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

Martin Symons

May 3, 2024

**Abstract**

abstract !!!

## 1 Introduction

Within the past ten years, video streaming traffic across the internet has exploded. Traditional client-server architectures places 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 *Coolstreaming* in 2004 these networks became defacto for IPTV streaming, and with their utility proven research accelerated in pace and complexity. Industry, however, did not pick up. Since the shutdown of New *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 that interrupt our efforts, which we discuss and document. *coolstreaming-spiked* is proposed as an alternative research-oriented model with a strict specification and these errors corrected. As part

of this effort, we produce the novel *Partnerlink* partnership algorithm, allowing partnership webs of any scale with minimal link breakage or video stream interruptions and including a self-healing *panic* mechanism to recover from unexpected failure.

The rest of the paper is structured as follows. Section 2 provides a background into P2P livestreaming and defines the research we intended to investigate. This research is compiled into three concrete models in Section 3. Section 4 timelines our attempts to produce these models, of which one is completed, tested for effectiveness in Section 5. Section 6 reveals the problems found in *New Coolstreaming* throughout our implementation attempts. A replacement network *coolstreaming-spiked* is formally specified in Section 7. We conclude in Section 8.

# 2 Literature Review

We review prior work to build an understanding of 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 universal 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 2.1 describes the analytical model nodes in many networks are built from, and its four constituent components. Heuristics used across the field for performance analysis are established in Section 2.2, alongside a discussion of our focus on churn. Components of the prior analytical model are deconstructed for churn resistance improvements in Sections 2.3, 2.4 and 2.5, investigating overlay structure, peer selection strategies and chunk schedulers respectively. We introduce *New Coolstreaming* and its older form *DONet* in Section 2.6, and pick compatible improvements from our investigations in Section 2.7. Section 2.8 concludes with a discussion on monolithic architectures.

## 2.1 Peer-to-Peer Streaming Fundamentals

Recent explosions in demand mean video streaming consumes 38% of total internet traffic (Sandvine 2024). Despite global innovations in CDN and edge computing technologies, an imbalance still arises in client and server availability, forming a bandwidth bottleneck (Ramzan, Park, and Izquierdo

2012). Content 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 QoS is measured mostly against upload speed and piece availability. These two priorities can be trivially solved 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, additional constraints on strict block ordering, end-to-end delay and a moving deadline that must be synchronized between peers are introduced necessitating unique network designs (Liu et al. 2008).

Friedman, Libov, and Vigfussony 2015 surveys past solutions and expresses their structure in four key modules:

- The *player module*, which consumes the received video signal to display to the user. The current playout index and any halting/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 module to retrieve chunks from the swarm ahead of playout. Depending on the strategy, discussed in 2.5, 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 an initial list retrieved from the origin node or through gossiping.
- The *network module*, abstracting the lower-level message management and applying any necessary encryption.

A node will therefore spin up a network module, learn of some nodes in its *known* buffer, adapt these to *neighbors* based on some criteria, and begin exchanging 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.

## 2.2 Performance Heuristics and Churn

Measurements can be broadly classified into two categories: service integrity, covering aspects that apply generally to any media playback, and audiovisual, describing the encoding-specific characteristics in a more subjective manner (Moltchanov 2011). These are primarily taken at the player module, where the observed values will directly impact user QoE. As our simulation study does not support analysis of the actual video playback, we focus for now on the former category.

Hei, Liu, and 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 the playout deadline. *Start-up latency* or *start-up delay* measures between time of entry to first block playout, being the first metric impacting user QoE upon network entry, and ranks just below in important. *Playback delay* or *end-to-end delay* measures the time between a block entering the network until playout at a given node. The importance of this metric depends on the interactivity of a stream: real-time purposes may place this above even start-up delay, whilst others may prove highly tolerant.

These measures correlate on various levels with user QoE, focusing mainly on characteristics of playout. Additional checks on *bandwidth utilization* and *messaging overhead* can provide additional insight into the efficiency of the overlay and streaming modules. In our paper, however, we focus on wider aspects of churn and topology for QoE improvement.

