A survey of peer-to-peer live video streaming schemes – An algorithmic perspective

Xiangyang Zhang, Hossam Hassanein

Abstract

Live video streaming applications have gained great popularity among users but exert great pressure on video servers and the Internet. Peer-to-Peer (P2P) networks provide an attractive solution due to their low cost and high scalability. A large number of P2P live video streaming schemes have been proposed and many deployments have appeared on the Internet. These schemes pursue vastly diverse directions, from mimicking IP multicast to BitTorrent-like swarming to distributed hash tables. In this paper, we provide a comprehensive and in-depth survey of P2P live video streaming schemes from an algorithmic perspective. Our purpose is to acquaint future designers with the critical design choices and their impacts on system performance. The primary objective of a P2P live video streaming system is to distribute packets from the video source to peers, and the collective paths through which a packet traverses form a tree. We focus on three aspects of how these trees are formed: determining the supplier–receiver relationships for each packet, handling the departure of the supplier or receiver before their relationship expires, and handling lost packets. We identify critical design choices in each aspect and propose a taxonomy according to these choices. Because the surveyed papers use different performance metrics and the reported results are heavily influenced by their experimental settings, we consider two measures to identify the impact of each design choice: we use a set of “internal” metrics in addition to the commonly used “external” metrics, and we examine performance metrics of schemes that have made the same design choice. For better understanding of how the design choices interact with one another and exposing future designers to the design choices specific to each individual scheme, we also provide systematic summaries for a large number of schemes.

1. Introduction

Video streaming applications are enormously popular on the Internet. The defining feature of video streaming is that a video is consumed while it is being downloaded. Video streaming can be classified into on-demand streaming and live streaming. In on-demand streaming applications, a user plays a video at his or her own pace and may seek new positions during the playback. A user can download the video at a rate higher than the video’s playback rate. In live streaming applications, all the users play a video in approximate synchronicity with the video source and download the video at its playback rate. The basic objective of live video streaming schemes is to distribute video packets from the video source to all the users before the playback deadline of each packet at each user. Some applications, such as video conferencing, require a playback delay within several hundred milliseconds. Other applications, such as Internet TV, are more tolerant, but...
the playback delay should not exceed a few minutes for an application to be called “live” streaming. In addition to distributing video packets with a delivery rate as high as possible and within the delay constraints, many schemes also strive to minimize the cost of the paths through which video packets traverse.

Live video streaming applications can be implemented using IP multicast, content delivery networks (CDNs), and peer-to-peer (P2P) networks. Among them, P2P networks have drawn great attention from both the research community and the industry due to their low cost and high scalability: only several video servers and bootstrap servers are required, and the upload bandwidth is contributed by users and grows as the number of users grows. Over the past two decades, a large number of P2P live video streaming schemes have been proposed. Many popular deployments have emerged on the Internet and some have grown into an enormous size. For example, PPSStream, a popular deployment in China, reported having tens of millions of users watching a popular show on the eve of the Chinese Spring Festival in 2010 [1].

In this paper, we provide a comprehensive and in-depth survey of P2P live video streaming schemes from an algorithmic perspective. The design of a P2P live video streaming scheme consists of a series of design choices. Categorizing the design choices made in each scheme and identifying their impacts on system performance will provide future designers with valuable knowledge and insights. Our approach is based on the following remarks. First, simply classifying P2P live video streaming schemes into several categories according to one or two design choices (e.g., tree-based and swarm-based) and arguing the pros and cons of each category does not expose future designers to the many design choices that have been proposed in the surveyed papers. However, categorizing all the design choices is not feasible since there are numerous design choices of different importance and some only apply to a small number of schemes. Second, system performance is often impacted by multiple design choices, and it may be hard to isolate the impact of a certain design choice. Third, the performance metrics reported in the surveyed papers do not provide an apple-to-apple comparison between schemes because they use different simulation or experimental settings and report different metrics. However, in order to make rational judgments and weigh trade-offs between different design choices, future designers need to understand the extent of impact of a design choice on system performance.

We address the above issues by first identifying a set of design choices that are critical to system performance. The primary objective of a P2P live video streaming system is to distribute video packets from the video source to all the peers within playback deadlines. In any P2P live video streaming system, if all the peer receive a packet and each peer receives the packet only once, the collective paths through which the packet traverses form a tree, which we call the propagation tree of the packet. We focus on three aspects of how propagation trees are formed. (1) Determining the supplier–receiver relationships between peers for each packet (typically determine the same relationship for a consecutive sequence of packets or for a period of time). (2) Handling the departure of the supplier or receiver before their relationship expires. (3) Handling lost packets. For each aspect, we identify a number of critical design choices and organize them in a hierarchical manner.

We take two measures to identify the impact of a design choice on system performance. First, since the surveyed papers use different performance metrics and may not report all the metrics we need, we identify a set of “internal” metrics, such as the propagation tree height and peer join latency in addition to the “external” metrics commonly used in the literature, such as the packet delivery rate, network traffic, and peers’ workload. These internal metrics are easier to relate to the design choices a scheme has made and are highly correlated with external metrics. Second, since the reported performance metrics in surveyed papers are heavily influenced by simulation settings and implementation details, we examine the design choices each scheme has made and examine the reported performance metrics of schemes that have made the same design choice. By examining a group of schemes, the impact of a design choice can be identified with higher confidence. The impacts of design choices are usually stated qualitatively (or as a range) because they are influenced by simulation settings. When applicable, we showcase the quantitative results (together with the simulation settings) reported in the surveyed paper as supplementing information.

The remainder of this paper is organized as follows. In Section 2, we first introduce the objectives and assumptions
used in the design of P2P live video streaming schemes and then introduce related research areas that are necessary to understand the discussions in following sections.

In Section 3, we first discuss the algorithmic choices to the design of P2P live video streaming schemes. We then present our taxonomy, which is based on the design choices of determining supplier–receiver relationships. We finally present the internal and external performance metrics, and provide a brief discussion of the impact of design choices on these metrics.

In Sections 4–7, we survey centralized and recursive schemes, structured tree-based schemes, swarm-based schemes, and unstructured tree-based schemes, respectively. In each section, we first provide a systematic summary to each scheme, focusing on their design objectives, decisions on the set of critical design choices, and the special design choices they employ for their individual objectives. We then make critical analysis and descriptive comparison between schemes according to their design choices.

In Section 8, we conclude this survey and discuss open issues.

2. Peer-to-peer live video streaming

A P2P system is a distributed system in which a transient population of peers self-organize into an overlay network on top of the substrate Internet to share computing power, storage, and communication bandwidth [2]. Each peer is both a client and a server. P2P networks have been used in many applications, such as file sharing, IP telephony, and video streaming. In this section, we first discuss objectives and assumptions used in the design of P2P live video streaming schemes. We then briefly introduce related research areas that are necessary to understand the discussion in the rest of this survey.

2.1. Preliminaries

A P2P live video streaming system consists of a number of peers connected via the Internet. Peers join a video channel according to a probability distribution. Each peer, except the video source, can elect to leave the channel with a probability. New video packets become available at the video source at a near-constant rate (i.e., peers cannot pre-fetch future video packets). Each peer can potentially reach any other peer, but due to its limited processing power, it can only maintain relationships with $M_i$ other peers. Each peer has an upload capacity and download capacity, which are the rate the peer can upload to or download from all other peers, respectively. The Internet path between a pair of peers is associated with a number of metrics; the most important of them are bandwidth, cost (used by IP routing protocols), delay, and packet error rate. The basic objective of a P2P live video streaming scheme is to distribute video packets from the video source to all the peers such that all the peers receive all the packets before their playback deadlines and each peer receives each packet only once. In the following, we first discuss the assumptions widely used in the analysis, simulation, and experiments in the literature and point out the differences between these assumptions and the real-world situations. We then discuss additional design objectives.

2.1.1. Peer churn

Peer arrivals and departures, collectively called peer churn, differentiates P2P systems from other types of distributed systems [2]. An empirical study on CoolStreaming [3] shows that peer arrivals exhibit a daily and weekly pattern, but neither short-term nor long-term arrivals can be described by a well-know distribution. Most peers stay in the system for a short period of time—more than half of the sessions are less than ten minutes, but peers tend to stay a long time after the initial “try-out” phase. These observations agree with those made on P2P video on-demand systems [4,5]. Most studies model peer arrivals as a Poisson process or a “flash crowd” (i.e., peers arrive in a short period of time), and use a fixed session length or an exponential distribution for session lengths.

2.1.2. Playback rate

Video frames are captured at a constant rate. The compressed video stream has a variable bit rate but the variance is typically insignificant, especially when the video source uses a traffic shaper to smooth out the packet rate. Most studies assume the video has a constant packet rate.

2.1.3. Reachability of peers

The use of firewall and network address translator (NAT) devices is common on the Internet. For example, 80% of peers are behind NATs in PPLive [4]. Firewalls and NATs may block UDP packets and TCP synchronize packets and thus cause some peers unreachable from outside. (These peers can reach other peers outside NATs or firewalls.) This problem is ignored in most studies because a peer can still find enough peers to establish relationships.

2.1.4. Upload capacity

It is helpful to divide the substrate Internet into two parts: access links from users’ premises to access points, and a core network connecting all the access points. An access point could be a digital subscriber line access multiplexer (DSLAM) for asymmetric digital subscriber line (ADSL) users, a cable modem termination system (CMTS) for cable modem users, or a router for local area network (LAN) users.

The access link’s outbound bandwidth and inbound bandwidth determine the maximum rates a peer can upload to or download from the P2P system. Peers with separate access links (e.g., ADSL users) have fixed bandwidth (disregarding the bandwidth consumed by other applications on the machine). Peers with shared access links (e.g., cable modem and LAN users) have dynamic available bandwidth. However, because two peers in a video channel rarely connect to the same access point, peers are usually assumed to have fixed bandwidth. Since the inbound bandwidth is equal to or greater than the outbound bandwidth for all the three types of access links, and a peer will not join a video channel if its download capacity is lower than the playback rate, some studies assume that peers have sufficient download capacity.
The abundance of upload capacity in a P2P system is measured by the resource index [6], defined as the ratio of the total upload capacity over the download requirements (i.e., the product of the number of peers and the video’s playback rate). Most upload capacity are contributed by a small fraction of peers. For example, 30% of peers contribute 80% of the upload bandwidth in CoolStreaming [3], and some peers upload at 16 folds of the playback rate in PPLive and Sopcast [7]. Most studies use a simple distribution for peers’ upload capacity.

### 2.1.5. Internet path between a pair of peers

The Internet path between a pair of peers consists of two access links and a core network path. Compared with the core network path, the access links typically have negligible costs and delays and smaller bandwidth. The delay and available bandwidth of the core network path depend on many factors of the Internet, such as the current route and traffic, and vary from time to time. The Meridian project [8] measured the delays between 2500 and 2500 hosts on the Internet in May 2004. The average delay was 75.6 ms with 90% between 10 ms and 200 ms. Errors may occur on the core network path or access links, and the error rate varies greatly from path to path and from time to time. Most studies use a fixed delay or a distribution for delay and assume that the core network has sufficient bandwidth. Some experimental studies measure delays and bandwidth on the fly. Lossy Internet links are considered in only a few studies.

The basic objective of a P2P live video streaming scheme is to distribute video packets from the source to all the peers within the playback deadlines. Many schemes have additional objectives. These objectives include:

- Propagation tree cost: The traffic that a P2P live video streaming scheme generated on the Internet is influenced by coding efficiency, communication overhead, and most importantly, and the paths through which video packets traverse. Building low-cost propagation trees is a major objective in many P2P live video streaming schemes.
- Playback delay: A scheme should guarantee the playback delay satisfying the intended application’s requirement.
- Number of users: A scheme may aim to support an arbitrary number of users or a limited number of users.
- Number of video sources: A scheme may aim to support multiple video sources or a single source in a channel. Some applications, such as Internet TV, have a single video source. In other applications, such as video conferencing, any participating party may be the video source.
- Peers’ workload: It is preferable that a scheme encourages peers to contribute upload bandwidth and places fair workload on peers.

### 2.2. Related research areas

The design of P2P live video streaming schemes may use techniques developed in other research areas. Covering such research areas is beyond the scope of this survey. In the following, we only provide a brief introduction necessary to understand the rest of this survey and give references.

#### 2.2.1. IP multicast and CDN

Live video streaming belongs to the one-to-many communication paradigm. To reduce network traffic and the video source’s workload, packets need to be duplicated by boxes in the network. Because users view the video in synchronicity, these boxes only need to cache a short period of the video. There are three choices for the duplication point: IP multicast duplicates packets using routers at the IP layer, CDNs duplicate packets using dedicated servers at the application layer, and P2P networks duplicate packets using peers, also at the application layer.

IP multicast makes efficient use of the Internet but has a number of problems that have hindered its deployment (refer to [9] for details). Today, it is the primary method for ISPs to provide IPTV services inside their own networks. A survey of IP multicast can be found in [10]. CDNs place a large number of cache servers strategically on the Internet to serve nearby users. CDNs are expensive to deploy and the number of servers needs to scale with the number of users. An introduction to CDN can be found in [11,12].

#### 2.2.2. Video coding, channel coding, and network coding

Videos can be encoded into a single layer or multiple layers. Single layer coding is the most efficient and may have features for reliable transmission. For example, H.264/MPEG-4 AVC supports slice data partitions. Different partitions, which are of different importance to decode the video, can be transmitted over multiple channels with unequal error-protection. With multiple layer coding, the fidelity of the decoded video degrades gracefully when fewer layers are available. Multiple layer coding includes layered coding (LC) and multiple description coding (MDC). LC, such as H.264/MPEG-4 SVC, encodes a video into a base layer and one or more enhancement layers. Higher layers are dependent on lower layers. MDC [13,14] encodes a video into multiple equal descriptions. A receiver can decode the video using any subset of the descriptions. LC can achieve coding efficiency comparable to single layer coding. For example, the increase of bit rate of SVC relative to AVC at the same fidelity can be as low as 10% [15]. MDC has large overhead [16] and is rarely used in the real-world. MDC is used in [17–21]; LC is used in [22,23].

Channel coding is an effective error control technique. The basic idea is that the sender encode \( k \) original symbols into \( n > k \) encoded symbols such that the original symbols can be reproduced even if some encoded symbols are lost in the transmission. Channel coding can be used between any pair of peers. Some P2P live video streaming schemes expand the use of channel coding to multiple points. For example, the design of [24] centers around a rateless channel coding scheme.

---

1. IP refers to TV services provided by telecom operators over IP networks with guaranteed quality. It differs from Internet TV, which refers to TV services delivered over the Internet.
Network coding [25] improves network throughput by encoding packets at intermediate nodes to make them more useful. It is used for pushing packets from multiple suppliers to a receiver in [26] and for error control in [27].

2.2.3. Distributed hash tables (DHTs)

DHTs were developed for P2P file sharing applications to locate the peers that store a data object given its object ID. In a structured file sharing system, each object and each peer has a unique ID. Each object is stored at a set of peers deterministically determined by its object ID. Peers self-organize in such a way that the look-up of a data object takes a short time. In P2P live video streaming applications, peer IDs and object IDs reside in the same space. Finding the route to the video source is the same as finding the route to the data object with the same ID, i.e., DHTs can be used as a unicast routing mechanism. DHT algorithms include [28–32]. In these algorithms, each peer has a routing table of size $O(\log N)$, and a query is guaranteed to reach the peer that stores the data object in $O(\log N)$ hops (i.e., the tree built using DHTs in P2P live video streaming applications has a height of $O(\log N)$). A comparison of DHT algorithms can be found in [33].

2.2.4. Network positioning

P2P live video streaming traffic can be greatly reduced if a peer can find peers that are nearby on the substrate Internet. The cost information between peers may be provided by ISPs, such as in the oracle [34] and P4P [35] proposals, or be inferred if ISPs’ help is unavailable. Peers can find nearby peers using CDN’s redirecting information [36] or the border gateway protocol (BGP) prefix [37], or use the propagation delays between peers to establish a virtual network positioning system. A centralized network positioning scheme where a peer measures the round trip time (RTT) to a set of landmark hosts to infer its network coordinate is proposed in [38], and a distributed scheme where a peer measures the RTTs to neighbors is proposed in [39].

