestreaming environment. Through simulation and mathematical analysis, the *random peer, latest useful chunk* algorithm is instead found to provide the best compromise between these factors, with methods suggested to reduce message overhead and implementation complexity. Regardless, this does not appear any more optimal than the aforementioned pull schemes, whilst requiring more complex mathematical models to run.

To simplify operations, hybrid-push-pull approaches allow the parent node to push received blocks as before. However, the child nodes are given autonomy to select a subset of blocks to receive, usually by splitting the video into substreams. Such an approach regards the pull mechanism primarily as a routing mechanism, reducing routing, messaging and modelling overhead, whilst maintaining the high bandwidth utilization offered by push networks (M. Zhang et al. 2007). The resulting granular control is tempting for those producing monolithic, tightly-optimize overlays, as explored in 4.9.

4.7 Coolstreaming

We now consider historic implementations of peer-to-peer streaming systems in relation to the above characteristics. An explosion in IPTV popularity in the mid-2000's lead to a great number of new networks - *PPLive*, *SopCast* (SopCast 2019), *UUSee* (UUSee 2007) et al. The majority of these networks were commercial endeavors, and therefore closed source.

Of most relevance to our paper is the *Coolstreaming* network. Thanks to its origins in academia, it remained open-source throughout its lifetime despite commercialization, whilst its eventual serving of over 80,000 simultaneous users lead to the gathering of an enormous quantity of data Bo Li et al. 2007. A great deal of benchmark information is therefore available for our use.

Coolstreaming was initially built on the *DONet* architecture, evolving over several years to form the *New Coolstreaming* framework. We provide a brief comparison of both.

DONet (X. Zhang et al. 2005) forms a mesh-pull topology with random peer selection. Nodes are built of four key components:

- The *membership manager*, maintaining a random partial view over the swarm - the *mCache*.
- The *partnership manager*, adapting nodes from this view into longer-lived connections suited for video transmission.
- The *scheduler*, holding responsibility over the chunk scheduler and requesting blocks from other nodes

- The *buffer*, performing playout and acting as a bridge between the overlay and player client.

Each node is provided a unique *NodeID*, usually an IP address. Nodes initially contact a central origin for entry, who redirects to a member chosen at random from its *mCache*. This *deputy node* provides a larger list of random candidates from which the entering node can initiate partnerships and request its first blocks. A membership message is also gossiped as defined under popular protocol *SCAMP* for further *mCache* upkeep.

Nodes begin to exchange buffer maps once partnerships have been established. Partnerships are bidirectional - a node requests and pushes blocks through the same connection. Buffer maps are represented as a single boolean bitmap of length B, where B is the length of the buffer window. Each entry denotes the availability of one block. Children process these maps through a *rarest-first* chunk scheduling strategy when making requests, giving blocks with only one supplier absolute priority.

Node departure is advertised either by a leaving node itself, or by a partner noticing the departure sending a message on its behalf. On interval, the partnership manager ranks all partners by a score: the average blocks uploaded or average blocks downloaded per unit time, whichever is higher. The worst scoring partner is then swapped out for a fresh member from the *mCache* to converge on better partnerships. A limiting factor M is placed on partnership count, provided 4 as example.

In contrast, *New Coolstreaming* (B. Li et al. 2008) employs hybrid-push-pull scheduling. This specification poses unique challenges, as explored in Section 6, and so we summarize here with hindsight information. The *SCAMP* protocol has been removed, replaced with a simpler method where the origin node maintains and provides a list of peers directly. Partners from this list are no longer regularly swapped, only *reselected* after any insufficient service is found. Two new heuristics detect this improper service, taking the form of inequalities; one limiting the gap between the local node's buffers and the parent's, and another limiting the gap between the parent's buffers and the rest of the swarm. If these inequalities do not hold, a new partner is swapped in for whom they do.

The buffer and scheduler are coupled to form the *stream manager*. Video is split into substreams, with buffer maps now comprising two tuples - one displaying the most recent block received in each substream, the other a node's subscriptions to the receiving parent. Nodes subscribe to a parent with a single buffer map message, after which the parent will continue to push blocks until the child explicitly unsubscribes or disconnects. The relationship is not broken under any other circumstances.

Finally, a new method for calculating the initial block number is defined, which was not covered as part of *DONet*.

The improvements *New Coolstreaming* proposes against *DONet* are typical of a pull vs hybrid-push-pull mechanism - lower messaging overhead, lower end-to-end delay, and better support for layered video encodings leading to improved QoS. In comparative analysis, median start-up time also reduces from 40 to 24 seconds (Bo Li et al. 2007). As more data is available on *New Coolstreaming* than *DONet*, and the approach is more in-line with contemporary monoliths, we intend to use *New Coolstreaming* as a benchmark for our analysis. Based on the key components identified above, we now discuss candidate churn-focused improvements that could be suited for *New Coolstreaming*.

4.8 Coolstreaming-Compatible Churn Improvements

New Coolstreaming's overlay management broadly overlaps with Chameleon's, whose QoS-aware peer selection strategy and resultant churn resistance improvement has already been discussed. Wang, Zhang, and Yang 2013 proposes a further measure to adjust a peer's partner count based on its bandwidth, mixing well with *New Coolstreaming*'s existing M factor. When taken with the analysis of Vassilakis and Stavrakakis 2010, we expect that this should further improve churn resistance - nodes with lesser bandwidth receive worse playout quality on average, and are thus more likely to churn. By subscribing less nodes to these peers, we drive network topology towards longer-lived partnerships. These two papers seem compatible together; we aim to adapt both for *New Coolstreaming*.

Ho et al. 2014 augments *New Coolstreaming*'s chunk scheduler with a complex heuristic involving upload bandwidth, latency and playout delay to determine its downstream quality. Blocks are then sent by priority to nodes which can best service their peers, encouraging lesser nodes to reselect onto them. Simulation experiments within this paper determine its superiority over other simple bandwidth-aware or locality-aware schedulers. Another paper Li, Tsang, and Lee 2010 performs more advanced bandwidth *weighting* which may prove competitive, as a substantial reduction in parent reselection rate is shown. However, these results are based on a network with additional functionality, namely a last-effort pull mechanism for missing blocks. Without any means to immediately determine the better approach, we choose to implement the former for its simplicity.

We also propose some improvements outside the traditional view of a P2P architecture. Wu, Liu, and Ross 2009 decouples what a peer uploads from what it views, spreading peer resources to smaller channels to improve

their playout quality. Switching delay also improves, and we anticipate that churn chance should also reduce as nodes "stick" to channels for longer, even through disruptive behaviour like channel surfing. Whilst we cannot simulate channel prediction, the inclusion of some ratio of non-playout nodes could provide useful insight. Huang, Ravindran, and Khan 2010 introduces powerful bootstrap agents into the network, selected for high bandwidth and predicted lifetime, which pre-schedule a download plan for incoming nodes and fill their initial buffer. Churning parents are most disruptive as a node is building its initial buffer for playout, greatly increasing startup delay, so this eliminates a major weak point in *New Coolstreaming*'s current strategy. The existing *DONet* deputy system provides a good baseline for implementation, although the downloading plan cannot be used under *New Coolstreaming* due to the hybrid-push-pull approach. We hope to incorporate both of these novel components into our solution.

We now discuss monolithic solutions developed since the introduction of *New Coolstreaming*.

4.9 Monolithic Single Solutions

WidePLive (Sina et al. 2020) forms a mesh-hybrid-push-pull topology, tightly coupling the overlay construction algorithm with the chunk scheduler via a shared database. Numerous statistics are tracked about each node that are taken into account when finding long-lived, high-contribution peers across both components. A node's actions during the chunk scheduling phase will therefore impact its future position in the overlay, and vice versa. Almost all actions are buffered, prioritized and acted upon after a short period, granting a node greater intelligence about its overlay position and future service. Nodes using the origin as a parent are known as *root peers*, forming a novel load-balancing network to equalize block dissemination between all children. Analysis of these changes against simpler mechanisms similar to those in *PPLive* shows improvement across almost all heuristics: playback latency improves 41.5%, startup delay improves 67.9%, and playout rate sees some small improvement. No analysis is performed directly on the impact of churn, however - a gap in literature which we would aim to resolve.

WidePLive's key disadvantage is its complexity. The reference implementation in *OMNeT++* comprises 3000 lines of code. Given the large scope of the project already proposed, an implementation of this network would likely exceed our time constraint.

A promising alternative is seen in the synthesis of two papers. The *AQCS* chunk scheduler (Guo, Liang, and Liu 2008) marks all blocks in the network as *F* (*forwarding*) or *NF* (*non-forwarding*). All client nodes receiving *F*

content will mark it as *NF* and forward it to other peers; any *NF* packets are filtered out on reception. *F* blocks are stored in a buffer. When this buffer becomes empty, the responsible node pulls three new *F* blocks from the origin. If the server completes its backlog of pull requests and becomes idle, it serves one duplicated block to all clients. A last-effort block recovery mechanism is also included. This approach achieves within 10% of the optimal bandwidth utilization for a P2P system as calculated by a stochastic fluid approach (Kumar, Liu, and Ross 2007), without the need for any complex scheduling algorithm. This drops to 12% under churn - an insignificant difference.