In a P2P system, nodes are not supervised or controlled as to their participation in the swarm. They are allowed to leave or join at any time without constraint or notification. Particularly under certain overlays, though, the QoE of any node's partner is reliant on that node's continued relationship - as that 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, which mainly suffer a hit to startup delay (Ho et al. 2013). Under the additional constraints of livestreaming, however, the impact expands to cover all measures of performance. Nanao et al. 2012 finds that the mean on/off churn time of a simulation noticably damages forwarding interval and end-to-end delay for as long as the search time for a new parent is above 100ms. In even the fastest-finding 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 worsens up to 17.2% under high churn.

We bring particular note to the conclusions of Vassilakis and Stavrakakis 2010. Correlations are drawn between the playout quality and QoE at a node to its chance to churn, harnessing a model revealing hidden interactions underlying many popular research systems. In other words: 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), the result of many users choosing to quickly leech the file and disconnect whilst others continue to share their resources with the network to improve the health of the swarm. This aspect of churn has since become a research topic in its own right to optimise file-sharing swarms (Bustamante and Qiao, n.d.); whether this characterization of P2P streaming churn should be as influential in simulating streaming networks remains an open research question, though its compounding nature leads to its prioritization in our research question. Real-world investigation of *New Coolstreaming* (discussed in 2.6) solidifies our interest in this area, describing churn as "*the most critical factor that affects the overall performance.*" (Bo Li et al. 2007)

## 2.3 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 amongst peers. this sucks but this source Ghoshal et al. 2007 has more chunkyspread also contains citations for this

Following this, designers moved to building tree overlays across the traditional networking stack. Tree overlays are structured, with each node forwarding the complete data stream to a number of children originating from a single server at the top of all nodes. Goh et al. 2013 finds through comparative simulations that playback delay in a tree structure is up to 114% lower than in a mesh, observing similar improvement in playout rate. We doubt the validity of these results, however - the paper proposes playback delays much shorter than the RTT of even a 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, as the tree quickly widens as its height increases. This comes at the expense of enormous susceptibility to peer churn, since a departing peer temporarily detaches all child nodes from the tree until the overlay recovers (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

cite

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 transmitted into 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 not even buffer, only degrade in quality. Analysis at the time suggested these topologies traded reliability and churn resistance for a more complex design and intensive media decoding requirements (Ghoshal et al. 2007). The former is not a concern for a production network; 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. We would now consider that a multi-tree overlay represents a guaranteed improvement over single-tree.

Still, mesh networks have become the norm in recent developments. Mesh networks form a randomly connected unstructured overlay of parent-child relationships. Each node holds multiple parents and multiple children. This resembles a multi-tree; however, nodes can now freely move anywhere in the swarm, using any other node as a source for blocks, without regard for the wider network structure. 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% in all experiments, whilst tree approaches are less efficient and more sensitive to improperly tuned parameters. This leads to at least one additional quality level being received at each node even under ideal tuning. Utilization is also shown to trough significantly under churn in a tree, whilst remaining high 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 only consider mesh networks due to their well-supported 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 2.8.

## 2.4 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 and provide inherent load balancing (Wang, Zhang, and Yang 2013), though this comes at a cost. Popular 2010's network *PPLive* was noted to suffer very high startup delay. This delay was caused by a failure to quickly accrue quality nodes - the first random batch usually contained nodes with insufficient bandwidth or whom cannot be reached due to network constraints. After 150 seconds, nodes only connected to four parents Hei, Liu, and Ross 2008b. PPLive's popularity proves that this approach can be sufficient, but alternatives should be considered.

we should replace this source with something About random

QoS-aware methods prioritize nodes by some network heuristic before supplying for connection. OCals tests RTT both during construction and during scheduling by proxy through TCP-Friendly Rate Control (Floyd et al. 2000), passing equal candidates for which the new node can provide good QoS as for 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 then selected based on matching upload/download bandwidth and lowest available layer count. This strategy massively improves playout rate at scale whilst boosting quality satisfaction up to 24.7% in smaller networks, when compared to a simpler bandwidth-based mechanism FABALAM (Liu, Dou, and Liu 2004).

is this the right term?

Locality-aware methods are the black sheep of this herd, primarily aiding underlay efficiency instead of overlay performance. They reduce end-to-end delays to a minimal, 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 choice can be amplified

by exercising the strategy as 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 by their locally perceived upload, propagation delay, lag and buffer before initiating connections. Throughout a node's lifetime, a final strategy compares all ancestors to slowly drive performance as playout continues. This thorough application of principles reduces startup delay by 10% in all cases, and playback delay and recovery 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.