The impact of using nearby peers as neighbors in P2P file sharing applications is studied in [35,36,40]. Both [36,35] report reduced autonomous system (AS) hops and file downloading time, but [40] claims that these benefits are difficult to achieve in real-world applications. The impact of using nearby peers as neighbors in P2P live video streaming applications is studied in [41]. Results show that although a low cost can be achieved, more packets are lost and the system is prone to partitioning because the neighboring graph no longer has the favorable properties of a random graph. Ref. [42] proposes that in addition to nearby peers, a peer should also select a small fraction of remote peers as neighbors to mitigate these side effects.

2.2.5. Incentive and fairness

Encouraging peers to contribute more upload bandwidth and discouraging “free-riders” improve system performance. Incentive mechanisms are widely used in P2P file sharing applications, where peers uploading more data are rewarded with a shorter downloading time. For example, BitTorrent employs a tit-for-tat policy where a peer rewards the peers from which it has the highest downloading rate [43]. In P2P live video streaming applications, because peers play in synchronicity and only cache a short period of the video, other incentives need to be used. Possible incentives include better video fidelity (i.e., more layers) [22], smoother playback (i.e., fewer lost packets), or a shorter playback delay (i.e., closer to the video source) [44]. A game theoretic incentive mechanism is proposed in [45] to deal with cheating and malicious users.

3. Algorithmic choices and taxonomy

A large number of P2P live video streaming schemes have been proposed in the past two decades, which can be classified from various perspectives. In this section, we first introduce existing classifications. We then discuss algorithmic choices to the design of P2P live video streaming schemes and present our taxonomy according to the algorithmic choices. We finally present the evaluation framework.

3.1. Classifications of P2P live video streaming schemes

According to whether there exist explicit trees, P2P live video streaming schemes can be classified into tree-based and swarm-based schemes. In tree-based schemes, peers form parent–child relationships. Upon receiving a packet, a peer immediately pushes the packet to its children. Therefore, tree-based schemes are sometimes called tree-push schemes or simply push schemes. Swarm-based schemes are sometimes called treeless schemes or mesh-based schemes. In swarm-based schemes, the video is sequentially split into chunks of fixed size (in bytes or in seconds). Peers form neighboring relationships and advertise their bit-maps, which describe the chunks they have, to neighbors and exchange missing chunks.3

According to the number of trees, tree-based P2P live video streaming schemes can be classified into single or multiple tree-based schemes. In single tree-based schemes, the video is encoded into a single layer and peers construct a single tree that all the packets follow. In multiple trees-based schemes, the video is either encoded into a single layer but interleaved into multiple substreams or encoded into multiple layers with MDC or LC. Peers construct a separate tree for each substream. Single tree-based schemes have lower overhead. Multiple trees-based schemes utilize peers’ upload capacity more efficiently—a peer can be a leaf node on one tree but upload to other peers on another tree. Multiple trees may be interior node-disjoint (i.e., a peer can be an interior node on at most one tree) or have common interior nodes.

According to whether peers first construct a mesh neighboring overlay, P2P live video streaming schemes can be classified into mesh-first or tree-first schemes. In

3 In this survey, we will not use the terms treeless, mesh-based, push, or tree-push to describe a scheme in order to avoid possible confusions. For example, tree-based schemes may first construct a mesh neighboring overlay, swarm-based schemes may use the push mode, and the term push has different meanings in swarm-push and tree-push. The terms swarm-based, swarm-pull, and swarm-push in this survey have the same meaning as mesh-based, mesh-pull, and mesh-push in [46], and data-driven has the same meaning as in [47].
mesh-first schemes, each peer maintains relations with a number of peers (called neighbors). These relationships form the edges of the neighboring overlay. Then peers build a spanning tree on the neighboring overlay. In tree-first schemes, there is no neighboring overlay and the tree is built in one step.

If an application allows multiple video sources, it needs to construct a separate source tree for each video source, or construct a shared tree. On a source tree, the video source is the tree root and tree edges are uni-directional (see Fig. 1a). There are two types of shared trees. The first type, as shown in Fig. 1b, makes all tree edges bi-directional; each node forwards packets to its parent and children except the one that the packets come from. With the second type, as shown in Fig. 1c, video packets are forwarded from the video source first to the tree root and then to all the nodes.

3.2. Algorithmic choices

In this section we discuss the algorithmic choices to the design of P2P live video streaming schemes. We assume that there exists a boot-up mechanism that peers can use to find the video source and focus on how the propagation tree for each packet is formed. To achieve the basic objective of distributing video packets from the video source to peers, a scheme needs to fulfill three tasks. (1) Determine the supplier–receiver relationships between peers for each packet such that the supplier–receiver relationships collectively form a tree that reaches all the peers (i.e., no loops, no partitions, and each node has an in-degree of one). (2) If a supplier–receiver relationship applies to more than one packet, deal with the situation when the supplier or receiver leaves before all the packets are sent. (3) Handle the situation when packets are lost due to Internet link errors or other reasons. These three tasks apply to both tree-based and swarm-based schemes, and examining schemes from the three aspects helps to understand the relation between tree-based and swarm-based schemes and how different choices can be combined in the design of new schemes. We only discuss the building of a single tree here unless the building of multiple trees leads to different conclusions. It is easy to expand a single tree-based scheme to a multiple tree-based scheme.

3.2.1. Determining supplier–receiver relationships

There are three basic algorithmic choices to determine the supplier–receiver relationships. The first choice is to use a central server to compute the supplier–receiver relationships using a centralized algorithm and inform each peer. The supplier–receiver relationships should apply to the whole stream (i.e., tree-based). It is possible that the server computes the supplier–receiver relationships for each chunk separately, but this brings no benefit and the server's workload and communication overhead would be too large. The centralized method can be explicitly or implicitly mesh-first. In the latter case, peers do not have neighbors but the central server implicitly maintains a mesh neighboring graph. For example, if the central server maintains each peer's virtual network position, it can implicitly construct a complete neighboring graph.

The second choice is to use a recursive algorithm. Each peer first requests the video source to be the supplier. The video source either accepts the request or delegates to another peer. This process repeats until peer \( i \) is accepted by some peer. The recursive method is tree-first. As long as a peer only delegates to its receivers, the recursive method guarantees building a tree that reaches all the peers. Same as the centralized method, the supplier–receiver relationships should apply to the whole stream. We remark that recursive algorithms are distributed but not fully distributed—peers close to the video source have high workload. We use the term recursive to differentiate them from fully distributed algorithms.

The third choice is to use a fully distributed algorithm. Most fully distributed schemes are mesh-first. (We will discuss tree-first distributed algorithms in Section 7.) Peers must maintain sufficient neighbors such that the neighboring graph remains connected in the presence of peer churn. In a dynamic environment, flooding each received video packet to all the neighbors can guarantee the packet reaching all the peers. However, flooding also results in large
amount of unnecessary traffic because multiple copies of the video packet are sent to peers. To reduce traffic, peers can advertise\(^5\) other information with a small size to neighbors such that peers can arrange the transmission of video packets using this information. There are two choices to do so: swarming and routing.

In the swarming method, each chunk is uniquely numbered, each peer advertises its bit-maps—each bit in the bit-map indicates whether the peer has a chunk—to neighbors, and peers establish the supplier–receiver relationship for each chunk. The establishment of supplier–receiver relationship may be initiated by the receiver or the supplier (called the pull and push mode of the swarm-based scheme in [48]). When initiated by the receiver, the receiver requests one and only one copy of each chunk from a selected neighbor, and hence it will not receive duplicated chunks. When initiated by the supplier, the supplier pushes chunks that neighbors do not have to neighbors. However, because a peer’s local copy of a neighbor’s bit-map may be stale, peers may receive duplicated chunks. Note that with the swarming method, peers establish supplier–receiver relationships for a chunk after knowing whether neighbors have the chunk; with other methods, a priori knowledge is not required. Because the establishment of supplier–receiver relationships are based on data availability, the swarming method is said to be “data-driven” [47,49].

In the routing method, each peer advertises routing information to neighbors periodically and establishes the supplier–receiver relationship for the whole stream. The establishment of supplier–receiver relationships is always initiated by the receiver. The routing information a peer advertises may be network-related (called network-driven routing), such as the peer’s geographic location (e.g., geographic routing), the peer’s ID (e.g., DHTs), the peer’s hop counts to the video source (e.g., distance-vector routing), and the costs between peers (e.g., link-state routing). There are well-known algorithms to build spanning trees on the neighboring graph with these information. The routing information a peer advertises may also be data-related (called data-driven routing), such as the peer’s bit-maps or the sequence number of the latest packet the peer has. Although there exist several data-driven routing schemes, it is not fully understood what algorithms can guarantee building a spanning tree (i.e., no partitions, no loops, and no duplicated packets) on the neighboring graph and what properties the tree will have.

Since peers have limited upload capacity, both the swarming method and the routing method (including centralized and full distributed) need to guarantee that a peer’s workload is within its upload capacity. For the routing method where the supplier–receiver relationships apply to the whole stream, this problem is typically addressed by one of the two methods. The first method is that a peer always accepts requests to be a parent. The second method is that a peer employs the connection admission control (CAC) mechanism. CAC is a widely used mechanism in communication systems where a server accepts a request only when it has the resources to serve the request. The first method takes a shorter time but the peer may become overloaded. The second method takes a longer time because a peer may be rejected multiple times before finding a supplier. For the swarming method where the supplier–receiver relationships apply to chunks, peers upload workload can be managed by limiting the output queue length of the supplier. Both the swarming method and routing method have a number of design choices specific to them; we will continue to discuss these choices in Sections 6.1 and 7.1 respectively.

A scheme may also aim to minimize the propagation tree cost. If the scheme is mesh-first, this objective is achieved by letting peers find nearby peers as neighbors and find nearby neighbors as suppliers. We have introduced the network positioning techniques to find nearby peers and the impact of using nearby peers as neighbors in Section 2.2. On the neighboring graph, two types of low-cost trees are often built: minimum weight spanning tree (MST), which can be built using the Prim’s algorithm or the Kruskal’s algorithm, and shortest path tree (SPT), which can be built using the Dijkstra algorithm or the Bellman-Ford algorithm. However, peers’ out-degrees are bounded due to their limited upload capacities. The degree-bounded MST and SPT problems are NP-hard [50]. In recursive schemes where there is no neighboring graph, the objective of minimizing the tree cost is typically addressed by optimize a cost function in each step of the recursive process. The cost function can be defined in two ways. The closest peer approach is similar to MST. A new peer \(i\) selects peer \(j\) that makes \(d_{ij}\) minimum, where \(d_{ij}\) is the cost of edge \(ij\). The \(en\), route approach is similar to SPT. Peer \(i\) selects peer \(j\) that makes \(d_{ij} + d_{ij}\) minimum, where \(s\) is the video source. Fig. 2b and c illustrate the trees built with the two approaches. When the propagation tree is optimum according to a cost function, the arrival of a new peer may cause existing peers to change their parents to make the new tree optimum. In the centralized method, the central server periodically computes the optimum tree and instructs peers to switch parents. In the fully distributed tree-based method, peers periodically checks whether to switch parents. In the recursive method, peers periodically re-join the tree.

In mesh-first schemes, peers usually first maintain a membership table and then select neighbors from the table. Unlike with neighbors, a peer does not periodically exchange information with peers in its membership table. A peer can obtain its membership table using a centralized method (i.e., a new peer asks a tracker for peers already in the system), or a recursively method (i.e., a peer uses a recursive process to find peers in the system), or a fully distributed method (i.e., a new peer knows at least one peer already in the system and uses it to find other peers in the system). The use of a membership table can reduce the time for a peer to find new neighbors and reduce the tracker’s workload. The neighboring graph can be structured (i.e., organized with DHTs) or unstructured. It can also “mirror” the substrate Internet such that the paths between peers will have lower costs.

---

5 Flooding refers to the practice where a peer forwards each received packet to all the neighbors except the neighbor from which the packet comes. Advertising refers to the practice where a peer forwards packets generated by itself to all its neighbors.
3.2.2. Handling the departure of the supplier or receiver before their relationship expires

When the receiver leaves, if the supplier–receiver relationship applies to a chunk, it does not matter whether the supplier continues to send the remaining packets or not. However, if the supplier–receiver relationship applies to the whole stream, the supplier should stop sending as soon as possible. This means that the supplier–receiver relationships must be renewed periodically. The statement that “the supplier–receiver relationship applies to the whole stream” should be more accurately expressed as “the supplier–receiver relationship applies to a segment of length $T_{rn}$”, where $T_{rn}$ is the interval between renewals. Typically $T_{rn}$ is one or two orders of magnitude larger than a chunk.

When the supplier leaves, if the supplier–receiver relationship applies to a chunk, the receiver simply request another peer to re-transmit the chunk. If the supplier–receiver relationship applies a video segment, a receiver needs to find a new supplier for the remaining packets in addition to recovering packets lost before the receiver finds the new supplier. (We call this process *tree-repairing*.) In the centralized method, the receiver requests the central server for a new supplier. In the recursive method, the receiver contacts the video source and repeats the recursive process. In the fully distributed routing method, the receiver relies on the routing algorithm. However, in a recursive or fully distributed scheme, repairing the tree in such a manner takes a long time. A fast tree-repairing (FTR) mechanism can greatly reduce the time that an orphaned peer takes to find a new parent. When its parent leaves, a peer immediately attaches to a temporary parent. There are many choices for candidate temporary parents, such as an ancestor or a pre-determined back-up parent. If peers employ CAC, the orphaned peer may need to try a list of candidate parents before it is accepted. Because FTR may cause forwarding loops, and if a scheme also aims to minimize the propagation tree cost, cause the tree no longer optimized, the peer needs to look for a new parent using the recursive process or using the distributed routing.
3.2.3. Handling lost packets

There are two choices for error control: coding and re-transmission. The coding method can be used for applications that only tolerate a delay of several hundred milliseconds, but has a larger overhead and performs poorly when errors occur in consecutive bursts. The re-transmission method has little overhead but introduces several seconds of delay. There are three choices for re-transmission. First, a peer can request its supplier to send a missing packet if it is in a supplier–receiver relationship. (We call this method point-to-point re-transmission.) With this method, if a packet is lost at a peer, the packet will be lost at all the peer’s descendants. Second, a peer can request the video source to send a missing packet. (We call this method source re-transmission.) This method requires that the video source has sufficient upload capacity. Third, a peer can request a neighbor to send a missing packet. (We call this method multi-point re-transmission.) This method is most effective, but is feasible only if a peer has neighbors (i.e., explicitly mesh-first) and knows which neighbors have the missing packets. The latter requirement can be met if peers advertise bit-maps.

Fig. 3 summarizes the algorithmic choices discussed in this section. The supplier–receiver relationships may be determined centrally, recursively, or using a fully distributed algorithm. In the centralized method, there is an implicit or explicit neighboring graph, the neighboring graph can be structured or unstructured, the supplier–receiver relationships apply to video segments, and the establishment of supplier–receiver relationships is network-driven. In the recursive method, there is no neighboring graph, the supplier–receiver relationships apply to video segments, and the relationship establishment is network-driven. In the fully distributed method, the neighboring graph can be structured or unstructured, the supplier–receiver relationships can apply to a chunk where the relationship establishment is data-driven, or apply to video segments where the relationship establishment can be network-driven or data-driven. If a relationship applies to a chunk, a receiver can rely on the error control mechanism to recover the chunk when the supplier leaves before all the packets in the chunk have been sent. If a relationship applies to a video segment, a receiver needs to find a new supplier to repair the tree, and it has many choices for the FTR mechanism. Errors can be corrected by coding, and if a playback delay of several seconds or more is allowed, by re-transmission. A peer can request the video source or its supplier to send missing packets. If a peer has neighbors and knows their bit-maps, it can request a neighbor to send missing packets.