AQCS's approach is limited by its need for a fully-connected network, making it unviable for a substantially-sized system. In Liang, Guo, and Liu 2007 a hierarchal overlay construction is therefore discussed, whereby the origin nodes of each *AQCS* cluster form a tree overlay themselves using QoS-aware peer selection. The churn susceptibility of this tree is not a concern due to its small size - in a system with 10 clusters directly owned by the origin, where each cluster holds 20 nodes, the hierarchy can support 4000 peers in two layers. In this way the connection overhead is split across the network, whilst retaining 90% of optimal bandwidth utilization.

Our results on this system will not fill any gaps in the paper itself, as churn has already been studied. Even so, the logic behind this combined network is much simpler than *WidePLive*, and is more feasible to produce within the time provided. We thus aim to implement this as an additional point of comparison beyond a simple upgrade to *New Coolstreaming*.

This marks the end of our background review. We now summarize the models we have produced, and discuss the means of our analysis.

5 Methodology

Our prior discussion has produced three candidate models:

- The default *New Coolstreaming* model, as a baseline.
- An enhanced *New Coolstreaming* containing substitute modules proposed in modern research. This includes replacement QoS-aware peer selection strategies and chunk scheduling, decoupling of viewing from uploaded material, and bootstrap nodes to accelerate node entry.
- The hierarchical *AQCS* model representing the modern monolithic approach.

We aim to produce these three models in a simulation, pitting each in numerous scenarios to gather detailed performance heuristics. Comparisons of this data will reveal the level of improvement at each stage, and any weaknesses of our considered research.

As a simulation environment, we chose *OverSim* (Baumgart, Heep, and Krause 2007). Its included churn mechanisms, quick-switching between simplified and realistic underlay models, and complete debugging suite eased the otherwise involved development cycle. Ejecting from *OverSim* to *OMNeT++* would also have been trivial, though this was never necessary.

Statistics have been gathered with the built-in *OMNeT++* collection and visualization tools.

6 Implementation

We based our initial experiments on *New Coolstreaming* as described in B. Li et al. 2008. We quickly found trouble - whilst the paper describes the stream manager and buffer map exchange in great detail, little space is given to the membership and partnership managers. The upkeep of the *mCache* with incoming peers is unspecified, as is most connection management action related to churning or failing nodes. Some key equations to system function also do not appear. We pushed forward and attempted to fill the blanks ourselves; the final result, whilst technically functional, invariably failed to meet playout across nodes and was in no way correspondent of *New Coolstreaming*'s measured real-world performance.

The *New Coolstreaming* paper concludes its discussion on the problem modules stating *"these basic modules form the base for the initial* Coolstreaming *system,"* and that *New Coolstreaming "has made significant changes in the design of other components."* We thus considered that these modules were holdovers from the older design, implying *New Coolstreaming* must be built up using *DONet/Coolstreaming* as groundwork. We thereby set about an implementation of this more primitive network.

The final *DONet* implementation took two weeks to complete. This network was not fully performant, though the cause was generally understood - as *New Coolstreaming* replaces the buffer, scheduler and related messaging completely, we saw no need to optimize these components. More worryingly, the partnership manager collapsed quickly under even minimal churn. Still, this constituted the groundwork needed to continue.

Returning to the original problem, we found that our architecture still did not align with the basic modules in *New Coolstreaming*. This proved troubling. As discussed later in Section 8.6, *DONet* has clarity problems of

its own when describing parts of other systems, and we reasoned this might similarly be the case for *New Coolstreaming*. Noticing that the output of these components - M-number exchanging partners ready for video transmission - *did* align with the older model, we therefore treated this as a simple faulty description, and moved on to the design of the stream manager.

The well-specified stream manager proved unproblematic, but placed new constraints on the partnership manager that our already brittle implementation could not bear. We hence designed *Partnerlink*, a relationship algorithm reconciling the high-churn overlay with *New Coolstreaming*'s low-churn subscription requirements and performance at scale. This new algorithm integrated well with the wider system, and brought our implementation to a close.

The full development process took over a month. We were therefore not able to complete any further models or make any inter-model comparisons on QoS benefits.

7 Results and Analysis

We repeat five tests on two scenarios, joining two origin nodes and 30 client nodes over the course of 90 seconds. Nodes live on average 500 seconds before churning; we allow 300 seconds for the overlay to stabilize before a 2000 second measurement period. The 500Kbps source video is split into 10 substreams, containing blocks of one second duration. Partner and membership count M is limited to 16.

Partner counts are recorded globally as a histogram at each node's playout to gain an overall view of the system. Block statistics are recorded locally as a scalar and averaged across all nodes.

We first test our model on a simple underlay without packet dropping, bandwidth limitations or timeouts. Table 1 shows that our partnership mechanism works well in this environment, averaging 15.89 partners across all nodes over the entire measurement period. We also find our manually-tracked potential partner count M_c to be the same value, though this may not be surprising, as our failure mechanisms are almost never engaged in this abstracted underlay. No node is ever seen below 12 partners in Figure 1. Our block efficiency shown in Table 2 is reasonable - of all received blocks, 3.78% are duplicates and 2.51% are outside the bounds of the receiver's buffer, almost all being too old. Despite this, our playout probability is poor: only 64.82% of blocks are received before playout time, far below expectation.

We test again on a realistic underlay simulating the best-effort internet, including bandwidth limitations, packet loss and queuing. Under this en-

vironment, the partnership mechanism fails: despite recording an average M_c of 13.65 in Table 3, our actual partner count averages 2.04. Figures 3 and 4 reveal a reasonable middle-ground surrounded by nodes simultaneously reporting $M_c = 16$ and *partner count* $= 0$ - we explore this and the resultant changes made to *Partnerlink* in section 9.5. Surprisingly, our playout performs very similarly, with a playout rate of 62.66%. Two problems are therefore implied: that our expected partner count is de-syncing over the course of network, and that our subscriptions are not wound down correctly after a partnership ends. This playout rate is in spite of a major hit to efficiency - 9.49% of all received blocks are now out of bounds, and 8.86% are now duplicates. The mechanics allowing playout rate to remain so high under these circumstances are unknown.

Partner Histograms at Each Simple Playout			
Histogram	Mean	Std.	Variance
Mc	15.89	0.33	0.11
Partner Count	15.89	0.33	0.11

Table 1: Statistical analysis of simple underlay partner counts at each playout

Block Scalars over Simple Node Lifetime	
Scalar	Mean
Hit Playouts	267.85
Missed Playouts	145.36
Accepted Blocks	273.27
Out-of-bounds Blocks	11.04
Duplicate Blocks	7.34

Table 2: Statistical analysis of simple underlay block statistics throughout the average node's lifetime

Partner Histograms at Each Inet Playout			
Histogram	Mean	Std.	Variance
Mc	13.65	4.24	17.98
Partner Count	2.04	3.79	14.44

Table 3: Statistical analysis of realistic internet underlay histogram values at each playout

Block Scalars over Inet Node Lifetime	
Scalar	Mean
Hit Playouts	171.33
Missed Playouts	102.09
Accepted Blocks	193.53
Out-of-bounds Blocks	18.36
Duplicate Blocks	20.59

Table 4: Statistical analysis of realistic internet underlay block statistics throughout the average node's lifetime

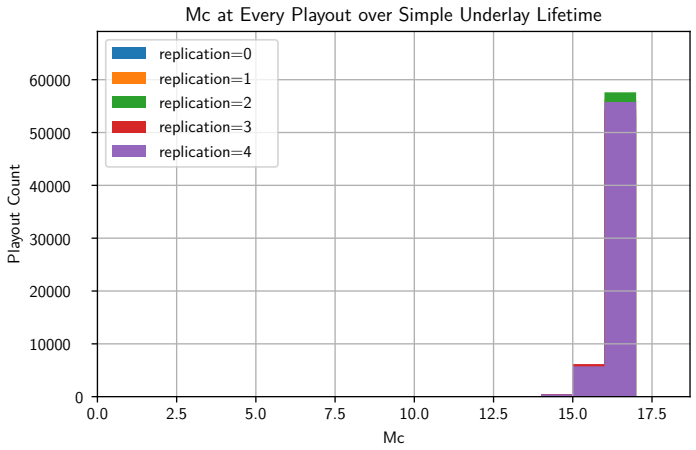

Figure 1: Histogram of M_c at each playout within the simple underlay experiment

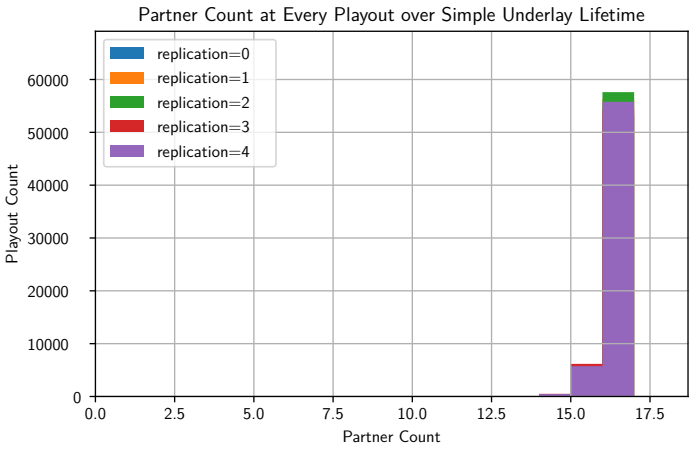

Figure 2: Histogram of actual partner count at each playout within the simple underlay experiment