## 2.5 Chunk Schedulers

Chunk schedulers under mesh topologies are generally split into pull, push and hybrid-push-pull approaches, depending on which node takes 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 to request blocks from its parents, updated from time to time with the status of their buffer maps. The process of neighbor selection, buffer map reception, chunk selection, request and finally delivery adds up under this approach, resulting in high end-to-end delay (Hei, Liu, and Ross 2008a); therefore, whilst most early overlays incorporate this basic idea, they are rarely proposed for low-latency real-time 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. This opinion is expressed in part by Liu, an essential figure in the field

Regardless, push schemes have seen some uptake. Under this arrangement, the scheduler is run on the parent node to automatically push 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

find the dissertation we need to talk about this?

receive. Parents must 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 new methods suggested to reduce message overhead and implementation complexity. This still does not appear any more optimal than the aforementioned pull schemes, whilst requiring more complex mathematical models to run.

To simplify things, 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 2.8.

## 2.6 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 corporate 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, whilst its commercialization 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 description and 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* provides a larger random list of candidates from which the entering node can initiate partnerships and request its first blocks. The deputy also gossips a membership message 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 along the same connection. Buffer maps are represented as a single boolean array 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 before 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 who sends a message on its behalf. Occasionally, the partnership manager ranks all partners by a score, calculated as either the average blocks uploaded or 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 total 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 4, and we summarize here with additional information from hindsight. 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.

Component-wise, 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 push blocks forever until the child explicitly unsubscribes or disconnects. The relationship is not broken under any other circumstances.

A new method for calculating the initial block number is defined, which was not covered as part of *DONet*, alongside two new heuristics to detect improper service from a node. These heuristics take the form of inequalities: if these inequalities do not hold, a new partner is swapped in for whom they do.

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.

## 2.7 Coolstreaming-Compatible Churn Improvements

Based on the key components identified above, we now discuss candidate churn-focused improvements that could be suited for *New Coolstreaming*.

*New Coolstreaming*'s overlay management broadly overlaps with Chameleon's, whose QoS-aware peer selection strategy is well-tested and churn resistance improvement has been discussed prior. 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. We aim to adapt this strategy 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 reselec-

tion 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. 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 the 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 deputy/bootstrap system should integrate well, although the downloading plan cannot be used due to the hybrid-push-pull approach. We hope to incorporate both of these novel components into our solution.

did we specify coolstreaming bootstrap nodes properly i forget

## 2.8 Monolithic Single Solutions

We now discuss monolithic solutions developed since the introduction of *New Coolstreaming*. 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 then taken into account when finding long-lived, high-contribution peers in both components. A node's actions during the chunk scheduling phase will therefore impact its future position in the overlay. Almost all actions are buffered, prioritized and acted upon after a short period, granting a node more intelligence about its overlay position and service. Nodes using the origin as a parent are known as *root peers*, and form a novel load-balancing network to equalize block dissemination across 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 minimal improvement. No analysis is performed directly on the impact of churn, however, which we would aim to investigate.

WidePLive's key disadvantage is its complexity. The reference implementation in OMNeT++ comprised 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. AQCS (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). 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 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 logical flow of 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 a new point of comparison beyond a simple upgrade to *New Coolstreaming*.

# 3 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.
- The hierarchical *AQCS* model representing the modern monolithic approach.

We aimed to pit three models against each other in various QoS tests to gather specific numbers on the improvements between each. Initial tests

we discussed these in lit review, replace

which?

would be run on New Coolstreaming, widely considered the last popular IPTV P2P system and a good benchmark . We would then extend this with fitting modular improvements for further measurements. were particular targets, because . Finally, we expected to implement a monolithic model, , to compare the capacity of incremental improvements versus the design of a single, deeply-coupled architecture.

As a simulation environment, we chose OverSim. 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 was never necessary.

Statistics would be presented with the built-in OMNET++ visualization tools.

we probably already explained this in the lit review

name some shit

blah

name

needs expansion. we could mention dates?? what do people normally put here

# 4 Implementation

We based our initial experiments on New Coolstreaming as described in B. Li et al. 2008. We quickly ran into problems - whilst the paper describes the stream manager and buffer map exchange in great detail, little time 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 and some key equations to system function. 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 the 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 with DONet/Coolstreaming as groundwork. We thereby set about an implementation of this more primitive design.