3.3. Taxonomy from an algorithmic perspective

We propose a taxonomy of P2P live video streaming schemes according to how the supplier–receiver relationships are determined. Compared with existing classifications, this taxonomy exposes more design details and hence is not as simple, but it helps to identify the impacts of design choices on system performance and to understand how different choices can be combined in the design of P2P live video streaming schemes. Table 1 lists the surveyed schemes and their categories. Table 1 also includes a list of attributes of these schemes. The size of small, medium, and large refers to systems with tens, hundreds, and thousands or more peers, respectively. About half of these schemes have additional objectives, and most of them aim to reduce the tree cost.

Centralized schemes include ALMI [51,17]. Both schemes aim at minimizing the tree cost. The neighboring graph is explicit in ALI and implicit in CoopNet. In ALMI, the central server instructs each peer to measure the RTTs to \( n \) other peers and maintains a regular neighboring graph. In CoopNet, each peer measures the RTTs to a set of landmark hosts to compute its virtual network coordinate and reports to the central server. Both schemes support applications with a sub-second playback delay, and hence cannot use the re-transmission error control technique.
Recursive schemes include Overcast [52], NICE [53], ZIGZAG [54], THAG [18], NHAG [19], TURINstream [21], HMTP [55], Yoid [56], and Island Multicast [57]. According to whether peers are first grouped into clusters, recursive schemes can be further classified into three sub-categories: without clusters, with clusters, with IP multicast “islands” (i.e., networks that support IP multicast). Overcast has no clusters. NICE, ZIGZAG, THAG, NHAG, and TURINstream first group peers into clusters and then build trees with clusters as nodes. HMTP, Yoid, and Island Multicast build trees with IP multicast islands as nodes and use scoped IP multicast inside each island.

Fully distributed schemes can be further classified according to three independent criteria: whether the neighboring graph is structured or unstructured, whether the supplier–receiver relationships apply to chunks or the whole stream (to be more accurate, video segments), and whether the establishment of supplier–receiver relationships is data-driven or network-driven. This will result in eight combinations. Some combinations are irrational or have no corresponding schemes in the literature. For example, swarm-based schemes are always data-driven, and no swarm-based schemes we surveyed use a structured neighboring graph. Structured tree-based schemes are always network-driven. After removing those combinations, we have four sub-categories: swarm-based schemes, structured tree-based schemes, unstructured tree-based network-driven schemes, and unstructured tree-based data-driven schemes.

Swarm-based schemes includes Chainsaw [62], the original CoolStreaming [49], Prime [20], Pulse [44], LayerP2P [22], and R² [26]. We call a scheme swarm-based if peers advertise bit-maps, peers use a priori knowledge of neighbors’ bit-maps to establish supplier–receiver relationships, and the relationships apply to chunks. All the swarm-based schemes use an unstructured neighboring graph. Swarm-based schemes can be further classified into swarm-pull and swarm-push schemes according whether the establishment of supplier–receiver relationships are initiated by receivers or suppliers. In the swarm-push schemes, peers may receive duplicated chunks because their local copies of neighbors’ bit-maps may be stale. R² [26] uses random network coding to address the problem. It is possible to address this problem by using a structured neighboring graph, but no studies that we are aware-of follow this direction. Note that the swarm technique itself is a multi-point re-transmission error control technique.

Structured tree-based schemes include SplitStream [60], Ratnassamy et al. [58], Bayeux [59], and Borg [61]. We call a scheme structured if it organizes peers with DHTs. The neighboring graph is defined by peers’ routing tables. If peer i has an entry to peer j in the routing table, there will be an edge in the neighboring graph. Structured tree-based schemes use DHTs as a unicast routing

### Table 1

Taxonomy of P2P live video streaming schemes.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Tree alg</th>
<th>Tree type</th>
<th>Opt. cost</th>
<th>Attributes of intended applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALMI [51]</td>
<td>C</td>
<td>Shared-Single</td>
<td>Yes</td>
<td>Sub-sec, Small, Multiple</td>
</tr>
<tr>
<td>CoopNet [17]</td>
<td>C</td>
<td>Src-Multi</td>
<td>Yes</td>
<td>Sub-sec, Medium, Single</td>
</tr>
<tr>
<td>Overcast [52]</td>
<td>R-NC</td>
<td>Src-Single</td>
<td>No</td>
<td>Seconds, Small, Single</td>
</tr>
<tr>
<td>NICE [53]</td>
<td>R-WC</td>
<td>Shared-Single</td>
<td>Yes</td>
<td>Sub-sec, Small, Multiple</td>
</tr>
<tr>
<td>ZIGZAG [54]</td>
<td>R-WC</td>
<td>Src-Single</td>
<td>Yes</td>
<td>Sub-sec, Medium, Single</td>
</tr>
<tr>
<td>THAG [18]</td>
<td>R-WC</td>
<td>Src-Multi-Disjoint</td>
<td>Yes</td>
<td>Sub-sec, Medium, Single</td>
</tr>
<tr>
<td>HMTP [55]</td>
<td>R-IPI</td>
<td>Shared-Single</td>
<td>Yes</td>
<td>Sub-sec, Small, Multiple</td>
</tr>
<tr>
<td>Yoid [56]</td>
<td>R-IPI</td>
<td>Shared-Single</td>
<td>Yes</td>
<td>Sub-sec, Small, Multiple</td>
</tr>
<tr>
<td>Island multicast [57]</td>
<td>N/A</td>
<td>Src-Single</td>
<td>Yes</td>
<td>Sub-sec, Small, Single</td>
</tr>
<tr>
<td>CAN-based [58]</td>
<td>D-DHT</td>
<td>Src-Single</td>
<td>No</td>
<td>Sub-sec, Large, Single</td>
</tr>
<tr>
<td>Bayeux [59]</td>
<td>D-DHT</td>
<td>Src-Single</td>
<td>No</td>
<td>Sub-sec, Large, Single</td>
</tr>
<tr>
<td>SplitStream [60]</td>
<td>D-DHT</td>
<td>Src-Multi-Disjoint</td>
<td>No</td>
<td>Sub-sec, Large, Single</td>
</tr>
<tr>
<td>Borg [61]</td>
<td>D-DHT</td>
<td>Src-Single</td>
<td>No</td>
<td>Sub-sec, Large, Single</td>
</tr>
<tr>
<td>Chainsaw [62]</td>
<td>D-U-S-Pull</td>
<td>Src-Chunk</td>
<td>No</td>
<td>Minute, Large, Single</td>
</tr>
<tr>
<td>CoolStreaming [49]</td>
<td>D-U-S-Pull</td>
<td>Src-Chunk</td>
<td>No</td>
<td>Minute, Large, Single</td>
</tr>
<tr>
<td>Prime [20]</td>
<td>D-U-S-Pull</td>
<td>Src-Chunk</td>
<td>No</td>
<td>Minute, Large, Single</td>
</tr>
<tr>
<td>Pulse [44]</td>
<td>D-U-S-Pull</td>
<td>Src-Chunk</td>
<td>No</td>
<td>Minute, Large, Single</td>
</tr>
<tr>
<td>LayerP2P [22]</td>
<td>D-U-S-Pull</td>
<td>Src-Chunk</td>
<td>No</td>
<td>Minute, Large, Single</td>
</tr>
<tr>
<td>R² [26]</td>
<td>D-U-S-Push</td>
<td>Src-Chunk</td>
<td>No</td>
<td>Seconds, Large, Single</td>
</tr>
<tr>
<td>SPANC [27]</td>
<td>D-U-T-DD</td>
<td>Src-Multi</td>
<td>No</td>
<td>Seconds, Large, Single</td>
</tr>
<tr>
<td>GridMedia [64]</td>
<td>D-U-T-DD</td>
<td>Src-Multi</td>
<td>No</td>
<td>Seconds, Large, Single</td>
</tr>
<tr>
<td>Narada [65]</td>
<td>D-U-T-ND</td>
<td>Src-Single</td>
<td>Yes</td>
<td>Sub-sec, Small, Multiple</td>
</tr>
<tr>
<td>Gossamer [66]</td>
<td>D-U-T-ND</td>
<td>Src-Single</td>
<td>Yes</td>
<td>Sub-sec, Small, Multiple</td>
</tr>
<tr>
<td>TreeClimber [67]</td>
<td>D-U-T-ND</td>
<td>Src-Single</td>
<td>Yes</td>
<td>Seconds, Large, Single</td>
</tr>
<tr>
<td>OMNI [68]</td>
<td>D-U-T-ND</td>
<td>Src-Single</td>
<td>Yes</td>
<td>Seconds, Large, Single</td>
</tr>
<tr>
<td>ChunkySpread [70]</td>
<td>D-U-T-ND</td>
<td>Src-Single</td>
<td>Yes</td>
<td>Seconds, Large, Single</td>
</tr>
<tr>
<td>Treebone [71]</td>
<td>D-U-T-ND</td>
<td>Src-Single</td>
<td>Yes</td>
<td>Seconds, Large, Single</td>
</tr>
</tbody>
</table>
mechanism, which can route packets between any pair of peers. As a result, structured tree-based schemes can build multicast trees,7 and support multiple video sources with little extra overhead by building a source tree for each video source.

Unstructured tree-based network-driven schemes includes Narada [65], Gossamer [66], ChunkySpread [70], Treebone [71], Fastmesh [69], and TreeClimber [67]. For convenience, we also include OMNI [68], which is tree-first, in this category. Unstructured tree-based data-driven schemes include GridMedia [64], the new CoolStreaming [63], Substream Trading [23], and SPANC [27]. We say a tree-based scheme is data-driven if the propagation tree is determined by the propagation of video packets. Peers may use neighbors’ bit-maps, latest packet’s sequence numbers, or history of exchanging video packets to select parents. Otherwise we say a tree-based scheme network-driven. In data-driven tree-based schemes, the propagation tree may change shapes even when there is no peer churn and all the network conditions remain the same. It is hard to predict the properties of the propagation trees or even guarantee that all the parent–child relationships form a tree. Network-driven tree-based schemes usually, but not necessarily, aim to minimize the propagation tree cost. Minimizing tree cost may cause extra overhead and have negative impacts on other objectives. For example, peers may need to switch parents more frequently, which will result in more lost packets. We remark that the boundary between data-driven and network-driven is not clear-cut; peers may use both data-related and network-related information to determine parent–child relationships. For example, in Fastmesh [69], a peer selects parents by their “power”, which is a function of neighbors’ residual upload bandwidth and the delay to the neighbor. All the unstructured tree-based schemes construct the neighboring graph using a central tracker or a distributed membership management protocol, and many schemes construct a random neighboring graph.

We remark that advertising bit-maps does not necessarily make a scheme swarm-based or data-driven. In swarm-based schemes, bit-maps are used to establish supplier–receiver relationships for chunks that neighbors already have. In data-driven tree-based schemes, bit-maps are used to establish supplier–receiver relationships for future video segments. Bit-maps can be used in the re-transmission error control technique in any mesh-first schemes, including centralized schemes, distributed network-driven tree-based schemes, and even IP multicast schemes [72].

3.4. Evaluation framework

In the literature, the performance of a P2P live video streaming system is evaluated mainly from four aspects: the smoothness of the playback, the timeliness of the playback, the network traffic generated by the system, and the workload of peers. However, the surveyed papers use different performance metrics, and the reported results are heavily influenced by their simulation or experimental settings. Therefore, we identify a set of “internal” metrics, which are easier to relate to the algorithmic choices a scheme has made and are highly correlated with the external metrics. We evaluate schemes using both the external and internal metrics. In the following, we first introduce the external metrics and then introduce the internal metrics and how they are related to external metrics. We also provide a briefly discussion of the relationship between design choices and performance metrics.

Playback smoothness is best measured by the packet (or chunk) delivery rate, defined as the fraction of video packets that arrive before their respective playback deadlines at a peer (after error-correction). The relationship between packet loss and video quality, called loss distortion model, is influenced by many factors but the delivery rate is the single most important one [73]. Some schemes measure playback smoothness by the fraction of time that a peer has a parent or by peers’ throughput (i.e., peers’ downloading rate). However, packets may get lost even when a peer has a parent, and throughput is meaningful only in situations where peers can play the video with part of the video packets.

Playback timeliness is measured by the playback delay, which is the sum of the delivery delay (the time a packet takes to travel from the video source to a peer) and the buffering delay (the time a packet stays in the peer’s buffer before being played back). The buffering delay should be large enough to absorb the variance of the delivery delays of different packets.

With the same coding overhead and communication overhead, the Internet traffic generated by a P2P live video streaming system is determined by the propagation tree cost and is usually benchmarked against the IP multicast tree. When IP path cost information is unavailable, all the surveyed papers use the delay of a path (obtained by measuring the RTT) as its cost. We adopt this practice and use the term cost, delay, or distance interchangeably. Except the tree cost, stretch and stress [65] are often used in the literature. The stretch of a peer refers to the ratio of the cost of the peer’s root-path on the application layer propagation tree over the cost of the peer’s root-path on the IP multicast tree. A node’s root-path on a tree refers to the path from the tree root to the node along tree edges. The stress of a substrate Internet link refers to the number of copies of a video packet traversing the link.

A peer’s workload includes upload workload and processing workload. A peer’s upload workload is measured by its uploading rate. A peer’s processing workload is hard to quantify; it is often qualitatively described by the number of requests the peer handles or the number of relationships it maintains. The communication overhead is measured by the ratio of control traffic volume over video traffic volume.

Internal metrics include propagation tree properties, the time an orphaned peer needs to find a new parent, the efficacy of error control mechanism, peer join complexity, and maintenance overhead. Propagation trees are

---

7 A multicast tree can reach a subset of the nodes on a graph while a spanning tree reaches all the nodes. DHT-based schemes allow the neighboring graph to have peers not in the video channel and build multicast trees. Most other tree-based schemes construct the neighboring graph only with peers in the video channel and construct spanning trees on the neighboring graph.
described by their cost, node degrees, heights, and root-path length (in terms of cost and hops). As mentioned above, the tree cost measures the network traffic. A peer’s out-degree on the propagation tree is linear to its upload workload. A peer’s root-path cost, when benchmarked against IP multicast trees, equals its stress. The tree height is related to more than one external metric. A higher tree (and a larger root-path length in terms of hops) may cause more lost packets, especially in tree-based schemes that have no error control mechanism, and lead to a longer playback delay in swarm-based schemes. It also leads to larger stretch and smaller stress. The packet delivery rate of a scheme is mainly determined by the tree-repairing time and the success rate to recover lost packets. Peer join complexity refers to the number of rounds and messages it takes when a new peer (or an orphaned peer) joins. We define a single communication between two peers to be one message and the time it takes to be one round, which is usually one round. Since the departure of a peer usually requires the same amount of rounds and folds of messages, where is the number of the peer’s children, we only discuss peer join complexity in Sections 4–7. The maintenance overhead refers to the amount of periodic operations and the size of soft states a peer needs to maintain. Peer join complexity and maintenance overhead reflect peers’ processing workload and the system’s communication overhead.

We now briefly discuss the relationship between design choices and performance metrics. The packet delivery rate of a scheme is mainly determined by its choice of the error control mechanism. The packet delivery rate is also impacted by the packet buffering delay, the tree repairing time, the propagation tree height, etc. There are four reasons for packet loss. First, the Internet only provides the best-effort packet delivery service and each Internet path has an error rate. Second, a peer’s parent may leave, which will cause a packet loss rate of \( \frac{r}{c} \), where is the tree-repairing time and is peers’ average lifespan. Third, a peer may switch parents even if its parent does not leave (e.g., to optimize a cost function), and because peers are in approximate synchronicity, packets may be lost. For example, if a peer switches from parent , which is about to send packet , to a new parent , because peer may have already proceeded to packet , packets to will be lost. If the sequence numbers of the packets peers are about to send are uniformly distributed in \((a, b)\), on average, packets will be lost for each switching. Fourth, when the upload bandwidth is scarce in the system and not efficiently utilized, a peer may not be able find a supplier all the time. Packet loss is usually the first three reasons in tree-based schemes, and by the first and fourth reasons in swarm-based schemes. It should be noted that swarm-based schemes have a high packet delivery rate. This is because the swarming technique is also an effective multi-point re-transmission error correction mechanism. A tree-based scheme that uses a multi-point re-transmission (or source re-transmission) error control mechanism can achieve the same packet delivery rate at a smaller playback delay.