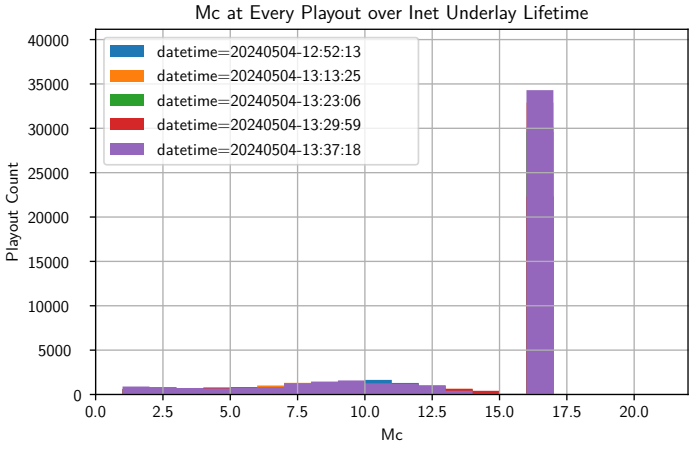

Figure 3: Histogram of M_c at each playout within the realistic internet underlay experiment

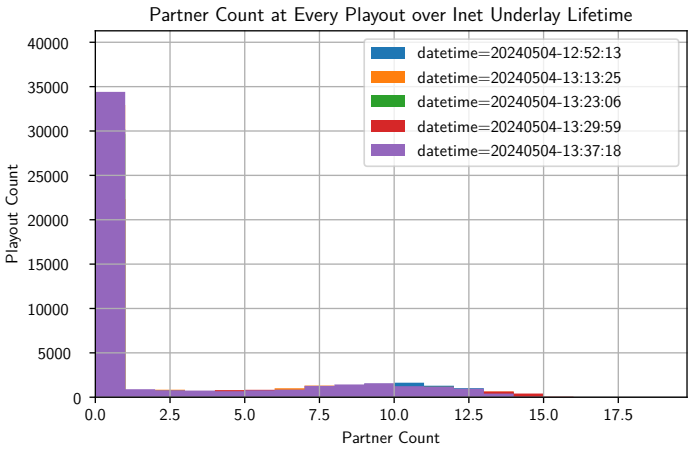

Figure 4: Histogram of actual partner count at each playout within the realistic internet underlay experiment

8 Discussion

What went wrong? New Coolstreaming is not particularly unique as an overlay; it fits our basic model and no features within should prove particularly challenging to implement. In this section, we explore the *Coolstreaming* family as a whole, and illuminate the many obstacles overcome in their implementation.

8.1 The Coolstreaming Family Tree

We have so far regarded *Coolstreaming* as an overarching name for the mesh-pull *DONet/Coolstreaming* and hybrid-push-pull *New Coolstreaming*, proposed across two papers. The reality is not so simple. *Coolstreaming* is formed of two models as described, though proposed under four different names across four papers:

- As *Coolstreaming* or *DONet*, in *CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming* X. Zhang et al. 2005
- As *Coolstreaming*, in *Coolstreaming: Design, Theory, and Practice* Xie et al. 2007
- As *Coolstreaming+*, in *An Empirical Study of the Coolstreaming+ System* Bo Li et al. 2007
- As *"the new Coolstreaming,"* in *Inside the New Coolstreaming: Principles, Measurements and Performance Implications* B. Li et al. 2008. Within this the name *Coolstreaming* is intended, but *New Coolstreaming* became colloquial as a result of its title.

The first paper defines what we have described as *DONet*; the remaining three regard *New Coolstreaming*. Hence, two papers describe the separate *DONet* and newer *New Coolstreaming* models under the shared name *Coolstreaming*.

This has lead to confusion amongst research peers. Kondo et al. 2014 describes the *SCAMP* membership protocol, the push-pull mechanism and the bootstrap node as belonging to the same model, despite *SCAMP* being specific to the mesh-pull *DONet*. Beraldi, Galiffa, and Alnuweiri 2010 makes similar errors. Lan et al. 2011 takes *DONet* as its key example of a buffer-map driven overlay, but ascribes it the synchronization method seen in *New Coolstreaming*. Several more examples can be found of valid discussion of a model, but with Xie et al. 2007 being cited in X. Zhang et al. 2005's place, or

vice versa. Further confusion exists within the papers themselves: B. Li et al. 2008 describes the stream manager and new mCache system as part of the *"initial Coolstreaming system"* and replaced in *"the new Coolstreaming system"*, despite all of these components being introduced in *New Coolstreaming* only.

This escalates considering the structure of the final three papers. Despite its name, Xie et al. 2007 is the canon definition of *New Coolstreaming*, provided alongside analysis of a real-world test period to determine convergence rates, start-up delay and other overlay-specific statistics. The other papers duplicate this and add further discussion: Bo Li et al. 2007 splits users into categories, identifying network traversal problems and their respective impact on contribution. B. Li et al. 2008 performs an additional simulation to identify ideal values for key system parameters.

As a result of this duplication, each paper hence opens with a definition of *New Coolstreaming*, though occasional and necessary details are cut in the latter two. For instance, connection management for partners is resolved in Xie et al. 2007 by a mention of TCP - which includes a leaving and timeout mechanism, if specified. In the other papers, TCP is never mentioned; no other connection management is described. Mechanisms to fill the bootstrap node's *mCache* alongside one key function related to playout initialization are similarly constrained to this earlier paper, despite these papers claiming to *"describe the architecture and components in the new Coolstreaming system"*. The outcome of this is that out of three papers, two with names implying upgrades, all appearing to contain complete specifications, only the oldest can be followed to completion. The correct specification for *New Coolstreaming* is not the paper named *New Coolstreaming*, but instead the paper sharing names with the older *DONet*.

Luckily, we appear to be the only researchers to fall into this trap. We accept responsibility in this instance: whilst the decision to focus on a single, seemingly up-to-date specification was valid, we should have been more eager to revisit the other papers as issues arose. Our results are partially affected, as we implement manual connection management over UDP instead of TCP, increasing our network's propensity to errors. We also retain the modified *SCAMP* implementation from *DONet*, although as later discussed this should not meaningfully influence our findings. Regardless, we stand by the conclusions we draw throughout the remaining paper. We next explore the role of *Coolstreaming* in current research and the priorities we aim to follow.

8.2 What does Coolstreaming need to be?

As part of this paper we aimed to determine if improvements to *Coolstreaming* could keep it viable in the present day. Its primary use, however, now lies entirely in the research domain - as a base for reproducible results (Liu, Wang, and Hsieh 2010), a testbed for new modular features (Ho et al. 2014) or a shared point of comparison between new models (Pal, Govil, and Ahmed 2017). In all of these cases, consistency and ease of development take priority over QoE performance.

The issues we will discuss are a sign that *Coolstreaming* is not designed for this purpose. *Coolstreaming* aimed to be the leading production overlay of its time, including optimizations that were inconvenient but deemed worth the miniscule performance boost. More than this, *Coolstreaming* as a paper was developed alongside the maintenance of a commercial system (Bo Li et al. 2007). This may explain some of the inconsistencies we see - updates to the system may have acted as a moving target for the paper authors.

In Section 8.3 we discuss complexities in joining a node to an already-full partnership graph. Section 8.4 finds a key limitation in the performance-monitoring inequalities. Two issues within the stream manager are compiled into Section 8.5. Finally, Section 8.6 defines the tendency to include complex optimizations unnecessary for a research network.

8.3 The Fully-Connected Network Problem

Both *DONet* and *New Coolstreaming* make the same demands on the partnership manager: given a random partial view of the network, create and maintain at most M connections ready for video transmission. Neither paper provides exact information on how this should be done.

Suppose a node n joins a network of size $M + 1$. All existing nodes in the network are fully saturated with partner count M. How can this node join the network?

In our early *DONet* implementation, we forced nodes to accept all incoming partnerships, removing an old partnership to make room. This created chains of dropped-and-replaced partnerships before the overlay settled, exponentially worsening the impact of churn. The move to *New Coolstreaming* came with the demand to maximize partnership lifetimes, as any additional churn would interfere with the push mechanism and result in wasted blocks being sent to a peer, thus totally invalidating our approach.

Several solutions exist, though none are trivial. Our final implementation requests two nodes to break their partnership, placing the new partner in the middle. This has a great number of implications on the network, most

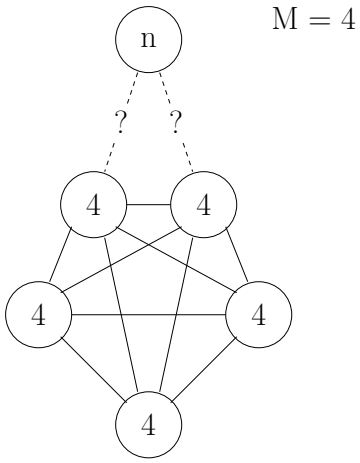

Figure 5: Node n cannot easily join this fully-connected network

notably that M must now be divisible by 2. Since *DONet* runs experiments on models with $M = 5$, this cannot be the implementation as intended by the original authors (X. Zhang et al. 2005).

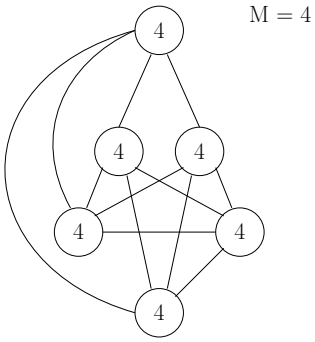

Figure 6: A solved topology, as according to the pairing method

We can at least infer some details on the expected implementation. The *mCache* data structure includes *num_partners* for each visible node, which is updated but never apparently used. In a healthy network, the likelihood of a node directly knowing another node with less than M partners is low. Instead, we assume this to be part of a gossiping mechanism where a gossiped partnership request is directed towards nodes with missing partners. This would also justify the use of *SCAMP* as opposed to a more standard partial view aggregator. We include a similar gossip strategy as part of our implementation; we discuss this in detail later in Section 9.3.3.