The final DONet implementation completed following two weeks of work. This network was similarly not ideal, though the cause was mostly banal - New Coolstreaming strips the buffer, scheduler and related messaging completely, so we saw no need to optimize these components. More worryingly, the partnership manager collapsed quickly under even minimal churn, discussed later in . Still, this constituted enough the groundwork needed to continue.

Returning with wiser eyes, we found that our architecture still did not align with the basic modules in New Coolstreaming. This proved troubling. As discussed later in section 6.6, DONet has clarity problems of its own when describing parts of other systems, and we noticed 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 came through without a hitch, 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 meshed well with the stream manager, 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 comparisons on QoS benefits.

# 5 Results and Analysis

In testing, we begin with two origin nodes and join 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 15.

We first test our model on a simple underlay without packet dropping, bandwidth limitations or timeouts. We find that our partnership mechanism works well in this environment, averaging 14.89 partners across all nodes over the entire measurement period. No node is ever seen with below 11 partners. We also, however, find our potential partner count $M_c$ to be the same value, as our failure mechanisms are almost never engaged in this ideal environment. We find that our model maintains a satisfactory number of parent-child relationships with an average of 9.30. Despite this, our playout is poor: only 61.58% of blocks are received before playout time. Our efficiency is reasonable - of all received blocks, 1.92% are duplicates and 4.645% are outside the bounds of the receiver's buffer, almost all being too old.

there is a fucking term for this i know

We test again on a realistic underlay simulating the best-effort internet, with bandwidth limitations, packet loss and queuing. Under this environment, the partnership mechanism begins to fail: despite recording an average

$M_c$ of 12.94, our actual partner count averages 1.78. We explore this and the resultant changes made to *Partnerlink* in section 7.5. Surprisingly, our playout performs very similarly - nodes maintain an average of 9.10 parents and a playout rate of 62.00%. This is despite a major hit to efficiency - 6.62% of all received blocks are now out of bounds, and 11.9% are now duplicates. As we believe the failure in the partnership manager activates slowly over a node's lifespan, this likely means that our partnerships are very long lived; transmitted blocks must be across relationships made before the failure. The mechanics that allow playout rate to remain so high, however, are unknown.

# 6 Discussion

*What went wrong?* New Coolstreaming is not unique as an overlay; no features within should prove particularly challenging to implement. In this section, we explore the Coolstreaming family as a whole, and illuminate the many challenges faced in their implementation amiss in the papers themselves.

## 6.1 The Coolstreaming Family Tree

We have so far regarded Coolstreaming as a dyad of 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. However, they have been proposed under *four* different names across *four* papers:

- As *DONet* and *Coolstreaming*, 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.

Note that the name *Coolstreaming* is intended by this final paper, but *The New Coolstreaming* became the colloquial model name as a result of its title. Only the first paper contains the old model we have discussed as

*DONet/Coolstreaming* - the others all specify *New Coolstreaming*, despite the name differences.

The name *Coolstreaming* legally refers both to the older *DONet* model and the newer *New Coolstreaming* model. The confusion that results is obvious. Kondo et al. 2014 describes the *SCAMP* membership protocol, the push-pull mechanism and the bootstrap node as part of the same model, despite *SCAMP* being specific to the mesh-pull *DONet*. Beraldi, Galiffa, and Alnuweiri 2010 makes much the same error. 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*. Further examples still can be found of correct prose, but with Xie et al. 2007 being cited in X. Zhang et al. 2005's place, or vice versa.

It is interesting to note that the papers themselves appear to have issues keeping the versions straight: 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 local to *New Coolstreaming* only - hence our initial hesitance to visit *DONet*.

This escalates considering the final three papers. Xie et al. 2007 is the canon definition of *New Coolstreaming*, 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 analysis: 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.

This duplication means each paper opens with a definition of *New Coolstreaming*. These are not summaries - the *Coolstreaming+* specification is shortest yet still clocks in at two and a half pages. The majority of this text is word-for-word identical between papers, or at best reworded. The membership and partnership managers, though, have their specifications cut down completely, no longer parsing as a working system. For instance, connection management is resolved in Xie et al. 2007 by an off-handed 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 paper, despite these papers claiming to *"describe the architecture and components in the new Coolstreaming system"*.