The playback delay of a scheme mainly depends on two algorithmic choices: whether the scheme is tree-based or swarm-based, and whether the re-transmission error control technique is used or not. A tree-based scheme without re-transmission error control can achieve a playback delay of several hundred milliseconds (even with 1000 or more peers in the system). A tree-based scheme with re-transmission error control typically have a playback delay between 5 and 20 s, which allows a lost packet to be re-transmitted once or twice. The playback delay of a swarm-pull scheme is typically more than a minute and is proportional to the propagation tree height. The playback delays of peers in a swarm-push scheme are smaller than in swarm-pull schemes, but are still significantly longer than in tree-based schemes.

The network traffic or propagation tree cost of a scheme depends on whether receivers can find the closest peers among the whole population as suppliers. Centralized schemes and recursive schemes can achieve a lower tree cost than fully distributed schemes. In mesh-first schemes, a peer first selects neighbors and then selects suppliers from neighbors. The first selection is more important because neighbors are selected from the whole population while parents are selected from neighbors. (The side-effects of selecting nearby peers as neighbors is discussed in Section 2.2.) When both selections are random with respect to the path cost between peers, each hop on propagation trees will have an average cost equal to the path cost between two arbitrary hosts on the Internet. This will cause enormous Internet traffic, and this is the case in most real-world P2P live video streaming systems.

The propagation tree height and node degrees depend on the algorithmic choices for supplier–receiver relationships, the upload capacity of peers and their positions on the Internet. Multiple tree-based schemes and swarm-based schemes have shorter trees than single tree-based schemes, and interior node-disjoint multiple tree-based schemes have shorter trees than their non-disjoint counterparts. In multiple tree-based schemes, because peers have stable upload workload, peers’ upload capacity is more efficiently utilized than in swarm-based schemes. This means that if the upload bandwidth is scarce in the system (e.g., the resource index is close to one), multiple tree-based schemes have higher packet delivery rate than swarm-based schemes. Tree-based schemes typically use CAC to guarantee that peers are not overloaded (i.e., to guarantee peers have bounded out-degrees); swarm-based schemes use chunk scheduling algorithms.

Peers processing workload depends on many factors. Peers have balanced workload in centralized and fully distributed schemes and unbalanced workload in recursive schemes. The communication overhead also depends on many factors.

We will discuss the impacts of design choices on system performance in more details in Sections 4–7. Since playback delays always follow the above analysis, we will not repeat the comparison of each scheme’s playback delay in Sections 4–7. For a fair comparison the schemes, we ignore the difference resulting from the number of trees. We remark that using multiple trees is superior to using a single tree, and almost all the single tree-based schemes can be extended to a multiple tree-based scheme with minor changes. We also remark that our discussions focus
on the objectives of distributing packets and reducing tree cost for video live streaming applications. Some schemes have additional objectives and target a wider range of applications. For better understanding of why each scheme has made its design choices as well as how different design choices interact with one another, we provide a systematic summary for each surveyed scheme.

4. Centralized and recursive schemes

In this section, we first survey centralized and recursive schemes and then provide comparative analysis.

4.1. Centralized schemes

Centralized schemes include ALMI [51] and CoopNet [17]. A peer queries the central server for parents upon its arrival at the system or departure of its parents. The main concern of centralized algorithms is scalability. In the study of CoopNet [17], the trace captured on September 11, 2001 at MSNBC is simulated. The study shows that an ordinary server can handle 18,000 hosts and 1000 churns per second.

ALMI [51] aim to support small group multicast applications with multiple sources. A multicast group (called ALMI session) consists of a controller and a number of members. Peers know of the controller using an out-of-band mechanism, such as a URL. When a peer wants to join a session, it sends a JOIN message to the controller. The controller assigns a member ID to peer x and gives peer x the member ID and IP address of its parent. Then peer x sends a GRAFT message to its parent to attach to the tree. A peer monitors the status of its parent. If its parent leaves, the peer asks the controller for a new parent. The controller instructs each member to monitor the distance (the paper uses RTT) to n other members and uses the results to maintain a regular neighboring overlay. The controller employs a heuristic degree bounded minimum spanning tree (DBMST) algorithm to build a bi-directional shared tree: it periodically calculates the cost reduction of a new tree, and if the reduction exceeds a threshold, it instructs peers to switch parents. Each member’s degree on the tree is bounded by its access link’s bandwidth.

CoopNet [17] targets the flash crowds at web servers with streaming content, live or on-demand. These videos are usually short and hence peers are very dynamic. A new peer browses the web server to select a video, and indicates its interest in joining the multicast group by attaching to the current node. Bandwidth is measured by downloading a 10 kB file. A peer periodically measures bandwidth to its siblings, parent, and grandparent. The peer relocates under a sibling if the relocation does not decrease its bandwidth to the root, or relocates under its grandparent if the relocation increases the bandwidth. Therefore, the tree may be unnecessarily high since each peer attempts to put itself as far away from the root as possible. When its parent fails, the peer first relocates under its grandparent then optimizes. Although Overcast has no explicit CAC, because the downloading rate from a peer decreases when the peer is overloaded, the scheme effectively controls the out-degrees of peers.

4.2. Recursive schemes without clusters

In a typical recursive scheme, a new (or orphaned) peer n, starting from a well-known RP, contacts a peer p for a list of its children. Peer n compares peers p and its children using a utility function. If p makes the function minimum, peer n selects p as parent. If a child c makes the function minimum, peer n contacts c and repeat the above process. There are many choices for the utility function f. For example, $f_n(x)$ can be defined as the cost or bandwidth between peer n and peer x.

Overcast [52] targets small group multicast applications that have stable peers and can tolerate a 10–15 s delay. The scheme aims to maximize the tree’s throughput. A new peer contacts a well-known registry to find the tree root of the group it intends to join, sets the root as the current node, and uses a recursive process to join the tree. In each round, the peer compares the bandwidths to the current node directly and via its children (i.e., uses the en route approach). If the two bandwidths are close (within 10%), it sets the child as the current node and repeats the process. If no such child exists, it terminates the process and attaches to the current node. Bandwidth is measured by downloading a 10 kB file. A peer periodically measures bandwidth to its siblings, parent, and grandparent. The peer relocates under a sibling if the relocation does not decrease its bandwidth to the root, or relocates under its grandparent if the relocation increases the bandwidth. Therefore, the tree may be unnecessarily high since each peer attempts to put itself as far away from the root as possible. When its parent fails, the peer first relocates under its grandparent then optimizes. Although Overcast has no explicit CAC, because the downloading rate from a peer decreases when the peer is overloaded, the scheme effectively controls the out-degrees of peers.

4.3. Recursive schemes with clusters

Recursive schemes may use clusters for different purposes, such as improving scalability, building multiple interior node-disjoint trees, or achieving a low tree cost. Consequently, they organize peers into clusters in different ways. All these schemes have a control plane and a data plane. On the control plane, peers organize into clusters, select cluster leaders, and split and merge clusters to keep the cluster size within a range. Clusters in [18,19,21] are flat, meaning that each peer belongs to one and only one cluster. Clusters in [53,54] are hierarchical, meaning that a peer may belong to multiple clusters. On the data plane, one or more trees are built to distribute video packets.

NICE [53] targets low data rate streaming applications with a small population and multiple sources. The control plane is hierarchical. As shown in Fig. 4a, nearby peers form clusters of size $[c, 3c]$ at each layer, where c is a system parameter. Each cluster has a leader, which is at the center of the cluster (i.e., it has min max distance to other
Leaders of layer $i$ clusters form layer $i+1$. The apex of the hierarchy is the well-known RP. This control plane is constructed using a recursive process, starting from RP. At each step, a peer $x$ queries the current peer in layer $i$ for its cluster members. Peer $x$ measures its distance (i.e., RTT) to each member and sets the closest member as the current peer, which is a cluster leader in layer $i-1$. The process repeats until peer $x$ reaches layer 0. Inside a cluster, each peer periodically measures and advertises its distance to other peers. When a peer joins or leaves a cluster at layer $i$, a new leader is selected, which re-joins the hierarchy using the same recursive process except terminating the process at layer $i+1$. Given a source, the data plane is obtained from the control plane in the same manner as a bi-directional shared tree. When a peer joins or leaves a cluster at layer $i$, a new leader is selected, which re-joins the hierarchy using the same recursive process except terminating the process at layer $i+1$. Given a source, the data plane is obtained from the control plane in the same manner as a bi-directional shared tree.8 In Fig. 4a, the arrows show the data topologies when peer 5 is the source. In NICE, The control hierarchy mitigates the impact of peer arrival order. However, high-layer peers have heavy control and data-forwarding workload. Peers in the highest layer are leaders in $\log_2 N$ clusters and have an out-degree of $\log_2 N$ on the data plane. As a result, the substrate links close to high-layer peers have high stress.

ZIGZAG [54] addresses the bottleneck at high-layer peers in NICE. A cluster leader (except the source) does not forward to its subordinates. Instead, a non-leader, when at its highest layer, forwards to subordinates of its cluster mates. For example, in Fig. 4b, peer 2 cannot forward to its subordinates, peers 1 and 3, but can forward to peers 4, 6, 7, and 9, the subordinates of its cluster mates 5 and 8. Peer 5 can forward to peers 2 and 8 because it is the source. Thus the data plane is a source tree, and a peer’s maximum out-degree is $9c^2$.

The scheme only allows the tree root to be the source. A new peer $x$ uses a recursive process, starting from the source $s$, to join a cluster in layer 0. In each step, peer $x$ calculates $d_{xy} + d_{ys}$, where $d_{xy}$ is the delay between peers $x$ and $y$, for each child $y$ of the current peer, and sets the child that makes the value minimum and has a path on the data plane to a cluster in layer 0 as the current peer. For example, in Fig. 4b, peer $x$ contacts peers 5 and 2 to join cluster 3 in layer 0. If a peer arrival causes over-sized clusters, these clusters split and new cluster leaders are selected. Each peer periodically advertises its out-degree on the data plane to its cluster mates; the peer with the lowest out-degree is chosen as the leader. When the parent $z$ of a cluster leaves, the cluster leader attempts to find the cluster mate of $z$ that has the minimum out-degree to be the new parent. In ZIGZAG, a leader is not at the center of its cluster, and peer arrival order influences the tree cost. Peers at the highest layer have a smaller out-degree than in NICE on the data plane, but are leaders in $\log_2 N$ clusters and have the same number of cluster mates on the control plane.

---

8 Because a peer can send to its cluster members, the data plane is slightly different from a shared tree.
THAG [18] and NHAG [19] encode the video into S descriptions using MDC and build interior node-disjoint trees. Peers are partitioned into arrangement graphs (AGs), whose sizes are fixed in THAG and variable in NHAG. We first introduce THAG then point out its differences from NHAG. An AG $A_{m,k}$ is a $(m - 2)$-regular graph with $\binom{m}{k}$ vertices. Each vertex is denoted by its AG coordinate $(x_1, x_2, \ldots, x_k)$, where $1 \leq x_i \leq m$ and $x_i \neq x_j$ for $i \neq j$. An edge exists between two vertices if and only if their coordinates differ in exactly one dimension. At most $\frac{m}{2}$ interior node-disjoint trees can be built on $A_{m,2}$. The roots have the largest degree $2(m - 2)$. Exactly two nodes are leaf nodes on all the trees. To construct $S$ interior-node-disjoint trees, AGs must have a size $m > S + 2$, and each root in a child AG connects to a source in the parent AG. Fig. 5 shows two trees using $A_{4,2}$.

On the control plane, AGs are organized into a tree. Each AG has a leader, which is one of the full-leaf nodes. Starting from RP, in each step of the recursive process, a new peer $x$ contacts the AG leader. If the AG is not full, peer $x$ is accepted. If the AG is full, the AG leader either replaces an existing peer, according to the distance to the AG sources, with peer $x$, or redirects peer $x$ to the child AG to which peer $x$ is closest. If the current AG has no child AG, the leader will create one and let $x$ be its leader. Because the leader uses proximity as the criteria, peers that are close to the video source on the substrate network have fewer root-path hops on the overlay. On the data plane, $m - 2$ interior node-disjoint trees are built: $S$ trees deliver data, and others are for fast tree-repair. When an interior node leaves, a non-delivery tree provides alternative paths temporarily. AG leaders maintains the structure of AGs. All the members periodically report to the leader. If an interior node on tree $T$ leaves, its parent on tree $T$ will act as a virtual node for the vacant position and undertake the responsibility. If an AG leader leaves, an assigned parent will undertake the responsibility. If an AG source leaves, one node of the child AG is promoted to fill the position.

THAG assumes that peers have homogeneous upload bandwidth. To accommodate peers’ heterogeneous upload bandwidth, NHAG uses AGs of variable size and AG leader considers peers’ upload bandwidth in the recursive process. Assume the AG’s size is $m$ and the new peer $x$ can serve $m_x$ peers. If $m > m_x$, the leader creates a new child-AG for peer $x$. If the AG cannot create child AG, the leader redirects peer $x$ to the child AG whose size is closest to $m_x$. If $m < m_x$, the joining process is similar to THAG, but
the request size is used as a replace criteria and peers with small requested size are replaced. When an AG source leaves, the peer with large requested size is promoted.

**TURINstream** [21], unlike other recursive schemes, does not optimize network cost. The video is encoded into S descriptions and each description is split into I stripes. The data rate of a stripe is called a “slot”. As shown in Fig. 6, each description may have a separate “root source”, each stripe has a separate tree, and each cluster has a “level”, which is the maximum root-path length, in terms of the number of clusters, of the cluster on all the SL stripe stress. Each cluster C has a “source”, which is a member of the clusters of lower levels, for each stripe. The source forwards video packets to a peer x, which is selected in a round-robin fashion, in cluster C, and peer x forwards the packets to cluster mates. On the control plane, peers are partitioned into clusters of size less than c and clusters form a tree. Each cluster C has a cluster-head, which is also a source. Starting with RP, a new peer x follows the control topology until it is accepted by a cluster. In each step, if the cluster is not “full”, the cluster-head accepts peer x; otherwise, the cluster-head may replace the “worst” peer in the cluster according to a function, or refers peer x to a cluster-head of the next level. A cluster is said to be full if it has no upload bandwidth to serve a new peer that wishes to join the cluster. Each member periodically reports its available slot to the cluster-head. The cluster-head manages the upload bandwidth and appoints sources to construct stripe trees.

**4.4. Recursive schemes with IP multicast islands**

The Internet is a network of networks. IP multicast is supported in some networks, but unless two ISPs have an agreement, IP multicast is not supported across their network boundary. A number of schemes [55–57,66,68] propose to connect these isolated IP multicast networks using proxies. The proxies may be dedicated servers or normal peers. These proxies form trees at the application layer, and IP multicast is used inside each island. Unlike clusters in the previous section, each IP multicast island can have an arbitrary number of peers.

**HMTP** [55] and **Void** [56] are similar, so we only introduce HMTP here. A multicast group is identified by an IP multicast address inside islands and by a group identifier (GID) across islands. Hosts can query a well-known registry for the GID of an IP address. Each IP multicast island has a dedicated member (DM). A single host is also considered as an island and the host is the DM. DMs encapsulate IP multicast packets and send to DMs in other islands using IP unicast. A DM joins a multicast group if any peer in the island wishes to join.