That said, the phrasing of the scheduled switching as established *"with*

nodes randomly selected from the mCache" may imply direct lines of communication. In this case, it is unknown how *DONet* or *New Coolstreaming* solve this problem.

8.4 Playout De-synchronization

The two inequalities measuring connection quality are also applied to all candidate partners following a detected failure. Inequality 1 limits the gap between a node's local buffers and the parent's. Assume that we have just failed a parent on this inequality: if we continue to apply this to our new candidates, we will filter out any nodes equal to- or further ahead our old node. The new node will therefore land further back in our buffer; assuming we never change our playout position, it will not be able to service our entire buffer area, reducing tolerances across the network.

Furthermore, the nodes we see passing in this scenario will be those similarly falling behind the swarm; that is, nodes with equally poor connections. This groups poorly performant nodes and drags them backwards across the buffers, eventually disconnecting them from the stream entirely. In our model we disabled inequality 1 when measuring candidates, and performance resumed.

This is an all-around improvement in models where playout will never pause. It is an expectation in modern streaming platforms, however, that the player buffers under interruption. In this case there is a real need to group peers by their buffer range - this would be better performed by block *proximity* instead of a hard limit, offering nodes the chance to stay connected to further-ahead peers that will not incur end-to-end delays. As we do not experiment with this playout strategy, we cannot prove any effectiveness of it - for safety, we do not specify this as part of *coolstreaming-spiked*. Further research on the topic may lead to its inclusion as default.

Some failure is still inevitable; neither *Coolstreaming* implementation specifies a recovery mechanism in case of overlay disruption beyond repair. In cases where playout does not halt, we can simply rebuild the partnership and membership managers without adjusting the stream manager, hoping to find a fresh set of peers. Where playout does halt, realigning the stream manager becomes a concern. It would be best to warn the viewer of the de-synchronization ahead of time, disconnecting if they ignore the warning, and provide the option to jump ahead in playout to rejoin the majority swarm.

8.5 Requesting the Block

Nodes entering *New Coolstreaming* are provided an algorithm to calculate the first block to request in each substream i. Upon receiving the complete set of buffer maps from its partners, a node finds the maximum block seen for i and subtracts a constant, intending to find a block approximately in the middle of the incoming buffers. We call this block B_i.

Without additional manipulation any new node would be forwarded the first partner's entire buffer on subscription, since the subscription map can only communicate "on" or "off." We must instead manipulate the latest block mapping to retrieve the correct stream. The node must advertise a latest block of -1 across all substreams until a first block is determined and a subscription is to be made. Upon subscription, the node must advertise to its new parent a latest block on substream i of $B_i - K$. This fools the parent into forwarding blocks from the correct index, after which the buffer map can be advertised as usual. Caution must be taken to only advertise these false blocks to their respective parent node, to avoid incurring illicit subscriptions from other newly joining nodes.

This mechanism is efficient and easily reproducible, but never explicitly stated. Our specification elaborates further to ease development. Though this raises the issue of starting playout index: now we know which block to request, where do we start playback in relation? Buffering systems can take the common approach, waiting at that block until some percentage of the buffer has been filled to begin. Non-buffering systems are more complex, as playback must be placed early to reach playout as the buffer is readily filled. *New Coolstreaming* regards the playout strategy as beyond its responsibility, despite playout position determining the buffer bounds. Resolution of this will require comparison of many possible approaches: we leave this as a future avenue of research.

8.6 Production Optimizations

DONet defers to *SCAMP* for membership connection management, before replacing several key components. Where *SCAMP* automatically adjusts membership count in sync with the size of the system, *DONet* enforces a hard limit of M nodes, usually 4. The indirection mechanism is also adjusted, with only nodes within the origin's view being available as contacts. As *DONet* only uses *SCAMP* to gather a random partial view, and does not use it as a means of message transmission itself, *DONet* does not require *SCAMP*'s full set of assurances. It is therefore free to make changes that influence the path of gossiped messages throughout the network. Whilst we have not

been able to perform experiments to prove this, we hypothesize that these restrictions make no negative impact on the network, acting only to reduce message overhead.

Other optimizations are less successful. *SCAMP* specifies a leasing system to remove dead nodes from the pool, which *DONet* incorporates. However, in the case that the partnership manager times out a partner, the partnership overrides the membership manager to remove the node from the *mCache*. The partnership manager then emits a departure message on its behalf, which is gossiped *"similarly to the membership message,"* i.e. once to a peer who then repeats it to every node in its *mCache*. Since exactly M departure messages are gossiped, every message would have to reach a peer knowing the failed node for a clean exit. The various indirections used under *SCAMP*, however, make it unlikely for this message to reach any such peer at all.

This mechanism is applied even to departure messages sent by a leaving node itself. Under *SCAMP*, each node knows the peers it is subscribed to through its *InView* list - thus, the only requirement for a clean exit is to send a departure to each node it contains. The gossiped alternative makes no gain in message overhead whilst almost totally invalidating the utility of a manual departure.

This appears to be a malformed optimization to allow nodes to depart each other in case of failure. Even if this were successful, the already-established leasing system means any performance gain would be negligible.

Successful or not, changes such as these represent a problem: we fall back to existing research to solve a problem, previously proven to provide optimal performance, and then modify it for some slight gain, in the process tampering with that proof. Analyzing the suitability of the new model becomes complicated, as the modified algorithm must be investigated from scratch to incorporate the changes. Furthermore, while this is convenient to build for the original researcher, any reproduction of the model requires careful cross-referencing of the two papers to find the correct feature set. New implementation of the model therefore becomes strenuous, and inaccuracies made much more likely.

This concludes our discussion on problems within the existing *New Coolstreaming* design. In the next section, we introduce a new model to the *Coolstreaming* family designed for research purposes, additionally describing its structure, function and rationale.

9 Introducing *coolstreaming-spiked*

coolstreaming-spiked is a new iteration of *Coolstreaming* bringing the two models into synthesis. Research usage is targeted through simpler internal logic, a thorough central specification and compatibility with a wide range of IPTV improvements. Specifically:

- The *SCAMP* protocol is retained for backwards compatibility with research performed on the original *DONet*.
- The partnership connection algorithm is solved and documented as *Partnerlink*, a novel approach providing realistic performance and absolutely minimal churn impact for networks of any scale.
- Production optimizations are stripped back to focus on a smaller scope.
- Other fixes, such as the disabling of inequality 1 on candidates, are implemented as standard.

We now proceed to describe the function of this model.

9.1 Architecture

Our component architecture is identical to that seen in *New Coolstreaming*: a membership manager provides the model with a continuous pool of random nodes. The partnership manager filters these nodes to find connections suited for video transmission, and handles the exchange of buffer maps. The stream manager creates parent-child relationships, owns the buffer and forwards blocks to peers. Each node within the architecture is provided a unique *NodeID* - an IP address will suffice.

We incorporate the substream system as defined in *New Coolstreaming*. The video sequence is split into blocks of equal size, each assigned a timestamp sequence number. These are shared equally between K-number zero-indexed substreams, where the i-th substream contains blocks with sequence numbers $(nK + i) : 0 \leq n < \infty$. Assuming $K = 4$, substream 2 would therefore contain blocks with ID $\{2, 6, 10, 14, ...\}$.

9.2 Membership Manager

The membership manager makes the initial contact to an origin node. We then subscribe as members to a subset of nodes using *SCAMP* to provide the partial view. No changes are made to *SCAMP* - the standard subscription,

unsubscription, indirection, leasing and recovery methods all apply. There is no limit on the size of the *mCache*. The partnership manager can no longer directly influence the mCache or send unsubscription messages on behalf of another node. These changes allow researchers to follow a well-specified, battle-tested paper for gossiping support without cross-referencing our own. The main output of this component is an arbitrarily sized set of random nodes across the network.

We note that *SCAMP* does not describe any starting topology. A bootstrapped *SCAMP* network must contain at least two nodes engaged in mutual subscription.

In the case that the node ever becomes completely isolated and can no longer aggregate members, the node should recontact the origin and gossip a fresh subscription message.

9.3 Partnership Manager and *Partnerlink*

The first known node provides a random list of candidate partners. The partnership manager takes these nodes and begins to form TCP connections ready for block transmission. TCP performs the majority of connection upkeep on our behalf - if the connection ends, times out or fails, the partnership should be considered *failed* - the partner connection is removed and M_c (discussed later) is reduced by 1.

The buffer map system remains as specified in *New Coolstreaming*. A buffer map comprises two tuples of length K - the first listing the highest block sequence number received in each substream, the second listing the subscription of substreams to the receiving partner. For instance, a node receiving tuples of $\{40, 41, 42, 39\}$ and $\{0, 0, 0, 1\}$ from a partner should infer a most recently received block 39 on substream 3, and prepare a corresponding subscription on that substream according to the subscription map. We also transfer the node's current *panic status* and a *NodeID* representing an *associated peer*, discussed later in Sections 9.3.3 and 9.3.4. A node receiving a buffer map should store these values alongside their relevant partner; the latest block should only be set if the subscription is new as of this message, as we update this value locally as part of the stream manager. If a partner does not transfer a buffer map in some specified length of time, the partner is missing and the partnership is once again *failed*.