Luckily, we appear to be the only researchers to fall into this trap. Resulting development time was certainly extended, but our criticisms of the

paper and our final solution remain valid.

## 6.2 What does Coolstreaming need to be?

flow this

The *Coolstreaming* family has been replaced in production by monolithic solutions like . *Coolstreaming*'s usage now lies entirely in the research domain - used as a base for reproducible results, a testbed for new modular features or a point of comparison between larger monolithic models. In all of these cases, consistency and ease of development take priority over QoS performance.

mention wio-ta-diof

The issues mentioned so far are a sign that *Coolstreaming* was never 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* was developed as a commercial system first and a research paper second. This may explain the missing features or heavy modifications of existing solutions - the system was still under active maintenance as the papers were written, and so work had to be quick moreso than correct.

In the next section, we introduce a new model to the *Coolstreaming* family designed for research purposes, outlining our priorities and rationale in doing so.

## 6.3 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 any 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 long chains of dropped-and-replaced partnerships before the overlay settled and worsened the impact radius of churn. The move to *New Coolstreaming* came with the demand to maximize partnership lifetimes, as churn would interfere with the push mechanism and result in wasted blocks being sent to a peer, totally invalidating this approach.

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

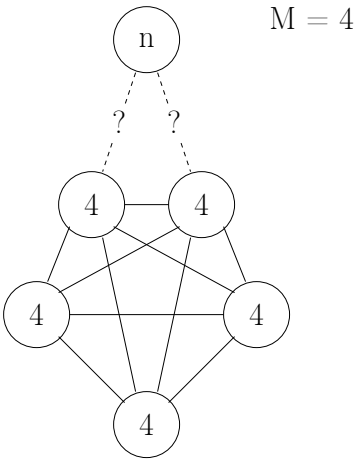


Figure 1: Node $n$ cannot easily join this fully-connected network.

that $M$ must now be divisible by 2. Since *DONet* uses $M = 15$ as example, this cannot be the implementation as intended by the original authors.

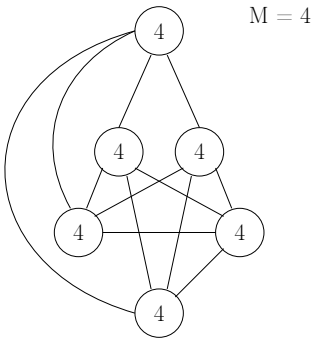


Figure 2: A solved topology, as according to the pairing method.

We can at least infer some details from *DONet*. 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, which would also justify the use of *SCAMP* as opposed to a more standard partial view aggregator. We include a similar strategy as part of our implementation; we discuss this in detail later in section 7.3.3.

That said, the phrasing of the scheduled switching as established *"with nodes randomly selected from [the] mCache"* implies a more direct communi-

cation. In this case, it is unknown how *DONet* or *New Coolstreaming* solve this problem.

## 6.4 Choosing New Partners

The two inequalities measuring connection quality are also applied to any candidate partners following a detected failure. Inequality 1 limits the gap between a node's buffers and the parent's. If we apply this to candidates after failing because of this inequality, we will filter out any nodes further ahead than our old node, slipping further back in our buffer. Assuming we never change our playout position, this new node will not be able to service our entire buffer area.

Furthermore, the nodes seen within this inequality will mostly 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 collapsing entirely. In our model we disabled inequality 1 when measuring candidates, and performance resumed - *New Coolstreaming* must use some other unspecified mechanism to avoid this.

## 6.5 Requesting the Block

*New Coolstreaming* specifies an algorithm to compare all buffer maps from a node's first partners and calculate the to request. The method through which this is done is somewhat obscure - the subscription map only communicates on-off booleans. We assume this is achieved by manipulating the latest blocks shown to each node to suggest the node has already received up to that block, despite having just joined. This requires some careful planning to execute without triggering subscriptions from nodes but is a clever usage of the limited data structure at play.