All the DMs in a multicast group build a bi-directional shared tree. A DM knows of the tree root using an out-of-band mechanism. Starting from the root, in each step of the recursive process, the DM measures RTTs to the current node and its children. If a child has the shortest RTT, the DM sets the child as the current node and repeats the process; otherwise the current node is selected as the potential parent. The potential parent may reject the DM to be a child (e.g., it has no spare upload bandwidth). If rejected, the DM resumes the process starting from the second last current node. A DM periodically re-joins the shared tree to find a better parent. To reduce the root’s workload, the re-joining process starts from a node on the DM’s root-path. If its parent leaves, the DM requests the peers on the root-path, in reverse order, to be its parent. An ancestor will reject the request if it has no spare bandwidth (so an orphaned DM may take several rounds to re-join the tree).

**Island multicast** [57] distributes the control and forwarding workload to several peers. The scheme uses an existing tree-building algorithm at the control plane. Each island has a leader, and all the leaders form a tree. Two islands are neighbors if their leaders are connected. Unlike HMTP, leaders do not forward data packets. Each island has one ingress and multiple egress bridge-nodes. An egress node of one island forwards data packets to ingress nodes of neighboring islands. Leaders of neighboring islands exchange the list of bridge-nodes in their islands. The paper proposes several methods to select bridge-nodes, such as by proximity to the leader or to bridge-nodes in neighboring islands.

**4.5. Comparative analysis**

In the following, we compare centralized and recursive schemes and summarize the results in Tables 2 and 3.

**4.5.1. FTR, error control, and delivery rate**

Centralized schemes take two RTTs to repair the tree. Most recursive schemes have an FTR mechanism. In HMTP,
an orphaned peer tries its ancestors on its root-path in turn until it is accepted. In Overcast, ZIGZAG, THAG, and NHAG, an orphaned peer attaches to grandparent (Overcast) or ex-parent’s siblings (ZIGZAG) or other peers (THAG and NHAG), which will always accept the orphaned peer. This method takes a shorter time but peers may get overloaded. No scheme has a re-transmission mechanism except that Overcast uses TCP. (Overcast allows tens of seconds of playback delay). As a result, these schemes have a low packet delivery rate. Most schemes report that peers receive less than 90% of packets. NICE reports best results: with 64 peers and 32 random churns in 900 s, 30% of peers receive all the packets when Internet links are error-free. However, the experiment settings is not fully described in the paper. Also note that in NICE, peers have sufficient upload capacity.

4.5.2. Tree cost and network traffic

All the schemes except TURINstream and Overcast aim to minimize the tree cost and report good results. There are two direct comparisons conducted using simulation. Ref. [54] compares ZIGZAG with NICE and shows that ZIGZAG has lower stretch (because ZIGZAG uses the en route approach while NICE uses the closest peer approach) and similar stress. Ref. [53] compares NICE with Narada and shows that NICE has lower stress (by about 25%). It is worth noting that some Internet links in NICE have very large stress because peers at the top of the hierarchy have an out-degree of \(cN\).

In recursive schemes, such as HMTP, Island Multicast, NICE, ZIGZAG, THAG, and NHAG, a peer can find the closest peers from the whole population and by directly measuring RTTs. In centralized schemes that use a virtual network positioning system, such as CoopNet, a peer can find the closest peers from the whole population but the distance is computed and subject to modeling errors of the virtual network positioning system. In centralized schemes that construct a mesh neighboring graph, such as ALI, because the neighbors of a peer \(i\) are randomly selected, they are unlikely to be the closest peers of peer \(i\).

4.5.3. Tree height, node degree, and peers’ upload workload

A small tree height can be achieved if a scheme places peers with higher upload capacity closer to the tree root and “fills” their out-degree bounds. THAG, NHAG, and TURINstream have a small tree height because they try to fill peers’ out-degree bounds. NICE and ZIGZAG have a fixed and small tree height of \(\log N\) because of their hierarchical structure. All other schemes have a tree height determined by peers’ positions on the substrate network. In Overcast, because each peer tries to position itself as far as possible from the video source provided that the bandwidth to the source does not decrease, the tree tends to be high.

It is easier to enforce out-degree bounds in centralized schemes and recursive schemes without clusters. However, HMTP assumes that proxies, like routers, have enough bandwidth and choose not to enforce out-degree bounds. In recursive schemes with flat clusters (i.e., THAG, NHAG, and TURINstream), peers’ bandwidth is used for both intra- and inter-cluster connections. In TURINstream, the workload is distributed into multiple peers in a flexible manner. In THAG and NHAG, peers’ out-degrees are determined by the AG size. THAG uses a fixed AG size. Although NHAG uses variable AG size, peers must have a minimum upload bandwidth to be accepted into an AG. In recursive schemes with hierarchical clusters (NICE and ZIGZAG), peers may be overloaded because the hierarchy hinders enforcement of out-degree bounds. In NICE, peers at the top of the hierarchy have an out-degree of \(c\log N\), where \(c\) is the cluster size. In ZIGZAG, in the worst case, a peer may be the parent of all the peers at the next lower level (i.e., has an out-degree of \(9\log N\)).

4.5.4. Peer join complexity

In centralized schemes, a new peer or an orphaned peer takes only two RTTs and two messages to attach to the tree. In recursive schemes without clusters, peers take a significantly longer time (\(\log N\) RTTs) and more messages (\(\log N\) messages) to attach to the tree. Using flat clusters reduces the time only by one RTT, but can reduce the number of messages by a factor of \(c\) if the cluster leader can compare the distance between the new peer and cluster members and redirect the new peer to a proper member. This is the case in THAG, NHAG, and TURINstream. Using hierarchical clusters reduces neither the time nor the number of messages. In NICE, peers actually take more time and messages to attach to the tree. This is because NICE maintains the cluster leaders to be at the center of their clusters. After a new peer arriving at a cluster, the cluster needs to re-select the leader, and the leader changes, the new leader will in turn join the tree. Because recursive schemes have
high join complexity, they are not necessarily more scalable than centralized schemes. All the surveyed recursive schemes aim at small to medium group size.

4.5.5. Maintenance overhead

At a minimum, a peer needs to monitor packet arrivals from its parent and whether its children are alive. This overhead applies to all the schemes and is not counted in the comparison tables. In centralized schemes, peers have little overhead. The major problem of recursive schemes is that peers close to the root have significantly more overhead because they are contacted by most joining peers. Recursive schemes with hierarchical clusters, such as NICE and ZIGZAG, incur even more overhead on peers at the top of the hierarchy, because these peers need to monitor delays or degrees of their $O(\log N)$ cluster mates. In [54], the authors report that NICE has higher overhead to repair tree than ZIGZAG because the departure of peers at the top of the hierarchy cause many peers to change parents in NICE. In all recursive schemes, peers need to periodically re-join the tree to restore the desired tree shape damaged by peer churn. They also need to monitor ancestors or siblings if the scheme has an FTR mechanism. Recursive schemes with clusters need to merge and split clusters to maintain a proper size.

To summarize, centralized schemes can construct short, balanced, and low-cost trees if the central server has global knowledge of peers’ virtual network positions and upload bandwidth, have low peer-join complexity, and can quickly repair broken trees. Peers have low maintenance overhead, and an ordinary server can handle the workload in a system with thousands of peers. Recursive schemes can build low-cost trees without any peer having global knowledge and enforce peers’ out-degree bound by CAC. However, peers close to the root tree have large processing workload. Grouping peers into clusters can improve scalability and achieve other benefits, but often imposes new constraints. The use of hierarchical clusters will severely increase the processing and uploading workload of peers at the top of the hierarchy. Peer churn incurs a long convergence time, and thus recursive schemes are not necessarily more scalable than centralized schemes. Most centralized and recursive schemes aim at applications with a sub-second playback delay and have no re-transmission error control mechanism, and hence have a high packet loss rate.

5. Structured tree-based schemes

Structured tree-based schemes organize peers with DHTs. DHTs provide a scalable unicast routing mechanism. Each peer has a routing table of size $O(\log N)$, and a peer can reach any other peer within $O(\log N)$ hops. In this section, we first survey structured tree-based schemes and then provide comparative analysis.

5.1. DHT-based schemes

Structured tree-based schemes include Ratnasamy et al. [58] (uses CAN [29]), Bayeux [59] (uses Tapestry [31]), SplitStream [60] and Borg [61] (both use Pastry [30]). They all target at applications with a sub-second playback delay.

Ratnasamy et al. [58] explore the architecture of CAN [29] to flood multicast packets to group members. CAN centers around a $d$-dimensional torus. Fig. 7 shows a 2-dimensional CAN where all the peers are members of a multicast group. The source forwards packets to all its neighbors, other nodes only forward packets to certain directions inferred from their positions in the torus. For example, node $a$ forwards to left, right, and up, and node $b$ forwards only to left. In a CAN where only a subset of the peers are members of a multicast group, group members first form a “mini-CAN” and then packets are flooded throughout the mini-CAN. A multicast group has a group ID that is in the same space as object IDs. The peer who “owns” the group ID serves as the bootstrap node. A peer that wishes to join the multicast group joins the mini-CAN in the same way as a new peer joins a CAN (refer to [29] for details).

Bayeux [59] uses Tapestry [31]. A new peer $x$ sends a JOIN message, which is routed to the root by Tapestry. The root replies with a TREE message, which is routed back to peer $x$. The two paths are usually different due to the asymmetry of Tapestry. Upon receiving the TREE message, intermediate nodes set up soft states so as to forward future packets from the root to peer $x$. When peer $x$ wishes to leave, it sends a LEAVE message to the root. The root replies with a PRUNE message, which is routed to peer $x$ along the same path as the TREE message. Upon receiving the PRUNE message, intermediate nodes reset their soft states.

Bayeux uses First Reachable Link Selection (FRLS) to deal with peer and overlay link failures. Each peer $x$ collects periodic reports from its next hop peers to predict their data delivery reliability. Upon receiving a packet, peer $x$ selects the next hop peer $y$ with a certain reliability level, and forwards the packet to peer $y$ together with a list of nodes that peer $y$ should forward the packet to. Bayeux proposes to name nearby nodes with close IDs because Tapestry approaches the destination digit by digit using
the destination’s ID; however, this task is not trivial and is not described in the paper.

SplitStream [60] uses Scribe [74] to build multiple interior node-disjoint source trees. The video is encoded into 5 substreams with MDC. Each substream has a unique stripe ID, in the same space as node and object IDs. The peer whose ID is closest to a stripe ID is the tree root for the substream. Scribe is a tree-building algorithm based on Pastry [30]. In Scribe, a node y that wishes to join a multicast group sends a request, which is routed towards the root by Pastry until it reaches an existing node on the tree, and intermediate nodes set up soft states to forward future packets using the reverse path. SplitStream chooses stripe IDs in such a way that the most significant bits are 0, 1, . . . , (S − 1). Therefore, each substream has a separate tree rooted at a separate source, and because Pastry routes messages digit by digit according to the destination’s ID, a node is an interior node on only one tree with high probability. Fig. 8 illustrates two interior node-disjoint multicast trees.

Each peer decides the number of substreams to receive depending on its download bandwidth (i.e., MDC is used for receiver-side rate control). Peers exercise CAC when being requested to be a parent. A new peer x can preempt an existing child y if peer x’s node ID has longer prefix match than that of y. If orphaned, peer y first requests former siblings to be its parent, and if being rejected, it resorts to the spare capacity tree for a new parent. All the peers that have spare upload bandwidth form a spare capacity tree.

Borg [61] explores the asymmetry of Pastry to reduce delays and link stress. Given two peers x and y, the overlay paths xy and yx, called forward path and reverse path, respectively, are usually different and have different delays. Each path has a probability of 50% to have a shorter delay. The links near the tree root will have higher link stress when more forward paths are used to build a multicast tree. Borg constructs the upper part and lower part of the multicast tree differently. The shorter of the forward and reverse paths is used to construct the upper part of the tree to reduce delays with the other path used to construct the lower part of the tree to reduce link stress. The boundary separating the upper part and lower part is a system parameter. The authors claim that setting it to half of the average path length gives a good trade-off between delays and link stress.

5.2. Comparative analysis

In the following, we compare structured distributed schemes and summarize the results in Tables 4 and 5.

5.2.1. FTR, error control, and delivery rate

Only Bayeux has an FTR mechanism. All the four schemes support sub-second delay applications, and hence have no re-transmission mechanism. The delivery rate is not reported in all the four schemes, but SplitStream reports the convergence time. When 25% of peers fail, it takes more than 20 s for the substream trees to converge. This is significantly longer than other categories of schemes, meaning DHT-based schemes have a lower packet delivery rate than other schemes.

5.2.2. Tree cost and network traffic

In CAN, a peer’s ID is independent from the portion of ID space it owns, and a new peer can choose nearby peers to be its neighbors in the multi-dimensional torus. In Tapestry and Pastry, a peer’s ID determines the portion of ID space it owns and its neighbors (i.e., entries in its routing table). In P2P systems, IDs are randomly assigned to peers from a large ID space. Therefore, it is easy for a CAN-based scheme to optimize a cost function, but it is hard for Bayeux, Borg, and SplitStream to achieve a low cost if they were to enforce peers’ out-degree bounds. Bayeux and SplitStream report having a low cost. However, this is because in their experiment setting, peers have a large out-degree and all the peers are within a few hops from the video source.

5.2.3. Tree height, node degree, and peers’ upload workload

DHT-based schemes have a tree height of \( \log_2 N \), which is determined by the DHT algorithms. Node degrees are also impacted by the DHT algorithms and peers typically have a degree of \( B \log_2 N \) in the worst case, where \( B \) is a system parameter.\(^9\) Enforcing degree bounds is cumbersome and hence peers may be overloaded. SplitStream uses a spare capacity tree while other schemes simply assume the peers have sufficient upload capacity.

5.2.4. Peer join complexity

DHT-based schemes have the longest convergence time of all categories of schemes, which will significant impact on the packet delivery rate. The complexity is determined by the underlying DHTs rather than the addition of a multicast function. Peers first take \( \log_2 N \) rounds to update their routing tables and then another \( \log_2 N \) rounds to join the tree.

5.2.5. Maintenance overhead

DHT-based schemes have large overhead because of maintaining the DHTs. Each peer maintains a routing table

\(^9\) When system parameter \( d \) in CAN is set to \( \frac{1}{2} \log_2 N \), The scheme in [58] has the same out-degree, tree height, and peer join complexity as Bayeux.
of size $O(\log n)$. Bayeux, SplitStream, and Borg have extra overhead. Borg splits the tree into the upper part and the lower part, Bayeux peers maintains redundant links to next hops peers, and SplitStream peers maintains a spare capacity tree.

To summarize, DHT-based schemes are not a good fit for P2P live video streaming applications. DHT’s main strength, to route between any pair of peers with a bounded hop count, does not bring significant benefits to P2P live video streaming applications. DHT’s weakness, such as large overhead, slow response to peer churn, and tight control of the neighboring overlay topology, deteriorates the performance of critical metrics to P2P live video streaming applications.

### 6. Swarm-based schemes

In this section, we first discuss the algorithmic choices specific to swarm-based schemes, then survey swarm-pull and swarm-push schemes, and finally give comparative analysis.

#### 6.1. Algorithmic choices

In swarm-based schemes, a peer, by contacting a well-known tracker (in BitTorrent parlance) or a peer already in the video channel, acquires the IP addresses of a list of peers in the system and establishes neighboring relationships. The video is split into chunks of size $T_{c, bh}$. Each chunk has a unique sequence number. Each peer maintains a sliding window of recently received chunks, which is described by a bit-map. A peer advertises its bit-map to neighbors immediately after receiving a new chunk or every interval of length $T_{ad}$ (called gossip interval). In swarm-pull schemes, upon receiving a bit-map message or every interval of length $T_{c, bh}$, a peer uses a scheduling algorithm to decide which chunks should be requested (i.e., chunk-selection) and from which neighbors these chunks should be requested (i.e., target-selection). In swarm-push schemes, upon receiving a chunk, a peer pushes the chunk to the neighbors that do not have the chunk (according to the peer’s local copies of neighbors’ bit-maps).