9.3.1 *Partnerlink* Description

We now describe the *Partnerlink* algorithm. We first make two constraints on the system - a maximum number of partners at each node M is introduced,

where $M \mod 2 = 0$. Our basic join operation grants a node two links per request, so an even partner cap is essential. We define a starting network as two or more nodes in a valid system topology, since at least two peers are necessary to begin splitting. The formation of this topology is determinant on use case.

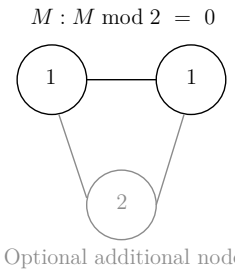

Figure 7: The base topology from which *Partnerlink* must operate

Each node tracks an additional set of variables. We maintain a set of *locks* against nodes for which operations are still in progress, to protect the system against overlapping requests regarding the same partner. We also maintain an *anticipated* partner count M_c, tracking what our partner count will become if all ongoing non-*panic* operations resolve successfully. Making decisions against this value rather than our actual partner count ensures we do not accidentally request partners above our hard limit M.

Partnerlink operations can be split into three categories:

- *Initiation*, allowing a healthy node to create new connections with known nodes.
- *Panic*, allowing an unhealthy node to recover back to M connections.
- *Leaving*, allowing the overlay to adapt under nominal churn.
- *Switching*, incorporating the *DONet* switching mechanism to converge on stronger partnerships.

We now discuss the systems and messages in place to allow for these operations.

9.3.2 *Partnerlink* Initiation

A node n entering the network for the first time awaits the first *SCAMP Inview* message to initialize the *Partnerlink* manager. A list of up to $M/2$ candidate partners is requested from the contained node, retrieved at random

from its *mCache*. This is necessary as the local node's *mCache* needs time to fill with peers; relying on it for first partners comes with substantial startup delay.

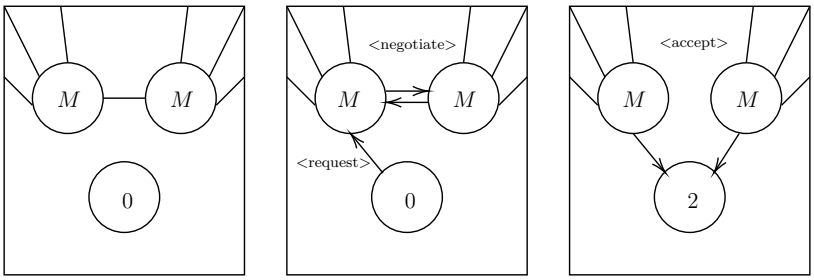

Figure 8: Process of a joining node splitting a partnership

Upon reception at node n, these nodes are all locked and sent a *Split* message. This performs the split operation discussed in Section 8.3, attempting to place the node in-between an existing partnership. The node also initializes M_c to $M_r * 2$, where M_r is the number of nodes received from the initial exchange. Node a receiving a *Split* message checks that no partnership nor lock is held against n - if not, the node forwards a *TrySplit* message to a random unlocked partner b, holding locks against it and n. Node b performs the same checks, additionally requiring that a is not locked. If the checks pass, it returns a *TrySplitSuccess* message, breaks the connection with its old partner and initiates with the new node. Node a receiving a *TrySplitSuccess* breaks similarly and unlocks the two nodes, after which the exchange is complete. Upon successful TCP connection, node n removes its lock on node a. The newly joined node has now gained two links, breaking only one in the process. If all *Split* messages respond successfully, the node will have created exactly M partnerships.

If node b fails its checks, it responds a *TrySplitFailure*. If node a receives a *TrySplitFailure*, fails its own checks or does not know another unlocked partner, a *SplitFailure* is forwarded back to the new node n, reducing M_c by 2 and unlocking the contacted node. This causes the node to immediately begin panicking.

9.3.3 *Partnerlink* Panicking

If a node does not satisfy $M_c = M$, it is described as *panicking*. A node checks its panic state after each update to M_c, from which we begin to look for alternative ways to create new links in the system.

- If $M_c = M - 1$, we cannot fill our empty partners by splitting, as we

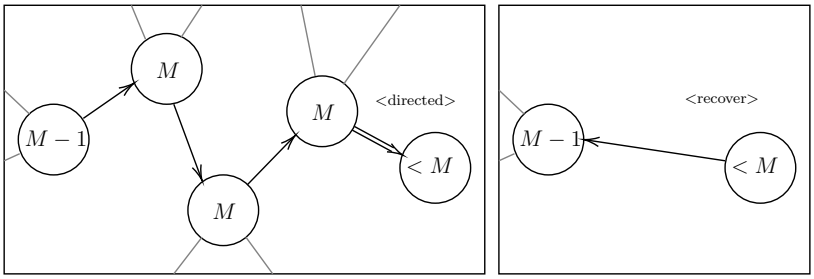

Figure 9: Gossiping and recovering a *Panic* message

would breach our limit. We instead gossip a *Panic* message containing our *NodeID* to a random partner, similar to the hypothesized mechanism behind *DONet* partnerships. This message is directed across the network towards panicking nodes with priority, as determined from their gossiped status. Otherwise, the message is directed towards any node that is not the original panicking node or the last hop. As a last resort the message is directed to any available node, or else dropped. If a panicking node receives a *Panic* message it initiates a connection with the contained node and increases its M_c by 1. The original node receiving this connection does the same, unless it has since stopped panicking, in which case the connection is rejected.

This joins two panicking nodes together directly and grants one new link each. This is mostly effective in very small networks, where nodes impacted by an improper exit are few hops apart. As networks grow and crystallize, the hops between panicking nodes will increase, worsening delay before node recovery. Note that locks do not influence the transmission of this message, as its result on a partner does not influence a node's local link count.

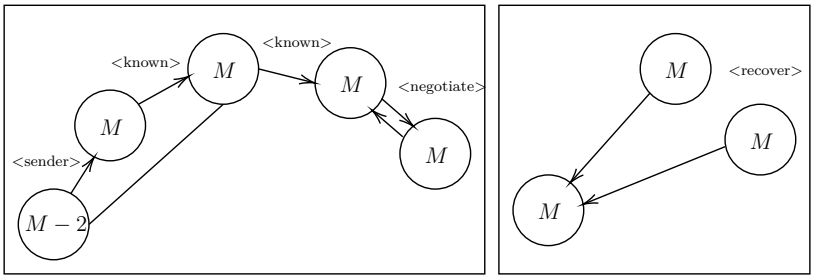

Figure 10: Gossiping and recovering a *PanicSplit* message

- If $M_c = M - 2$, we fall back to the splitting method. We gossip a *Pan-*

icSplit message containing our *NodeID* to a random partner - mechanically, this operates as a gossiped *Split* request. This message contains a field *LastHopOpinion*, initialized to *CANT_HELP*. The same direction priorities apply, although now targetting non-panicking nodes and with a restriction on gossiping to locked partners. If a node receives a *PanicSplit* with *LastHopOpinion CANT_HELP*, the node performs similar checks as in a regular *Split* - that we are not already partnered with nor hold any locks against the contained node. If the checks pass, the message is gossiped to a random unlocked partner with a *LastHopOpinion CAN_HELP*, and the contained node and random partner are locked; else, it is marked *CANT_HELP*.

If a node receives a *PanicSplit* with *LastHopOpinion CAN_HELP*, we perform the same checks with an additional requirement that we do not hold a lock against the last hop node. If these checks fail, we throw a *PanicSplitFailed* back to the last hop, who unlocks the contained node and partner. We then repeat our treatment of the message as if *LastHopOpinion* had been *CANT_HELP*. If these checks pass, we proceed as if from a successful *TrySplit* - a *PanicSplitSuccess* is gossiped back to the last hop, our partnership with it is closed and a new connection is initiated with the contained node. The node receiving the *PanicSplitSuccess* message switches connections similarly. The panicking node receiving the connections increases M_c by 1 each, unless it has since stopped panicking, in which case it rejects the connection.

This finds two relatively stable nodes within the network and places the panicking node between them. This is mostly effective in large networks, where the gossiped message will quickly find its way to nodes with no knowledge of the sender. In networks with node count $Nc \leq M + 1$, this message will never resolve. The resolution rate quickly improves from there.

- If $0 < M_c \leq M - 3$, we gossip both of the above messages at once.
- If $M_c = 0$, we should assume global disturbance amongst our membership peers. The standard *SCAMP* leave procedure should be followed to cleanly disconnect from the membership network in the membership manager, before recontacting the origin to gather a fresh partial view and begin new partnerships from scratch.

Each of the above messages contains a time-to-live which is checked at each receiving node. If the timeout expires, the message is dropped. It is the responsibility of each panicking node to resend messages after they time out.

These messages are gossiped to partners, though they could instead be gossiped to members. We suspect that gossiping to partners will guide messages towards nodes with a larger number of partnerships (i.e. those with more stable connections) allowing for more effective recovery, whereas the *mCache* may contain a greater concentration of already-poor connections. We cannot provide any proof of this theory, however, which would prove an interesting topic for future research.