A feature unmentioned here is the behaviour-of and restrictions-on the playout index. Neither *Coolstreaming* implementation specifies a recovery mechanism - nodes are assumed to never fall completely behind the swarm. This leaves us two options - either the playout index must never buffer or pause after sync, and continue to play forward on encounter with missing blocks, or the recovery mechanism is left undefined. Given that there is a startup delay listed amongst the results, we can assume the latter. If the stream does include buffering or the node is suffering poor local performance, though, the need to recover is inevitable. There are many options for approach that impact performance, and one best solution should ideally be provided.

## 6.6 Production Optimizations

*DONet* defers to *SCAMP* for membership connection management, before overruling it on several counts. Where *SCAMP* automatically adjusts membership count in sync with the size of the system, *DONet* enforces a hard limit of $M$ nodes, usually 15. The indirection mechanism is implemented only in primitive form, 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 guarantees. Whilst we have not been able to perform experiments to prove this, we hypothesize that these restrictions therefore make no negative impact on the network, included to reduce message overhead.

we need to establish ahead of time the fanout proofs of SCAMP etc.

Other optimizations are less successful. *SCAMP* specifies a leasing system to remove dead nodes from the pool, which *DONet* accepts. However, in the case that the partnership manager finds a partner that fails to send a buffer map in a given period, 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 layers of indirections involved in membership management, 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. *SCAMP* notes that 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 its effect on network topology.

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, these changes represent a problem: we fall back to some existing research solution for a problem and then brain slug it for some slight gain, in the process tampering with that solution's proven performance. This is convenient for the original researcher, but requires careful cross-referencing of the two papers for accurate reproduction. New implementation of the model therefore becomes strenuous, and inaccuracies made much more likely.

is this actually a good turn of phrase

we need to flow this

# 7 Introducing *coolstreaming-spiked*

*coolstreaming-spiked* is a new iteration of *Coolstreaming* bringing the two models into synthesis. Research usage is targeted through an easier implementation, 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.

## 7.1 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 $(nK + i)$ where $n$ is a positive integer from zero to infinity. For instance, assuming $K = 4$, substream 2 would contain blocks with ID $\{2, 6, 10, 14, ...\}$.

## 7.2 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.

SCAMP does not describe any starting topology. A bootstrapped SCAMP network must contain at least two nodes engaged in a shared 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.

## 7.3 Partnership Manager and *Partnerlink*

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*.

this doesn't mean anything

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 the node to have most recently received block 39 on substream 3, and prepare a subscription for the node 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 7.3.3 and 7.4. 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 should be considered *failed* - the partner connection is removed and $M_c$ (discussed later) is reduced by 1.

### 7.3.1 *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 base 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. This basic network could be comprised of two guaranteed-trustworthy origin nodes, or built cooperatively with potentially-malicious client nodes.

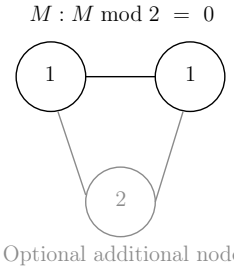


Figure 3: The base topology from which *Partnerlink* must operate.

Each node tracks an additional set of variables. We maintain a set of *locks* against *NodeID*s for which operations are still in progress, to protect the system when a node receives several requests regarding the same node. We also maintain an *anticipated* partner count $M_c$, tracking what our partner count will become if all ongoing 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.
- **Management**, allowing the overlay to adapt after a failed partnership.
- **Panic**, allowing an unhealthy node to recover back to $M$ connections.

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

### 7.3.2 *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. Upon reception at node $n$, these nodes are all locked and sent a *Split* message. This performs the split operation discussed prior, attempting to

place the node in-between an existing partnership. The node also sets $M_c$ to $M_r * 2$, where $M_r$ is the number of nodes received from the initial exchange. A 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 the connection similarly and unlocks the two nodes, after which the exchange is complete. Upon 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 $n$ which reduces $M_c$ by 2 and unlocks the contacted node. This causes the node to immediately begin panicking.

### 7.3.3 *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 $M_c$. 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 would breach our limit. We instead gossip a *Panic* message containing our *NodeID* to a random partner. This message is directed across the network towards panicking nodes with priority, as determined from their buffer maps. Otherwise, the message is directed towards any node that is not the original panicking node or the last hop. Otherwise, the message is directed anywhere, 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 it rejects the connection.

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 very few hops together. As networks grow and stabilize, 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.