In swarm-pull schemes, peers will not receive duplicated chunks, and if the playback delay is large enough, a chunk will eventually reach all the peers in time. However, with a limited playback delay, a chunk may never reach a peer since the peer will not request the chunk after its playback deadline. There are numerous choices for chunk-selection and target-selection in the scheduling algorithm. An algorithm may select chunks randomly, or prefer chunks that are rare among neighbors (rarest-first), or chunks that have the latest sequence numbers (latest-first), or chunks that are approaching their playback deadlines (most-urgent-first). Target-selection is more versatile. A peer may select from neighbors randomly, or select neighbors according to their workload or pair-wise bandwidth and delays. A peer may also implement an incentive policy in target-selection. Ref. [75] provides a summary of the chunk selection and peer selection algorithms of swarm-based schemes.

We remark that although swarm-pull schemes are inspired by BitTorrent, the rarest-first and tit-for-tat policies, to which BitTorrent owes its success, are not as effective. The swarming window is the whole file in BitTorrent but is only a small portion of the video in P2P live video streaming applications, and two peers are less likely to have chunks that the other party wants at the same time.

In swarm-push schemes, peers will receive duplicated chunks if their local copies of neighbors’ bit-maps are stale, but notifying neighbors immediately after receiving a chunk will incur high overhead. $R^2$ [26] addresses this dilemma using network coding. Ref. [76] uses “pull tokens”: a supplier requests pull tokens from a receiver before pushing chunk to the receiver. (This will increase the playback delay.) There are numerous choices for chunk-selection and target-selection in swarm-push schemes as well. Eight supplier-initiated algorithms are analyzed in [48]. The authors conclude that the “random peer, latest chunk” algorithm (i.e., the supplier first chooses a neighbor randomly and then chooses the latest chunk the neighbor does.

---

### Table 4
Comparison of structured (DHT-based) schemes.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>DHT</th>
<th>Tree cost</th>
<th>Peers’ out-degrees</th>
<th>Tree height</th>
<th>Peer join complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratnasamy et al.</td>
<td>CAN</td>
<td>Low</td>
<td>Max $2d$</td>
<td>$\frac{1}{d}\sqrt{N}$</td>
<td>$O\left(\frac{4d}{\sqrt{N}}\right)$</td>
</tr>
<tr>
<td>Bayeux</td>
<td>Tapestry</td>
<td>High</td>
<td>Max $8\log n$</td>
<td>$\log_2 N$</td>
<td>$O\left(\log n\right)$</td>
</tr>
<tr>
<td>SplitStream</td>
<td>Pastry</td>
<td>High</td>
<td>Bounded by bandwidth</td>
<td>$\log_2 N$</td>
<td>$O\left(\log n\right)$</td>
</tr>
<tr>
<td>Borg</td>
<td>Pastry</td>
<td>High</td>
<td>Max $8\log n$</td>
<td>$\log_2 n$</td>
<td>$O\left(\log n\right)$</td>
</tr>
</tbody>
</table>

### Table 5
Comparison of structured (DHT-based) schemes, Continued.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Fast tree repair</th>
<th>Error control</th>
<th>Maintenance overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratnasamy et al.</td>
<td>No. Restructure overlay</td>
<td>No</td>
<td>$2d$ Unicast entries</td>
</tr>
<tr>
<td>Bayeux</td>
<td>Yes. Use FRLS</td>
<td>No</td>
<td>$O\left(8\log n\right)$ routing entries, backup links</td>
</tr>
<tr>
<td>SplitStream</td>
<td>No. Restructure overlay</td>
<td>MDC</td>
<td>$O\left(8\log n\right)$ routing entries, spare capacity tree</td>
</tr>
<tr>
<td>Borg</td>
<td>No. Restructure overlay</td>
<td>No</td>
<td>$O\left(8\log n\right)$ routing entries</td>
</tr>
</tbody>
</table>
not have) achieves the optimal packet delivery rate and delay.

All the swarm-based schemes construct a random neighboring graph for its favorable properties, such as high connectivity and short diameter. Some schemes simply rely on the tracker to provide a list of random peers in the system. Others use a distributed membership management protocol, such as the scalable probabilistic membership protocol (SCAMP) [77], to guarantee “true” randomness. However, no evidence in the literature shows that the extra complexity of using such protocols can be justified by the benefits.

6.2. Swarm-pull schemes

Swarm-pull schemes include Chainsaw [62], the original CoolStreaming [49], Prime [20], Pulse [44] and LayerP2P [22]. Large-scale P2P live video streaming deployments on the Internet also use swarm-pull schemes.

Chainsaw [62] is a proof-of-concept implementation of the swarm-pull scheme. Each peer connects to a set of randomly selected neighbors to form a neighboring overlay. Peers employ the “random chunk, random peer” scheduling algorithm and notify their neighbors immediately upon receiving a new chunk. (Each peer has 30–40 neighbors, and hence each chunk results in 30–40 messages.) Each peer maintains a window-of-interest, which is the set of chunks that the peer is interested to acquire, and a windows-of-availability, which is the set of chunks that the peer already has. A peer randomly selects a chunk in its window-of-interest, and requests it from a randomly selected neighbor. A peer limits the number of outstanding requests to a neighbor to balance the workload of neighbors. A problem of random chunk-selection is that some chunks are never requested from the source and hence are lost at all the peers. To fix this problem, the source records the chunks it has uploaded. When a peer requests a chunk that has been uploaded, the source will send the oldest chunk that has never been uploaded rather than the requested chunk.

CoolStreaming is one of the few schemes that have been deployed on the Internet. The original version [49] uses a swarm-pull scheme; the new version [63] changes to a tree-based scheme. The original CoolStreaming constructs a random neighboring overlay using the SCAMP [77]. A peer randomly selects a fixed number of peers in its membership table to neighbor with. The authors try different peer degrees on the neighboring overlay and conclude that 4 is reasonably good. The packet loss rate decreases marginally for higher degrees. A peer updates its neighbor list periodically or when triggered by the departure of an existing neighbor. In a periodic update, the peer replaces the neighbor that has the least traffic to and from itself.

The video is split into chunks of one second. Each peer has a sliding window of 120 chunks. Every one second, a peer advertises its bit-map to its neighbors and requests missing chunks from its neighbors. The scheduling algorithm first considers chunks available at one neighbor, two neighbors, etc. in order. If a chunk is available at more than one neighbor, the neighbor with the highest bandwidth and from which the estimated chunk arrival time is before the deadline selected. The authors evaluate the scheme with 500 Kb/s playback rate and about 200 hosts on PlanetLab [78]. A peer buffers 10 s video before playing back. For the nodes that have an ON/OFF period of more than 400 s, 95–97% of the chunks arrive in time; for stable nodes, 97–98% of chunks arrive in time. When the playback rate is 200 Kb/s, the packet delivery rate is more than 99.5%. The overhead for exchanging bit-maps is about 2%.

Prime [20] introduces the concept of bandwidth bottleneck and content bottleneck. The scheme constructs a directed random neighboring overlay. A peer contacts the RP to find a random subset of peers of size \( \frac{\text{bw}_{\text{wpf}}}{C2} \) in the system to be parents and accepts up to \( \frac{\text{bw}_{\text{wpf}}}{C2} \) peers to be children, where \( \text{bw}_{\text{wpf}} \) is a system parameter, and \( \text{obw} \) and \( \text{ibw} \) are the peer’s outbound and inbound bandwidth respectively. Each edge is able to transmit a data unit every interval of length \( t \); \( \frac{\text{bw}_{\text{wpf}}}{C2} \times t \) is called a data unit. A peer’s inbound link from peer \( y \) may not be filled due to either a bandwidth or content bottleneck. The bandwidth bottleneck occurs when peer \( y \) has content to send but has no available bandwidth to peer \( x \). The content bottleneck occurs when peer \( y \) has available bandwidth to peer \( x \) but has no content to send. Prime has no bandwidth bottleneck because the manner the neighboring overlay is constructed, and attempt to minimize the content bottleneck by quickly diffusing new packets. However, this method has a downside: when peers have fewer neighbors, content bottlenecks are more likely to occur and hence peers’ upload bandwidth is under-utilized. Among all the swarm-based schemes as well as unstructured tree-based schemes, only Prime uses this method.

The sliding window is divided, in order, into a diffusion window of size \( \text{depth} \times t \), a swarming window of size \( K_{\text{min}} \times t \), and a play window of size \( t \). The scheduling algorithm is essentially receiver-initiated “latest useful chunk, random peer”. Every interval of length \( t \), a peer requests all missing packets in the diffusion and play window and a subset of packets in the swarming window (using random or rarest-first algorithm). If more than one parent has a packet, the peer selects one randomly or selects the parent that has the lowest workload during the last interval. All peers play the video simultaneously \( \text{depth} + \frac{K_{\text{min}}}{C2} \times t \) seconds behind the source. The purpose of the diffusion window is to diffuse new packets quickly. If new packets are selected with probability 1 by peers, it takes \( \text{depth} = \log N \) intervals for a new packet to reach the bottom of chunk trees. The purpose of the swarming window is for peers to exchange packets after new packets reach the bottom of chunk trees. The system parameter \( K_{\text{min}} \) reflects the extra swarming time after packets reach the bottom of the tree; it depends on the number of diffusion trees and the inbound degrees of peers.

Pulse [44] modifies the tit-for-tat and optimistic unchoosing mechanism of BitTorrent to reward peers that upload more chunks with a shorter playback delay by moving them closer to the source. Because these peers have larger upload bandwidth, the average delay of peers in the system is also reduced. Peers uses an epidemic protocol such as SCAMP to maintain a list of neighbors. Each peer uses
two incentive policies: a primary optimistic tit-for-tat policy to maintain a MISSING list of fixed size (four in the paper, same as BitTorrent), and an excess-based altruistic policy to maintain a FORWARD list whose size is determined by the peer’s upload bandwidth (in excess to that required to serve the four missing neighbors).

Every interval of length $t$, a peer $x$ adds the top three neighbors from which it receives the most number of non-duplicate chunks during the last interval to the MISSING list; it also optimistically adds the peer whose sliding window range overlaps least with its own to the MISSING list. Peer $x$ selects FORWARD neighbors by their history scores. Neighbor $y$’s history score is initialized to a fixed value when peer $x$ first knows of $y$, increases if peer $x$ receives a useful chunk from $y$ when $y$ is neither in the MISSING nor the FORWARD list, and decreases each time $y$ is selected to the FORWARD list. Peer $x$ only selects neighbors whose sliding window range does not overlap with its own (i.e., FORWARD peers are “farther” away from the source than peer $x$). Peer $x$ uses an algorithm similar to that in CoolStreaming [49] to select which chunks to pull. If more than one neighbor has the chunk, peer $x$ prefers the MISSING neighbor. Peer $x$ grants the pull requests for chunks, regardless from MISSING or FORWARD neighbors, in order of the number of times it has sent these chunks in the past, and prefers the least sent chunks and breaks ties by selecting randomly.

LayerP2P [22] encodes the video with LC and rewards peers that upload more with more layers (and hence higher fidelity). Each peer maintains a sliding window for each layer, periodically exchange bit-maps with neighbors, and requests chunks from neighbors. Pull requests for base layer chunks are regular requests; others are probing requests. Each peer also monitors the rate it receives from each neighbor.

Each peer obtains a list of peers from the RP and selects some to neighbor with. Each peer maintains $[n_i, n_u]$ neighbors, and periodically replaces the neighbor from which it has the lowest receiving rate with a new peer. Depending on which party initiates the neighboring relationship, a peer classifies a neighbor as an initiator or receptor. When an initiator $x$ and a receptor $y$ first establish neighboring relationship, $x$ treats $y$ as if it has received from $y$ with a high rate, but $y$ treats $x$ as if it has received from $x$ with a low rate. This measure prevents a free-ride from obtaining a high download rate by adding neighbors. Every interval of length $t$, a peer requests missing chunks from randomly-chosen neighbors. Pull requests have an expiry time of one interval; a peer must request again in the next interval if it has not received a requested chunk. A peer allocates its upload bandwidth to neighbors in proportion to the rate with which it downloads from neighbors. A peer expects regular requests to be served on time with high probability, and probing requests to be served when the supplier has allocated enough upload bandwidth to it. Therefore, peers that upload more chunk can receive more layers, and hence have high fidelity.

Popular commercial deployments include PPStream, PPLive, Sopcast, and TVAnt. They usually offer videos with a 300–500 Kb/s playback rate. They use proprietary software and their designs are unavailable in the literature. Most measurement studies [7, 79–86] target PPStream and PPLive. In the following, we introduce the two deployments according to these studies.

In PPStream, according to [80, 84], the video was split to windows of 1024 KB (about 20 s), and each window was in turn split to chunks of 8 KB. Each window was uniquely identified by a timestamp denoting the start time of the window. A peer swarmed in one window. It advertised its bit-map, which described the chunks in the window, to neighbors, and requested missing chunks from neighbors. The chunk scheduling algorithm is unknown, but [84] claims the behavior was different from the rarest-first policy. Peers typically had 5 to 15 neighbors [80], and a peer did not prefer nearby peers when selecting neighbors [83]. The average playback delay correlated with the number of peers. The playback delay was 20 s on average but reached 140 s at some peers [80]. Video packets were carried on TCP [83, 85].

According to [79], PPLive was a typical swarm-pull scheme. Each video might have a different chunk size. Each peer strived to buffer 200 s of the video. Campus users and residential users had about 40 and 20 neighbors, respectively, for popular channels, and 5–10 for unpopular channels. The start-up delay between a user tunes in a channel to the video being played was 10–20 s for popular channels. The difference of playback delays at uses was up to 140 s. According to [80], the neighboring graph was random when the channel size was small, but peers began to cluster with the increase of peers. However, [7, 86] did not observe traffic locality. Peers had no tit-for-tat policy reciprocal mechanism [7].

6.3. Swarm-push schemes

Swarm-push schemes are less popular than swarm-pull schemes. $R^2$ is the only swarm-push scheme we surveyed. $R^2$ [26] addresses the dilemma between large duplicated chunks and large overhead of advertising bit-maps by using a large chunk size, immediate notice of chunk arrivals, and randomized linear network coding. In order to illustrate the trade-offs, we introduce the parameters in the simulation. The video streaming rate is 64 KB/s. The video is split into chunks of 180 KB (4 s of video). Each chunk is further split into $n = 180$ blocks. Each peer maintains a sliding window of 40 chunks, which can be described by a bit-map of five bytes. A peer notifies its neighbors whenever it receives or plays a chunk. A peer has 12 neighbors. The overhead of advertising bit-maps is low.

Each supplier randomly selects chunks that neighbors do not have to push. The number of receivers for each supplier is proportional to its upload capacity, and the receivers are randomly selected. All the peers try to maintain a playback delay of $\delta$, which is determined by the population. In the simulation of 800 peers, $\delta$ is set to 6 s (about half of that in the comparative swarm-pull scheme). Each chunk is independently coded with random linear code (a type of rateless code) such that the encoded block pushed by multiple suppliers to a receiver will have a high probability to be useful. Each supplier starts encoding after receiving $zn$ blocks, where $z < 1$. A smaller $z$ means the
peer is more aggressive to become a seed. Each supplier selects $m$ coefficients from the Galois field $GF(2^k)$ for each receiver independently. The coefficients are sent at the beginning of coded blocks. The receiver use Gauss-Jordan elimination to obtain the original blocks progressively. The redundant blocks is less than 5% in the simulation.

6.4. Comparative analysis

The performance of a swarm-based scheme is more influenced by its choice of system parameters, which often conflict with one another, than by its choice of the scheduling algorithm. For example, all the peers will receive all the packets when the resource index is large and a long playback delay can be tolerated. Three scheduling algorithms are compared in [87]. When the resource index is 2.2 and the playback delay is 30 s, in a system with 40–100 peers, the naive "random peer, random chunk" algorithm performs as good as the other two more sophisticated algorithms. This observation explains why commercial deployments that use simple scheduling algorithms usually have good performance. The overhead caused by advertising bit-maps and the optimum period-hop delay are determined by three conflicting parameters: the per-hop delay is computed assuming that a peer can pull a chunk immediately after knowing a neighbor has the chunk (i.e., peers have sufficient upload capacities). However, under the same conditions, the scheduling algorithm can make a difference.