9.3.4 *Partnerlink* Leaving

To avoid unneeded panics when disconnecting a node, it is intuitive to pair nodes back together, undoing the splitting we performed to enter. If we perform this at leave time, however, an unclean disconnect will still panic all children, adding M panicking nodes to the network - a substantial breakdown in overlay structure. Instead, we continually inform nodes of their *associated peer* alongside buffer maps. A clean leave then commands nodes to switch connections to the node they were previously informed of, and nodes detecting a failing node can switch automatically without external assistance. This mechanism allows for overlay repair without any additional control in most cases.

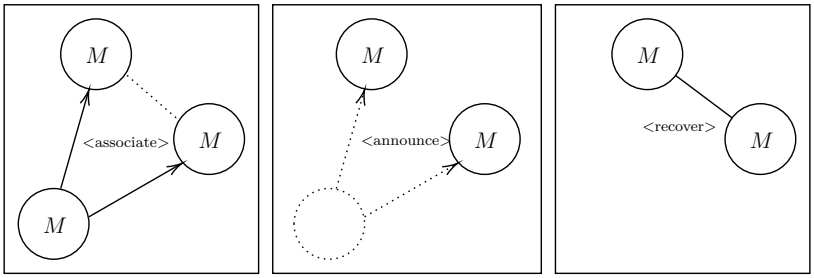

Figure 11: Associating peers and cleanly leaving the network

Nodes should be associated deterministically - that is, for as long as a node holds the same set of partners, the associated peer for each partner should not change. Given a set of partners $\{A, B, C, ...\}$, the associated peer for A is B, and for B is A. If there is an odd number of partners, the final peer is associated with a null *NodeID*. Peers receiving a buffer map should store the new associated peer corresponding to this partnership, replacing any prior peer. If the partnership fails or times out, the node should first attempt to connect to their associated peer. If this connection fails, the node was associated with null, or no association was ever provided, the node should reduce M_c by 1, entering a panic state.

A leaving node should send a *Leave* message to each of its peers before ending the connection. Each receiving node should attempt the above procedure regardless of the state of the partnership.

9.3.5 *Partnerlink* Switching

coolstreaming-spiked is designed for compatibility with both *DONet* and *New Coolstreaming* research models. To provide this support, we retain the switching system unique to *DONet*. At a set interval, each node should lock and perform a standard *Switch* procedure on a random node in its *mCache*, without changing M_c. Two additional randomly-chosen associated partners should be locked. If both new links from the *Switch* are successful, these two locked partners should be sent *Leave* messages, so that they reconnect to each other; else, the procedure is abandoned and the two partners are unlocked.

We move on to provide additional insight into the rationale of our design.

9.3.6 *Partnerlink* Rationale

The *Split* and *Panic* messages are near-identical to those hypothesized to act within the *DONet* model. Switching could also not be achieved without the inclusion of some similar mechanism. These are essential items for the output required of the partner component.

The *SplitPanic* system primarily acts as a caution against early failure when connecting to initial candidate partners. One downside of the *Split* joining model is the chance of collisions - there is some chance that two candidates contact the same node to perform two splits, which will be rejected under the locking mechanism. Network conditions may also lead to a node being impossible to contact. In either case, it is not unreasonable to expect a number of initially chosen nodes to be unreachable, resulting in a starting M_c far below M which would take a long time to recover with the vanilla *Panic* strategy. Whilst performance is not a priority for this model, there is still a minimum bar beyond which research analysis would become clouded by its limitations; such startup delay would pose problematic.

The new *associated peers* approach is ultimately easier to implement than the intuitive leave method associated with a splitting approach. Nodes would have to associate peers and forward them to nodes regardless, with the addition of some special case for nodes leaving without notification. An extra opportunity to apply peer selection algorithms also arises in the choosing pairs of peers to associate for filtering, which researchers may find beneficial. Thus, we consider this approach an improvement over the immediately

obvious solution.

Whilst the switching mechanism does provide compatibility, its impact on the model is limited compared to the more integral role the *mCache* takes in replacing partners in *DONet*. Research surrounding peer selection algorithms will therefore appear less impactful under *coolstreaming-spiked* - a more in-depth adaptation may also include the algorithm as part of the *associated peer* system to accentuate its effect.

We move on to the final component of the *coolstreaming-spiked* model - the stream manager.

9.4 Stream Manager

The stream manager is responsible for converting partnerships to parent-child subscriptions, pushing blocks as necessary, as well as the dissemination of buffer maps.

Each node regularly emits a buffer map to every partner. The syntax of these maps has already been discussed - however, an exception on valid syntax is made for when the node has just entered the network. Once buffer maps have been received from some defined minimum percentage of partners, the node must calculate an ideal starting index based on the known set of latest received blocks. Designating $H_{S_i,A}$ as the latest block received block for substream S_i at node A, the starting index at substream i is calculated as

$$I_{S_i} = \max\{H_{S_i,q} : q \in partners\} - T_p$$

where T_p is a constant to be introduced later. The ideal placement of the starting playout index is specific to the playout strategy and is an open research question. We expect that buffering systems should place playout at $\min\{I_{S_i}\}$ and begin playout once a minimum buffer percentage has been filled. Non-buffering approaches are more complicated - playout should be placed such that this buffer percentage will have been filled by the time playout reaches $\min\{I_{S_i}\}$. The actual ideal calculation for this will likely require additional information to be gossiped between nodes.

A new node has not yet receiving blocks in a given substream i should fill its latest block with some exceptional value, either a manually-filtered *null* or a value of -1, unless the node intends to make its first subscription on i to that partner. In this case, the latest block on this substream should be listed as $I_{S_i} - K$ to ensure the partner's stream manager begins transmission from the relevant point. The exceptional value should be filtered to ensure the node is never selected as a parent.

When a node n subscribes to a substream through their buffer map,

the partnership manager reads and stores the node's latest received block R_n. The manager attends to each subscribing partner with a round-robin strategy. At each step, if the node's buffers contains block with sequence number $R_n + K$, the manager pushes this block to node n and updates R_n accordingly, moving on to the next node. The stream manager should aim to fully saturate a node's outgoing bandwidth - as soon as one block finishes transmission, the next should begin. The stream manager will continue to push all blocks in the substream until the partnership or subscription ends; a parent node will not drop a child under any other circumstance.

For monitoring the service of substream j to child node A by parent p, two inequalities are defined:

$$\max\{|H_{S_i,A} - H_{S_j,p}| : i \leq K\} < T_s \tag{1}$$

$$\max\{|H_{S_i,q}| : q \in partners\} - H_{S_j,p} < T_p \tag{2}$$

Inequality (1) caps the distance between the most up-to-date child substream and the target parent substream. If this inequality does not hold, the parent has blocks we need that we are not receiving, implying issues with the connecting link. Inequality (2) caps the distance between the target parent substream and the most up-to-date substream we know across all partners. If this inequality does not hold, the parent is lagging behind the wider swarm, implying connection issues further upstream. At each buffer map and block reception, these inequalities are measured against each substream parent - if either fail, the parent is reselected. The replacement parent is selected randomly from all other partners meeting inequality (2) for the target substream - if no partner meets the inequality, the reselection is cancelled. If any reselection occurs, a new buffer map is immediately sent to the affected parents. A cooldown timer T_i is set on a per-substream basis to prevent repeated reselection and stabilize the overlay topology. Substreams with active cooldowns will not be checked.

The failure state for the stream manager depends on playout strategy. If buffering is incorporated, the node may fall behind the blocks available in the wider swarm. If playout does not halt, we may still find ourselves in a situation where all partners provide unsatisfactory connections. A number of heuristics could detect either case; a threshold over the filled percentage of the buffer is one simple example. Whatever the method, the stream manager should assume a widespread failure amongst its underlying partners, and initiate the same recovery mechanism as defined in the partnership manager.

9.5 Comparison of *coolstreaming-spiked* to our implementation

The implementation tested in Section 7 is not an implementation of *coolstreaming-spiked*; indeed, our results from those experiments have informed our design. They are, however, basically similar. The key differences are:

- Our implementation retains the *DONet* version of *SCAMP*, with all its oddities. We do not anticipate this aspect of the system to have any impact on performance, and the component outputs are the same.
- Our implementation does not include any locking mechanism - instead, more advanced link counting is used to resolve overlapping message streams without hiding nodes from overlay adjustment. The interactions that arise cannot be reasoned with in any structured form, and are very likely the reason for the de-syncing M_c we note in our results; the locking mechanism is proposed due to this experience. This will have an impact on results, as we expect *coolstreaming-spiked* to perform much more reliably.
- Our implementation does not make any use of TCP connections, instead relying on UDP either directly or through *OverSim*'s RPC support. This is related at once to the cut-down paper we worked from, that TCP support in *OverSim* is a somewhat hidden feature, and our own oversight. This negatively impacts our results, as TCP networking is essential in the design of *Coolstreaming*.
- Our implementation initiates playout similarly to the described buffering playout strategy, though no other buffering is performed. This strange hybrid approach, whilst not invalid, is unlikely to be seen in a real research model and should ideally instead align with one or the other.

The changes we have made to *coolstreaming-spiked* since our implementation aim to resolve the poor performance seen in our results. Although this invalidates our final performance figures, we are confident the changes we have made should improve rather than dampen the model.

10 Conclusion

In this paper, we discuss several research improvements that could be made on top of the popular P2P streaming network *New Coolstreaming*. The ob-

jective of the paper is to implement these improvements in simulation alongside a monolithic *AQCS* solution, taking measurements to quantify their impact on QoE. During implementation, we instead hit upon several key issues within the *New Coolstreaming* protocol for research purposes. The new research-oriented model *coolstreaming-spiked* is produced, with a stronger specification and less variable implementation. As part of this, *Partnerlink* allows nodes within the network to form wide-spanning partnership webs at any scale, without triggering any unnecessary churn or reselection.