If $M_c = M - 2$, we fall back to the splitting method. We gossip a *PanicSplit* message containing our *NodeID* to a random partner. This message contains a field *LastHopOpinion*, initialized to *CANT_HELP*. The same direction mechanism is applied, although with priority now given to non-panicking nodes and a ban 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 check 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 recurse and treat 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 major 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 this 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 bias recipients towards those 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 - this would prove a fruitful future research question.

### 7.3.4 *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 via messaging, however, failing nodes will still panic all of their nodes, adding $M$ panicking nodes to the network - a substantial overlay breakdown. Instead, we continually inform nodes of their *associated peer* when exchanging buffer maps. A clean leave then commands nodes to switch connections to the node they have already been informed of, and nodes detecting timeouts or a failing node can switch automatically without external assistance. This mechanism allows for overlay repair without any gossiped messages in most cases.

Nodes should be associated deterministically - that is, for as long as a node holds the same 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 an exceptional *NodeID* to be caught later. 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, or the node was associated with the exceptional *NodeID*, 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 then immediately attempt the above procedure regardless of the state of the partnership.

### 7.3.5 *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 perform a standard *Switch* procedure on a random node in its *mCache*, without changing $M_c$. Two randomly-chosen associated partners should be locked. If both new links are successful, these two partners should be sent *Leave* messages, so that they reconnect to each other; else, the procedure is abandoned and the two partners are unlocked.

### 7.3.6 *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 some similar mechanism. These are essential items for the minimum 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* model is the chance of collisions - there is some chance that two candidates contact the same node to perform two splits. Network conditions may also lead to a node being impossible to contact. In either case, it is not unreasonable to expect a large number of contacted 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; starting delay is one such case.

The new *associated peers* model is ultimately easier to implement than the *de facto* 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. The model also grants extra opportunities for filtering . Thus, we consider this approach an improvement over the immediately obvious solution.

however we phrase this is determined by our lit review

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 port may also include the algorithm as part of the *associated peer* system to accentuate the impact it makes.

## 7.4 Stream Manager

The stream manager is responsible for converting partnerships to parent-child subscriptions, pushing data around 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 this buffer percentage will have been filled by the time playout reaches $\min\{I_{S_i}\}$. The ideal calculation for this remains an open research question and will likely require additional information to be gossiped between nodes.

When pushing buffer maps to partners when it has not yet received blocks in a given substream $i$, a new node should fill its latest block with some exceptional value, either a manually-filtered *null* or a very large negative number to ensure the node is never selected, 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.

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 are interested in that we are not receiving, implying issues with the 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.

## 7.5 Comparison of *coolstreaming-spiked* to our implementation

Our implementation discussed prior is *not* an implementation of *coolstreaming-spiked*, though they are 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 are nearly impossible to reason and are very likely the reason for the desyncing $M_c$ and partner counts we note in our results. The locking mechanism is proposed due to our experiences with this other approach. This will have an impact on results, though almost certainly a positive one.
- 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 both to the cut-down paper we worked from and that TCP support in *OverSim* is a somewhat hidden feature. This negatively impacts our results, as TCP networking is essential in the design of *Coolstreaming*.

- Our implementation initiates playout similarly to a buffering playout strategy, syncing to the starting block index and waiting for a threshold percentage of the buffer to be filled to play. No other buffering is performed during playout. This strange hybrid approach, whilst not invalid, is unlikely to be seen in a real research model and would 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. Whilst the final performance figures are unverified, we believe this to be a good future vein of research.

# 8 Conclusion

holy fuck. this sucks

# References


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.

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.

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://api.semanticscholar.org/CorpusID:14967108.

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://api.semanticscholar.org/CorpusID:13803281.

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 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.

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

———. 2008b. Understanding the start-up delay of mesh-pull peer-to-peer live streaming systems. https://api.semanticscholar.org/CorpusID:12141682.

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, 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://api.semanticscholar.org/CorpusID:16884016.

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://api.semanticscholar.org/CorpusID:1728223.

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.

Pouwelse, Johan, Paweł 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.

Sina, Majid, Mehdi Dehghan, Amir Masoud Rahmani, and Midia Reshadi. 2020. Wideplive: a coupled low-delay overlay construction mechanism and peer-chunk priority-based 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://api.semanticscholar.org/CorpusID:18754274.

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.

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.

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://api.semanticscholar.org/CorpusID:11798656.