6.4.1. FTR, error control, and delivery rate

In swarm-based schemes, peers usually use TCP for reliable transmission of video chunks. A supplier has a small probability to leave before sending all the packets of a chunk, and the swarm technique itself is multi-point retransmission error control mechanism. In this case, the receiver will find a new supplier for the chunk, which has the same effect as a multi-point re-transmission error control mechanism. Therefore, peer churn has little impact on the chunk delivery rate. As discussed earlier, the delivery rate is mainly influenced by the resource index and playback delay. In CoolStreaming, with 200 PlanetLab hosts and a playback rate of 500 Kb/s, 97% of packets arrive at peers in time when peers buffer packets for 10 s before playing. Commercial deployments usually have a packet delivery rate close to 100%.

Liang et al. [87] observe that the packet delivery rate improves more when increasing the upload capacity of the video server than increasing the upload capacities of other peers. This observation implies that placing high-capacity peers closer to the video server on the propagation trees can improve the packet delivery rate.

6.4.2. Tree cost and network traffic

None of the surveyed schemes reports the tree cost, but it is obvious that these schemes have a high tree cost. Since peers do not consider cost when selecting neighbors to pull or push chunks, the average cost of propagation tree edges is the same as that of the neighboring graph edges. In all the schemes, peers construct a random neighboring graph, and hence the tree cost is high. We remark that swarm-based schemes can reduce network traffic if peers select nearby peers when constructing the neighboring overlay, but this measure will have unfavorable side-effect (refer to Section 2.22).

6.4.3. Tree height, node degree, and peers’ upload workload

Swarm-based schemes typically have a small tree height. Zhang et al. [41] study the propagation paths in swarm-pull schemes where peers select rarest chunks and request from neighbors that have the least upload workload, and show that the propagation trees have a height comparable to a DBSPT (w.r.t. hops). These results show that the rarest-first or latest-first chunk-selection policy can build short propagation trees. Prime reported having a tree height of $\log N + K_{\min}$. (Prime uses homogeneous peers and there is no bandwidth bottleneck.) In CoolStreaming, the authors show a snapshot of a propagation tree, which is balanced and short.

All the surveyed swarm-pull schemes guarantee that a peer’s workload is within its upload capacity (given that the resource index is greater than one). In Chainsaw, a peer limits its output queue length to each neighbor. In CoolStreaming and Pulse, a peer pulls a chunk from a neighbor only if the estimated arriving time of the chunk is within the playback deadline. In LayerP2P, the supplier allocates its upload bandwidth to neighbors (according to the tit-for-tat policy) in each interval. Prime limits the number of neighbors a peer can have such that “bandwidth bottlenecks” cannot occur. This method will cause peers’ upload capacities under-utilized if peers have heterogeneous upload capacity.

In swarm-push schemes, a peer can allocate its upload capacity to neighbors, and hence a peer’s upload workload is bounded and its upload capacity can be more efficiently utilized than in swarm-pull schemes. $R^2$ makes the number of receivers of a peer to be proportional to the peer’s upload capacity to further improve the utilization. $R^2$ is compared with a swarm-pull scheme in [26]. When the resource index is between 1 and 1.5, the chunk loss rate is between 0.02 and 0.37 in $R^2$ and more than 1.5% in the comparative scheme.

<table>
<thead>
<tr>
<th>Chunk size (ms)</th>
<th>Setting A</th>
<th>Setting B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer size (s)</td>
<td>23</td>
<td>1000</td>
</tr>
<tr>
<td>Gossip interval</td>
<td>Immediate</td>
<td>1000 ms</td>
</tr>
<tr>
<td>Overhead</td>
<td>213%</td>
<td>0.1%</td>
</tr>
<tr>
<td>Per-hop delay (ms)</td>
<td>237 ms</td>
<td>1225 ms</td>
</tr>
</tbody>
</table>
6.4.4. Peer join complexity
Peer churn causes little overhead. A new peer takes one gossip interval to receive bit-maps from neighbors and begin to pull chunks. When a peer leaves, its neighbors simply delete the peer from their neighbor list. However, since swarm-pull schemes have a large chunk buffering delay, users usually have to wait for more than ten seconds after tuning into a video channel (e.g., PPLive).

6.4.5. Maintenance overhead
In swarm-based schemes, each peer need to maintain the bit-maps of neighbors and advertise its own bit-maps. Peers also need to exchange messages to arrange the transmission of video chunks. CoolStreaming reports communication overhead of 2%. \(R^2\) has lower communication overhead for advertising bit-maps but reports having 5% duplicated packets. In \(R^2\), peers also have heavy processing overhead to perform network coding.

To summarize, the performance of swarm-based schemes are heavily influenced by the resource index and playback delay. Commercial deployment can achieve smooth playback at a playback rate of 300–500 Kb/s and a playback delay of a few minutes. FTR is not needed and the swarm technique itself is a multi-point re-transmission error control mechanism. When peers use the rarest-first or latest-first policy in the scheduling algorithm, the propagation tree will be short. All the swarm-based schemes construct a random neighboring graph and have a high tree cost. Peer churn cause little overhead and has little impact on the packet delivery rate. The communication overhead is mainly caused by advertising bit-maps, and if using the swarm-push mode, by pushing duplicated chunks.

7. Unstructured tree-based schemes
In this section, we first discuss the algorithmic choices specific to unstructured tree-based schemes, then survey data-driven and network-driven schemes respectively, and finally give comparative analysis.

7.1. Algorithmic choices
In distributed tree-based schemes, a child must renew the parent–child relationship every \(T_m\). A parent forwards each received packets to its children for a period of length \(T_m > T_n\). As long as each peer has a parent and there is no forwarding loop, the parent–child relationships between peers will collectively form a tree.

Forwarding loops present a major challenge to distributed tree-building algorithms and are the focus when designing IP routing protocols. At the application layer, each packet can be assigned a unique sequence number, and peers use it to detect and discard duplicated packets. This means that loops are less harmful in P2P live video streaming applications than in IP multicast—a loop will only cause one redundant copy for each packet and peers in the loop not receiving new packets. P2P live video streaming schemes deal with loops in two ways. The first choice is to tolerate them, if they have a small probability to occur and can be quickly broken. Most schemes select this choice. The second choice is to prevent loops using a method similar to that of BGP [88]: each peer \(x\) checks potential parent \(y\)'s root-path and will not request peer \(y\) to be its parent if peer \(x\) is on peer \(y\)'s root path. This method is used in Narada [65], Gossamer [66], and ChunkySpread [70].

Unstructured tree-based schemes have two choices to build trees: the switch-parents method and the swap-positions method. In the switch-parents method, every interval of length \(T_{sch}\), a peer advertises its routing information to neighbors. Every interval of length \(T_{sch}\), a peer checks whether to switch to a new parent or to renew the relationship with its current parent. Each peer \(i\) ranks its neighbors according to a utility function \(f_i\), and selects the neighbor \(x\) that makes \(f_i(x)\) minimum as its parent on the tree rooted at peer \(s\). There are numerous choices to the utility function. For network-driven schemes, it is usually a function of network cost; for data-driven schemes, it is a function of data availability.

In the swap-positions method, every interval of length \(T_{sch}\), a peer checks whether to swap positions with its ancestors (or attaches to an ancestor if it has spare bandwidth) to optimize a cost function. Note that all the schemes that use the swap-positions method are network-driven, and the neighboring graph changes when peers swap positions. When a new peer \(i\) arrives, it randomly select an existing peer \(j\) to be its parent. If peer \(j\) is already on the tree, the addition of peer \(i\) still makes a tree. If peers arrive in flash crowd, a peer can try other peers randomly until it finds a peer on the tree. The system can converge to a tree in a short time, as is shown in [70]. The swap-positions method does not rely on the existence of the neighboring graph. For example, OMNI [68] is tree-first.

Two schemes intentionally add ”cross-links” to the tree. Bullet [89] and PRM [90] assume the existence of a tree. Bullet adds cross links to improve throughput. In Bullet, peers lack the bandwidth to push the video to their children. The video is split into chunks, which are further split into blocks. Blocks are pushed along the tree in such a way that each block is equally likely to appear at any node on the tree. Each peer then requests the peer with the most different content for missing chunks using a cross link. PRM adds cross links to reduce packet loss. In PRM, upon receiving a packet from its parent, a peer forwards the packet to a small number of randomly picked peers via cross links. These peers then forward the packet to their parents and children.

7.2. Data-driven tree-based schemes
Data-driven tree-based schemes include the new CoolStreaming [63], SPANC [27], GridMedia [64], and Substream Trading [23]. They all split the video into \(S\) substreams. CoolStreaming and SPANC uses peers’ positions in the video to build trees. In CoolStreaming, a peer tries to maintain that it advances at similar pace in each substream and its parent advances at similar pace as its neighbors. Therefore it is easier for an orphaned peer to find a parent and less packets will be lost when a peer switches parent. In SPANC, a peer selects the neighbor with the latest
position in the video to be parent to reduce delays. GridMedia and Substream Trading uses exchange history. In GridMedia, a peer subscribes to the peer that it has received more packets in the last interval with higher probability. The rationale is that such parents are more capable of supplying packets in the future. In Substream Trading, peers reciprocate with contributing neighbors as a tit-fortat incentive mechanism.

In the new CoolStreaming [63], the neighboring overlay construction is simplified. A new peer \( x \) contacts a well-known RP for its membership table of size \( M \), from which it selects neighbors randomly. Peer \( x \) exchanges its membership table with a neighbor only once, immediately after establishing the neighboring relation. When a neighbor becomes unavailable, \( x \) deletes the neighbor from its membership table but will not flood the information.

The video is split into chunks of equal size and interleaved into \( S \) substreams. Neighbors periodically exchange bit-maps. The bit-map peer \( x \) sends to peer \( y \) consists of two parts: a \( S \)-tuple of sequence numbers of the latest received chunks for every substream, and a \( S \)-tuple indicating which substreams peer \( x \) subscribes to peer \( y \). A peer always accepts subscription requests from its neighbors until it has \( M \) children. If a parent does not have enough upload capacity, its children compete for the upload capacity. A parent will not voluntarily drop a child; it is up to the child to decide whether to switch parents. A peer tries to maintain that all the substreams it receives advance at similar pace, and its parent for a substream proceeds at a similar pace as its neighbors. A peer monitors its parents for two upper bounds: the upper bound of acceptable deviation between substreams, and the upper bound of acceptable deviation between a parent and other neighbors. If they are not satisfied, the peer will start looking for a new parent.

SPANC [27] aims to minimize the packet delivery delay. It uses an existing neighboring overlay construction scheme. The video is split into chunks of \( t \) seconds and interleaved into \( S \) substreams. Each packet has a sequence number and each peer advertises its latest packet sequence number in each substream to neighbors periodically. A new peer \( x \) is assigned a set of potential parents, and each parent reserves a certain amount of bandwidth for peer \( x \). Given the latest packet sequence number of potential parents in each substream and the delivery delay \( d(y, k) \) from the source to peer \( x \) if peer \( y \) is the parent in substream \( k \), peer \( x \) calculates a “schedule” that minimizes \( \sum_{k=1}^{S} d(p_k, k) \), where \( p_k \) is peer \( x \)'s parent in substream \( k \). Every interval of length \( T \), if its network conditions have changed (e.g., departure of parents or change of packet loss rate), peer \( x \) calculates a new schedule.

Unlike most other schemes, SPANC considers lossy Internet links and uses network coding for error control. For each chunk of \( i \) original packets, \( j \) extra NC packets, which are linear combination of the original packets, are generated by parents of peer \( x \). Peer \( x \) can reproduce the \( i \) original packets if it receives any \( i \) packets of the \( i+j \) packets. Every interval of length \( t \), peer \( x \) calculates the value \( j \) as the sum of the exponential moving average and variance of the number of lost packets in previous chunks. Peer \( x \) then assigns the \( j \) NC packets to parents in such a way that minimizes the worst-case delay over all parents.

GridMedia [64] splits the video into \( S \) substreams and uses packet-sized chunks. A new peer contacts a well-known RP for its initial membership table, and exchanges membership tables with neighbors periodically. A peer selects some neighbors that have the shortest RTT and selects other neighbors randomly. This effort aims to mirror the overlay to the substrate while retaining the favorable property of random graphs. However, because the membership table is small compared with the population, the probability of choosing nearby neighbors is low. Neighbors periodically exchange keep-alive messages; a departing peer notifies its neighbors and the message is flooded within a limited number of hops. Each peer keeps track of the data packets sent to and received from neighbors, and drops a neighbor if the volume is below a certain level.

The pull mechanism is similar to the original CoolStreaming [49]. The pull mechanism is used in the start-up phase when trees do not exist and as an error control mechanism. GridMedia requires that peers have synchronous clocks; every peer synchronizes with RP when joining the system. At the end of every \textit{subscribing-pushing-packets-interval}, a peer \( x \) requests substreams from neighbors, which push packets of the subscribed substreams to peer \( x \) during the next interval. Peer \( x \) uses the roulette wheel selection algorithm to select parents for each substream; it selects a neighbor \( y \) to be the parent with a probability proportional to the percentage of packets it has received from peer \( y \) during the last interval. Peer \( x \) pulls the remaining packets and the lost packets from neighbors; it uses the same roulette wheel selection algorithm to select the neighbors to pull from.

Substream Trading [23] aims to design an "open" live streaming system in the sense that the incentive mechanism can function when the protocol and source code are open to the public. The video is encoded into \( S \) layers with LC. The scheme requires fine-granular substreams and \( S \) is set to be 20 in the simulation. Higher layers are dependent on lower layers and hence are assigned with smaller weights. Peers with fewer layers will watch low fidelity videos.

A peer \( x \) obtains its membership table from RP and requests some members to be neighbors. Assume peers \( x \) and \( y \) are receiving \( S_x \) and \( S_y \), \( S_x \geq S_y \), substreams respectively. If peer \( y \) receives a request from peer \( x \), it will grant the request since peer \( x \) has more substreams. However, if peer \( x \) receives a request from peer \( y \), it will grant the request with a probability of \( \frac{S_y}{S_x} \); i.e., a peer that is receiving fewer substreams is more likely to accept a neighbor. Peers trade substreams using the tit-for-tat policy; i.e., a peer \( x \) rewards peer \( y \) with \( k \) substreams if it receives \( k \) substreams from peer \( y \). A peer will only donate substreams if it is receiving all the substreams. Peers periodically query their neighbors which substreams they have and use an algorithm to select neighbors to trade substreams. Given a list of neighbors, the substreams they have, and the weight of each substream, the algorithm maximizes the total weighted substreams. This is a maximum weight matching problem in a bipartite graph and has complexity of \( O(S^4) \).
7.3. Network-driven tree-based schemes

Network-driven schemes include Narada [65], Gossamer [66], OMNI [68], ChunkySpread [70], Treecbone [71], Fastmesh [69], and TreeClimber [67]. Narada, Gossamer, and TreeClimber use a distance-vector routing protocol. Narada and Gossamer use RTTs as the metric, and TreeClimber uses hops. Narada allows any peer to be the video source and each peer maintains a routing table of size \(O(N)\). In Gossamer, peers only maintain routes to video sources and hence have a routing table of size \(O(s)\), where \(s\) is the number of sources. TreeClimber allows a single video source and the routing table has only one entry. In OMNI, ChunkySpread, Treebone, and Fastmesh, peers swap positions with other peers to optimize the tree. In Treebone and Fastmesh, peers swap positions only with ancestors. In ChunkySpread and OMNI, the selection of swapping targets is more flexible. OMNI is tree-first; all others are mesh-first.