11 Further Research

More investigation on *coolstreaming-spiked* should be performed ahead of its use in research. An updated simulation would prove beneficial to validate the issues identified with our first solution have been rectified by the changes since. Assuming this is successful, it would then be possible to perform the testing initially specified in our methodology. A mathematical model of the network, particularly *Partnerlink*, would also prove fruitful in pinpointing any further technical failures, and could produce optimizations that do not significantly impact our low-complexity requirements. Optimized calculations for the ideal starting playout index and proximity-based inequality 1 would round out the model, suitable for more avenues of research.

References

Awiphan, Suphakit, Zhou Su, and Jiro Katto. 2010. Tomo: a two-layer mesh/tree structure for live streaming in p2p overlay network. In *2010 7th ieee consumer communications and networking conference,* 1–5. https://doi.org/10.1109/CCNC.2010.5421709.

Baumgart, Ingmar, Bernhard Heep, and Stephan Krause. 2007. OverSim: a flexible overlay network simulation framework. In *Proceedings of 10th ieee global internet symposium (gi '07) in conjunction with ieee infocom 2007, anchorage, ak, usa,* 79–84. May.

Beraldi, Roberto, Marco Galiffa, and Hussein Alnuweiri. 2010. W-coolstreaming a protocol for collaborative data streaming for wireless networks. In *2010 ieee 30th international conference on distributed computing systems workshops,* 221–226. https://doi.org/10.1109/ICDCSW.2010.78.

Bonald, Thomas, Laurent Massoulié, Fabien Mathieu, Diego Perino, and Andrew Twigg. 2008. Epidemic live streaming: optimal performance trade-offs. *SIGMETRICS Perform. Eval. Rev.* (New York, NY, USA) 36, no. 1 (June): 325–336. ISSN: 0163-5999. https://doi.org/10.1145/1384529.1375494. https://doi.org/10.1145/1384529.1375494.

Budhkar, Shilpa, and Venkatesh Tamarapalli. 2017. Delay management in mesh-based p2p live streaming using a three-stage peer selection strategy. *Journal of Network and Systems Management* 26, no. 2 (August): 401–425. ISSN: 1573-7705. https://doi.org/10.1007/s10922-017-9420-5.

Bustamante, Fabian E., and Yi Qiao. n.d. Friendships that last: peer lifespan and its role in p2p protocols. In *Web content caching and distribution,* 233–246. Kluwer Academic Publishers. ISBN: 1402022573. https://doi.org/10.1007/1-4020-2258-1_16.

Carta, Alessandra, Marco Mellia, Michela Meo, and Stefano Traverso. 2010. Efficient uplink bandwidth utilization in p2p-tv streaming systems. In *2010 ieee global telecommunications conference globecom 2010,* 1–6. IEEE.

Chu, Yang-hua, S.G. Rao, S. Seshan, and Hui Zhang. 2002. A case for end system multicast. *IEEE Journal on Selected Areas in Communications* 20 (8): 1456–1471. https://doi.org/10.1109/JSAC.2002.803066.

Cohen, Bram. 2017. *The bittorrent protocol specification.* BitTorrent, February 2017; Digital. http://bittorrent.org/beps/bep_0003.html.

Couto da Silva, Ana Paula, Emilio Leonardi, Marco Mellia, and Michela Meo. 2011. Chunk distribution in mesh-based large-scale p2p streaming systems: a fluid approach. *IEEE Transactions on Parallel and Distributed Systems* 22 (3): 451–463. https://doi.org/10.1109/TPDS.2010.63.

Deering, Dr. Steve E. 1989. *Host extensions for IP multicasting.* RFC 1112, 1112, August. https://doi.org/10.17487/RFC1112. https://www.rfc-editor.org/info/rfc1112.

Floyd, Sally, Mark Handley, Jitendra Padhye, and Jörg Widmer. 2000. Equation-based congestion control for unicast applications. *SIGCOMM Comput. Commun. Rev.* (New York, NY, USA) 30, no. 4 (August): 43–56. ISSN: 0146-4833. https://doi.org/10.1145/347057.347397. https://doi.org/10.1145/347057.347397.

Friedman, Roy, Alexander Libov, and Ymir Vigfussony. 2015. Distilling the ingredients of p2p live streaming systems. In *2015 ieee international conference on peer-to-peer computing (p2p),* 1–10. https://doi.org/10.1109/P2P.2015.7328519.

Ghoshal, Jagannath, Lisong Xu, Byrav Ramamurthy, and Miao Wang. 2007. Network architectures for live peer-to-peer media streaming. https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1082&context=csetechreports.

Goh, Chin Yong, Hui-Shyong Yeo, Hyotaek Lim, Poo Kuan Hoong, Jay W. Y. Lim, and Ian K. T. Tan. 2013. A comparative study of tree-based and mesh-based overlay p2p media streaming. https://gvpress.com/journals/IJMUE/vol8_no4/10.pdf.

Guo, Yang, Chao Liang, and Yong Liu. 2008. Aqcs: adaptive queue-based chunk scheduling for p2p live streaming. In *Lecture notes in computer science,* 433–444. Springer Berlin Heidelberg. ISBN: 9783540795490. https://doi.org/10.1007/978-3-540-79549-0_38.

Hei, Xiaojun, Yong Liu, and Keith Ross. 2008. Understanding the start-up delay of mesh-pull peer-to-peer live streaming systems. https://eeweb.engineering.nyu.edu/faculty/yongliu/docs/xiaojun_delay.pdf.

Hei, Xiaojun, Yong Liu, and Keith W. Ross. 2007. Inferring network-wide quality in p2p live streaming systems. *IEEE Journal on Selected Areas in Communications* 25 (9): 1640–1654. https://doi.org/10.1109/JSAC.2007.071204.

———. 2008. Iptv over p2p streaming networks: the mesh-pull approach. *IEEE Communications Magazine* 46 (2): 86–92. https://doi.org/10.1109/MCOM.2008.4473088.

Ho, Cheng-Yun, Ming-Chen Chung, Li-Hsing Yen, and Chien-Chao Tseng. 2013. Churn: a key effect on real-world p2p software. In *2013 42nd international conference on parallel processing,* 140–149. https://doi.org/10.1109/ICPP.2013.23.

Ho, Cheng-Yun, Ming-Hsiang Huang, Cheng-Yuan Ho, and Chien-Chao Tseng. 2014. Bandwidth and latency aware contribution estimation in p2p streaming system. *IEEE Communications Letters* 18 (9): 1511–1514. https://doi.org/10.1109/LCOMM.2014.2343612.

Huang, F., B. Ravindran, and M. Khan. 2010. Nap: an agent-based scheme on reducing churn-induced delays for p2p live streaming. In *2010 ieee tenth international conference on peer-to-peer computing (p2p),* 1–10. https://doi.org/10.1109/P2P.2010.5569961.

Kang, Xiaohan, Juan José Jaramillo, and Lei Ying. 2012. Impacts of peer churn on p2p streaming networks. In *2012 50th annual allerton conference on communication, control, and computing (allerton),* 1417–1424. https://doi.org/10.1109/Allerton.2012.6483384.

Kim, Eunsam, Jinsung Kim, and Choonhwa Lee. 2018. Efficient neighbor selection through connection switching for p2p live streaming. *Journal of Ambient Intelligence and Humanized Computing* 10, no. 4 (January): 1413–1423. ISSN: 1868-5145. https://doi.org/10.1007/s12652-018-0691-9.

Kondo, Daishi, Yusuke Hirota, Akihiro Fujimoto, Hideki Tode, and Koso Murakami. 2014. P2p live streaming system for multi-view video with fast switching. In *2014 16th international telecommunications network strategy and planning symposium (networks),* 1–7. https://doi.org/10.1109/NETWKS.2014.6959253.

Kumar, R., Y. Liu, and K. Ross. 2007. Stochastic fluid theory for p2p streaming systems. In *Ieee infocom 2007 - 26th ieee international conference on computer communications,* 919–927. https://doi.org/10.1109/INFCOM.2007.112.

Lan, Shanzhen, Qi Zhang, Xinggong Zhang, and Zongming Guo. 2011. Dynamic asynchronous buffer management to improve data continuity in p2p live streaming. In *2011 3rd international conference on computer research and development,* 2:65–69. https://doi.org/10.1109/ICCRD.2011.5764085.

Li, B., S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang. 2008. Inside the new coolstreaming: principles, measurements and performance implications. In *Ieee infocom 2008 - the 27th conference on computer communications,* 1031–1039. https://doi.org/10.1109/INFOCOM.2008.157.

Li, Bo, Susu Xie, Gabriel Y. Keung, Jiangchuan Liu, Ion Stoica, Hui Zhang, and Xinyan Zhang. 2007. An empirical study of the coolstreaming+ system. *IEEE Journal on Selected Areas in Communications* 25 (9): 1627–1639. https://doi.org/10.1109/JSAC.2007.071203.

Li, Zhenjiang, Danny H. K. Tsang, and Wang-Chien Lee. 2010. Understanding sub-stream scheduling in p2p hybrid live streaming systems. In *2010 proceedings ieee infocom,* 1–5. https://doi.org/10.1109/INFCOM.2010.5462227.

Liang, Chao, Yang Guo, and Yong Liu. 2007. Hierarchically clustered p2p streaming system. In *Ieee globecom 2007 - ieee global telecommunications conference,* 236–241. https://doi.org/10.1109/GLOCOM.2007.52.

———. 2009. Investigating the scheduling sensitivity of p2p video streaming: an experimental study. *IEEE Transactions on Multimedia* 11 (3): 348–360. https://doi.org/10.1109/TMM.2009.2012909.

Liu, Chia-Yi, Kuochen Wang, and Yi-Ling Hsieh. 2010. Efficient push-pull based p2p multi-streaming using application level multicast. In *21st annual ieee international symposium on personal, indoor and mobile radio communications,* 2586–2590. https://doi.org/10.1109/PIMRC.2010.5671773.

Liu, Jiangchuan, Sanjay G. Rao, Bo Li, and Hui Zhang. 2008. Opportunities and challenges of peer-to-peer internet video broadcast. *Proceedings of the IEEE* 96 (1): 11–24. https://doi.org/10.1109/JPROC.2007.909921.

Liu, Yajie, Wenhua Dou, and Zhifeng Liu. 2004. Layer allocation algorithms in layered peer-to-peer streaming. In *Ifip international conference on network and parallel computing.* https://link.springer.com/chapter/10.1007/978-3-540-30141-7_25.

Lo Cigno, Renato, Alessandro Russo, and Damiano Carra. 2008. On some fundamental properties of p2p push/pull protocols. In *2008 second international conference on communications and electronics,* 67–73. https://doi.org/10.1109/CCE.2008.4578935.

Magharei, N., R. Rejaie, and Y. Guo. 2007. Mesh or multiple-tree: a comparative study of live p2p streaming approaches. In *Ieee infocom 2007 - 26th ieee international conference on computer communications,* 1424–1432. https://doi.org/10.1109/INFCOM.2007.168.

Magharei, Nazanin, and Reza Rejaie. 2006. Understanding mesh-based peer-to-peer streaming. In *International workshop on network and operating system support for digital audio and video.* https://dl.acm.org/doi/10.1145/1378191.1378204.

Magharei, Nazanin, Reza Rejaie, Ivica Rimac, Volker Hilt, and Markus Hofmann. 2014. Isp-friendly live p2p streaming. *IEEE/ACM Transactions on Networking* 22 (1): 244–256. https://doi.org/10.1109/TNET.2013.2257840.

Massoulie, L., A. Twigg, C. Gkantsidis, and P. Rodriguez. 2007. Randomized decentralized broadcasting algorithms. In *Ieee infocom 2007 - 26th ieee international conference on computer communications,* 1073–1081. https://doi.org/10.1109/INFCOM.2007.129.

Moltchanov, Dmitri. 2011. Service quality in p2p streaming systems. *Computer Science Review* 5 (4): 319–340. ISSN: 1574-0137. https://doi.org/https://doi.org/10.1016/j.cosrev.2011.09.003. https://www.sciencedirect.com/science/article/pii/S1574013711000219.

Nanao, Sho, Hiroyuki Masuyama, Shoji Kasahara, and Yutaka Takahashi. 2012. Effect of node churn on frame interval for peer-to-peer video streaming with data-block synchronization mechanism. *Peer-to-Peer Networking and Applications* 5 (3): 244–256. ISSN: 1936-6450. https://doi.org/10.1007/s12083-011-0120-8. https://doi.org/10.1007/s12083-011-0120-8.

Nguyen, Anh Tuan, Baochun Li, and Frank Eliassen. 2010. Chameleon: adaptive peer-to-peer streaming with network coding. In *2010 proceedings ieee infocom,* 1–9. https://doi.org/10.1109/INFCOM.2010.5462032.

Pal, Kunwar, Mahesh Chadra Govil, and Mushtaq Ahmed. 2017. Comparative analysis of utilization based hybrid overlay for live video streaming in p2p network. *International Journal of Intelligent Engineering & Systems* 10 (3).

Pouwelse, Johan, Pawe Garbacki, Dick Epema, and Henk Sips. 2005. The bittorrent p2p file-sharing system: measurements and analysis. In *Lecture notes in computer science,* 205–216. Springer Berlin Heidelberg. ISBN: 9783540319061. https://doi.org/10.1007/11558989_19.

PPLive. 2008. Pplive homepage. Accessed May 2, 2024. https://web.archive.org/web/20080617091219/http://www.pplive.com/.

Ramzan, Naeem, Hyunggon Park, and Ebroul Izquierdo. 2012. Video streaming over p2p networks: challenges and opportunities. ADVANCES IN 2D/3D VIDEO STREAMING OVER P2P NETWORKS, *Signal Processing: Image Communication* 27 (5): 401–411. ISSN: 0923-5965. https://doi.org/https://doi.org/10.1016/j.image.2012.02.004. https://www.sciencedirect.com/science/article/pii/S0923596512000331.

Sandvine. 2024. *The global internet phenomena report march 2024.* Digital, March. https://www.sandvine.com/phenomena.

Sanghavi, Sujay, Bruce E. Hajek, and Laurent Massoulié. 2006. Gossiping with multiple messages. *CoRR* abs/cs/0612118. arXiv: cs/0612118. http://arxiv.org/abs/cs/0612118.

Seeling, Patrick, and Martin Reisslein. 2012. Video transport evaluation with h.264 video traces. *IEEE Communications Surveys & Tutorials* 14 (4): 1142–1165. https://doi.org/10.1109/SURV.2011.082911.00067.

Sina, Majid, Mehdi Dehghan, Amir Masoud Rahmani, and Midia Reshadi. 2020. Wideplive: a coupled lowdelay overlay construction mechanism and peerchunk prioritybased chunk scheduling for p2p live video streaming. *IET Communications* 14, no. 6 (April): 937–947. ISSN: 1751-8636. https://doi.org/10.1049/iet-com.2019.0618.

SopCast. 2019. Sopcast homepage. Accessed May 2, 2024. https://web.archive.org/web/20191009112802/http://www.sopcast.org/.

Stutzbach, Daniel, and Reza Rejaie. 2004. Towards a better understanding of churn in peer-to-peer networks. https://www.cs.uoregon.edu/Reports/TR-2004-006.pdf.

UUSee. 2007. Uusee homepage. Accessed May 2, 2024. https://web.archive.org/web/20071013072457/http://www.uusee.com/.

Vassilakis, Constantinos, and Ioannis Stavrakakis. 2010. Minimizing node churn in peer-to-peer streaming. *Comput. Commun.* (NLD) 33, no. 14 (September): 1598–1614. ISSN: 0140-3664. https://doi.org/10.1016/j.comcom.2010.04.014. https://doi.org/10.1016/j.comcom.2010.04.014.

Vishnumurthy, Vivek, and Paul Francis. 2007. A comparison of structured and unstructured p2p approaches to heterogeneous random peer selection. In *Usenix annual technical conference,* 309–322.

Wang, Lei, Dengyi Zhang, and Hongyun Yang. 2013. Qos-awareness variable neighbor selection for mesh-based p2p live streaming system. In *2013 ieee third international conference on information science and technology (icist),* 1197–1201. https://doi.org/10.1109/ICIST.2013.6747752.

Wang, Yongzhi, Tom Z.J. Fu, and Dah Ming Chiu. 2008. Analysis of load balancing algorithms in p2p streaming. In *2008 46th annual allerton conference on communication, control, and computing,* 960–967. https://doi.org/10.1109/ALLERTON.2008.4797662.

Wu, D., Y. Liu, and K. Ross. 2009. Queuing network models for multi-channel p2p live streaming systems. In *Ieee infocom 2009,* 73–81. https://doi.org/10.1109/INFCOM.2009.5061908.

Xie, Susu, Bo Li, Gabriel Y. Keung, and Xinyan Zhang. 2007. Coolstreaming: design, theory, and practice. *IEEE Transactions on Multimedia* 9 (8): 1661–1671. https://doi.org/10.1109/TMM.2007.907469.

Zhang, Jianwei, Wei Xing, Yongchao Wang, and Dongming Lu. 2014. Modeling and performance analysis of pull-based live streaming schemes in peer-to-peer network. *Computer Communications* 40:22–32. ISSN: 0140-3664. https://doi.org/https://doi.org/10.1016/j.comcom.2013.12.002. https://www.sciencedirect.com/science/article/pii/S0140366413002752.

Zhang, Meng, Qian Zhang, Lifeng Sun, and Shiqiang Yang. 2007. Understanding the power of pull-based streaming protocol: can we do better? *IEEE Journal on Selected Areas in Communications* 25 (9): 1678–1694. https://doi.org/10.1109/JSAC.2007.071207.

Zhang, Xinyan, Jiangchuan Liu, Bo Li, and Y.-S.P. Yum. 2005. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *Proceedings ieee 24th annual joint conference of the ieee computer and communications societies.* Vol. 3, 2102–2111 vol. 3. https://doi.org/10.1109/INFCOM.2005.1498486.

Zhao, Jian, and Chuan Wu. 2012. Characterizing locality-aware p2p streaming. *J. Commun.* 7:222–231. https://www.jocm.us/index.php?m=content&c=index&a=show&catid=39&id=89.

Zink, M., and A. Mauthe. 2004. P2p streaming using multiple description coded video. In *Proceedings. 30th euromicro conference, 2004.* 240–247. https://doi.org/10.1109/EURMIC.2004.1333377.