Narada/ESM [65] aims to replace IP multicast for small group multicast applications. The protocol, called Narada, mimics IP multicast and strives to minimize the cost. It first constructs a mesh neighboring overlay \(G\) that mirrors the substrate Internet. Each peer periodically measures its delay to every other peer and adds or drops edges on \(G\) according to the utility gain or loss, which are defined in terms of the number of peers to whom the delays via overlay edges become less or more and the extent of changes. Then peers run a BGP-like protocol to maintain a unicast routing table. Because peers’ out-degrees on the tree are bounded by their upload bandwidth, Narada uses a variant of the shortest widest path algorithm. Similar to BGP, each routing entry contains a list of peers on the path to the destination to prevent forwarding loops. Finally, peers construct a source tree on \(G\) using RPF for each source. Narada achieves low-cost trees at the expense of scalability. Each peer needs to monitor the delays from itself to every other peer to optimize the overlay \(G\).

Gossamer/Scattercast [66] aims to connect IP multicast islands on the Internet using a set of strategically placed agents. The protocol, called Gossamer, organizes agents in a manner similar to Narada. Gossamer is also used in RMX [91], which focuses on reliable multicast. Each agent consists of a front-end, a network probe module, and a set of Scattercast proxies (SCXs). For convenience of expression, we ignore the internals of an agent and treat an agent as an SCX. SCXs connect to one another using IP unicast to form a neighboring overlay \(G\). An SCX’s degree on \(G\) is determined by its bandwidth. A SCX forwards packets to its clients using either unicast or multicast depending on whether the client is on the same IP multicast island as the SCX (see Fig. 9). Each multicast group may have more than one sources; an SCX is called a source SCX if it has a client that is a source.

Each multicast group corresponds to a Scattercast session, which has a unique URL-like name. A client \(x\) that wishes to join a multicast group connects to a nearby SCX. The SCX contacts a well-known RP for a list of SCXs in the session. SCXs in a session periodically advertise member tables and update their own member tables accordingly. SCXs run a BGP-like routing protocol on \(G\), with delays as the cost metric. As BGP, the routing messages include the path information to avoid loops. Each SCX maintains a routing table, which contains entries only for source SCXs. Gossamer periodically optimizes the overlay \(G\) by evaluating a utility function using the routing table. A new overlay link is added if it results in better routes toward source SCXs, and an existing link is removed if it does not result in worse routes.

TreeClimber [67] targets large-group multicast applications that allow tens of seconds of delay. The scheme tries to reduce network cost while maintaining a packet delivery rate and playback delay similar to data-driven tree-based schemes. The scheme first constructs the overlay to be a “small-world” graph\(^{11}\). Most edges are “short” (i.e., have a low cost) and hence any tree on the overlay will have a low cost. The original scheme [67] adds “long” edges randomly; a later improvement [42] adds long edges according to a distribution such that the overlay exhibits an hierarchy. The overlay is constructed by the use of RP. A new peer reports its virtual network position and upload bandwidth to RP. RP maintains a representative subset of the population and provides the new peer with a membership table. On the overlay, the scheme tries to build a short and balanced tree. It uses a heuristic degree bounded shortest path tree (in terms of hops) algorithm, which make the propagation tree to follow the hierarchy and has a height of \(O(\log N)\). The tree-repair time is same as data-driven schemes. When a peer finds its parent has left, it immediately switches to the peer with the smallest root-path hops. The scheme uses the multi-point re-transmission technique to recover erroneous packets. Peers exchange bit-maps and request missing chunks from one another.

OMNI [68] uses proxies, called Multicast Service Nodes (MSNs), to multicast from one source to a large number of users. Unlike in Gossamer, each user connects to an MSN using IP unicast. MSNs are organized into a tree rooted at the root MSN (i.e., the MSN that the source connects to). OMNI formulates the tree-building as a degree-bounded minimum average delay problem. The overlay is assumed to be a complete directed graph \(G(V, E)\). Each MSN is a node and has a out-degree bound on the tree determined by its upload bandwidth. Each edge \(xy \in E\) has a delay, which is the substrate IP unicast path delay. Queuing delays at MSNs and transmission delays are ignored. Each MSN \(x\) serves \(c_x\) users. Therefore, each user’s delay is the sum of the propagation delays of the user’s root-path on

---

\(^{11}\) A small graph is a graph with a diameter of \(O(\log N)\) and a large clustering coefficient
the overlay. The objective is to minimize the average delay of all the users.

The scheme uses a heuristic algorithm that consists of an optional initialization process and an optimization process. A new MSN contacts the root MSN to join the tree. If users join as a flash crowd before the video starts, the root MSN attaches MSNs in order of delays to itself to fill up the degree bounds of MSNs already on the tree, which results in a “good” initial tree. MSNs optimize the tree by swapping positions such that MSNs serving more users are moved closer to the root MSN. Each MSN maintains the root-path and aggregate subtree latency of its ancestors and children. An MSN can attach to its grandparent (i.e., be promoted) if its grandparent has spare capacity. An MSN can swap positions with its ancestors if the swapping reduces the total latency of their descendants. An MSN can also swap with another randomly selected MSN to reduce delays. When an MSN leaves, one child is promoted and the other children attach to the promoted child.

**Fastmesh** [69] targets P2P systems with “mild” peer churn. The authors formulate the tree-building as a minimum delay multiple trees (MDMT) problem. The overlay is assumed to be a complete directed graph $G(V, E)$. The cost of an edge $xy \in E$ is the worst-case delay from peer $x$ to $y$. The video is split into $S$ substrates and $T$ trees are constructed. Each peer is associated with a maximum out-degree on all the trees, which is determined by the peer’s upload bandwidth. Denote $d_x$ as the maximum delay of peer $x$ from the source on all the trees. The objective is to find the substrate trees that has min max $d_x, x \in V$.

The scheme uses a rather simple heuristic algorithm. A new peer $x$ contacts RP for a list of peers already in the system. Peer $x$ evaluates the “power” of each peer $y, P_{xy} = \frac{d_y}{r_y + s_y}$, where $r_y$ is the residual bandwidth of peer $y, s_y$ is the streaming rate, and $c_{xy}$ is the delay from peer $y$ to $x$. Peer $x$ greedily selects substream parents with higher powers. The idea is to select peers that have a small delay to the source and large residual bandwidth. It is worth noting that, in the problem formulation, the queuing delays at peers are not considered, which is usually the major part of the total packet delivery delay. Because the use of $r_y$ in the formula, the queuing delay is effectively addressed in the heuristic. Each peer periodically checks its ancestors’ upload bandwidth and swap position with an ancestor if it has less bandwidth. Quali et al. [92] extend Fastmesh and proposed several other power functions.

**ChunkySpread** [70] targets multicast applications that allow a delay of 10–20 s. The scheme first constructs a random neighboring overlay using a weighted random walk algorithm called Swaplink [93]. Each peer’s degree on $G$ is proportional to its target load. The video is sliced into $n$ substrates. The source $s$ selects $n$ peers, called stripe sources, and forwards each a substrate. Each stripe source is the root of a tree. A separate tree is built for each stripe source on the random neighboring overlay $G$.

Each peer periodically advertises its load, delay to the sources, and Bloom filter for each substream to neighbors.

Bloom filter is an alternative to the BGP-like loop prevention mechanism. It uses a smaller storage space to test whether an item is in a set; it has no false negatives but may have false positives. ChunkySpread optimizes the trees first by loads then by latencies. Each peer monitors the workload of its parent and neighbors, and drops overloaded parents and adds underloaded neighbors as parents. When all of a peer’s parents have the target load, the peer attempts to switch parent to reduce its delay to the source.

**Treebone** [71] classifies peers into stable Treebone nodes and unstable outskirt nodes. The scheme constructs a “compact” tree using only stable peers as interior nodes of the tree. Suppose the video length is $L$, and a peer joins at time $t$. A peer is an outskirt node when it arrives, and it declares itself to be a Treebone node after staying in the system for $T(t) = (L - t) \times x$, i.e., the age threshold for a node arriving at time $t$ is 30% of the residual time of the video. To avoid the situation in which there is no Treebone node immediately after the video begins, a peer checks periodically and declares itself to be a Treebone node at time $s < T(t)$ with probability $1 - e^{-1}$. The motivation is that older nodes tend to stay longer [94,95], and the lifespan of peers typically obeys a Pareto distribution. The authors claim that $x = 0.3$ achieves the best result.

Similar to the original CoolStreaming [49], peers maintain a membership table using SCAMP [77] and randomly select some as neighbors. A peer $x$ only requests Treebone nodes to be its parent. The tree is optimized using two methods: (1) if a peer finds that its parent has fewer children than itself, it switches position with its parent; (2) if a peer finds that its grandparent has spare bandwidth, it attaches to its grandparent. The sliding window is sequentially divided into a pull window and a push window, with the push window storing the latest chunks. Peers exchange bit-maps with neighbors and pull chunks that fall into the pull window.

### 7.4. Comparative analysis

We first compare swarm-based and tree-based schemes as a whole, and then compare individual tree-based schemes and summarize the results in Tables 7 and 8. In the tables, $n$ refers to the number of neighbors a peer has, $Dia(G)$ refers to the diameter of the neighboring graph $G$.

Comparative studies between swarm-pull and tree-based schemes are conducted in [96,97]. Ref. [96] compares Prime and a centralized tree-based scheme and reports that the former “consistently exhibits a superior performance”, but the experiment settings also favor the former. Ref. [97] compares GridMedia with a swarm-pull scheme using more practical settings, and reports that GridMedia is “more effective in throughput” and has “far lower delay and much smaller overhead”. Also recall that CoolStreaming, which originally uses a swarm-pull scheme, later switches to a data-driven tree-based scheme to reduce the playback delay. We argue that, compared with swarm-pull schemes, tree-based schemes with the multi-point re-transmission mechanism can achieve the same packet delivery rate. Since only a small fraction of

---

12 The “worst-case scheduling delay” in [69] is actually the transmission delay.
FTR, error control, and delivery rate

All the tree-based schemes have an FTR mechanism. The delivery delay of a scheme is mainly determined by its error control mechanism. In tree-based schemes with multi-point re-transmission error control mechanism, including both data-driven and network-driven schemes, if the playback delay is large enough (e.g., 20 s), all the schemes can achieve a packet delivery rate close to 100%, as evidenced in CoolStreaming, GridMedia, and TreeClimber. In these three schemes, more than 98% of the packets are pushed along trees; the rest are recovered by the multi-point re-transmission mechanism.

SPANC uses network coding and is one of the few schemes that considers Internet link errors. The scheme simulates with 200–1000 peers on a random neighboring overlay where each edge has 2–10% error rate. Results shown that, with 1000 peers, SPANC achieves less than 1% packet loss rate and about 2 s delivery delay, about 30% and 50% less than GridMedia-like schemes. However, link errors in the simulation are not bursty. The paper also reports that it takes several seconds for peers’ downloading rates to recover after a substream parent leave. This observation implies that the coding technique, although can achieve a short delay, are not effective against bursty packet loss.

Narada showcases the packet loss rate for schemes without an error control mechanism. Narada had been used to broadcast several academic conferences on the Internet; “80–90% hosts experience[d] loss for less than 5% of their session” [6]. Note that the experiment is conducted using stable helper hosts with good connections.

Tree cost and network traffic

Distributed tree-based schemes can achieve a tree cost, but their tree cost cannot be as low as centralized or recursive schemes. Except OMNI, all the schemes have a neighboring overlay. OMNI aims to optimize the average (propagation) delay of peers’ root-paths, not the tree cost, but this optimization will reduce the tree cost considerably.

The neighboring overlays constructed in TreeClimber and Narada have low cost edges. TreeClimber uses a central server that has all the peers’ virtual network positions. In Narada, each peer measures the RTTs to every other peer. (This method scales poorly.) Gossamer, Fastmesh and GridMedia also attempt to reduce neighboring overlay edge cost. However, in these schemes, peers only measure the RTTs to a list of randomly selected peers and select...
neighbors from them. In a large system, it is unlikely that this list will contain the closest peers. Gossamer has an optimization step: peers periodically add edges to and drop edges from the neighboring overlay to reduce the tree cost. Therefore, Gossamer is likely to have a lower tree cost than Fastmesh and GridMedia. All other schemes construct a random overlay and most of them have a high tree cost.

When building the trees, Narada and Gossamer aim to minimize peers root-path cost, and hence reduce the tree cost. TreeClimber aims to minimize peers’ hop counts to the video source, which reduces peers’ root-path costs but not tree cost. SPANC aims to minimize the delivery delay. Because the queuing delay is typically larger than the propagation delay, SPANC cannot have a low-cost tree even if propagation delay of a link is used as its cost. Chunky-Spread considers peers’ load and delivery delay, but the scheme uses TCP and hence propagation delay is not correlated with the delivery delay. CoolStreaming, Treebone, Substream Trading, and Gridmedia do not consider the distances when selecting parents.

7.4.3. Tree height, node degree, and peers’ upload workload

The tree height is not reported in all four data-driven schemes. Zhang et al. [41] study the propagation path of chunks in tree-based schemes where a peer selects the neighbor with the latest chunks as its parent. They find that the tree grows to a large height in the presence of peer churn. Narada, Gossamer, and TreeClimber build SPTs on the neighboring overlay; they also try to mirror the overlay to the substrate Internet. Therefore, the tree is typically short but is influenced by peers’ positions on the substrate Internet. In Fastmesh, each peer tries to swap positions with its ancestors if it has a larger bandwidth. Because peers with larger bandwidth are closer to the source, they have a small tree height. Treebone perhaps has the shortest tree because it compacts the tree. In OMNI, proxies swap positions to reduce aggregate subtree latency; proxies that serve more peers are moved closer to the source.

In Narada, Gossamer, TreeClimber, OMNI, SPANC, and Substream Trading, peers either employ CAC when handling requests or send requests only to neighbors with spare bandwidth to control peers’ upload workload. However, proxies in OMNI need to serve clients, and the number of clients are out-of-control. In GridMedia and CoolStreaming, peers compete for bandwidth. A peer always accepts subscription requests, and it is up to the children to switch to another parent. This method makes efficient use of peers’ upload capacity, but may cause peers to be temporarily overloaded. In Fastmesh, peers select parents according to their “power”, which is proportional to their residual bandwidth. This method also makes efficient use of peers’ upload capacity.

7.4.4. Peer join complexity

In all the data-driven schemes, a new peer attaches to the tree after receiving bit-maps from its $n$ neighbors. Peers already on the tree will not change parents. In network-driven schemes that aim to optimize a utility function, a new peer can attach to the tree right after receiving routing information from neighbors, but it may take up to $Dia(G)$ routing update intervals for the system to stabilize. The utility function is not optimized in the meantime, and some peers need to change parents, which will result in lost packets. Treebone has low peer join complexity because a new peer is an out-skin peer, but the peer will swap positions with other peers after it becomes a Treebone peer.

7.4.5. Maintenance overhead

In distributed tree-based schemes, peers have reasonable processing overhead. The communication overhead is typically lower than in swarm-based schemes even when a multi-point re-transmission mechanism is used. This is tree-based schemes can use a smaller bit-map. Peers in Narada have the largest overhead. Narada first runs a unicast routing protocol and then build trees by reverse path forwarding (RPF). This method is reasonable for IP multicast because the Internet already has IP unicast protocols, but cannot be justified at the application layer.

To summarize, in unstructured tree-based schemes, peers have stable workload and hence can achieve a higher throughput in a resource-scarce system than in swarm-based schemes. Peers usually use CAC to control workload. It is possible to control the propagation tree height or build a low-cost tree on a given neighboring overlay in network-driven schemes, but it is hard in data-driven schemes. A scheme can reduce the propagation tree cost when constructing the neighboring overlay and when building the trees on the neighboring overlay. But selecting nearby peers as neighbors often results in side-effects. In data-driven schemes, the arrival of peer incurs little overhead, and peers already in the system do not change parents. In network-driven schemes, the system may take a long time to converge, and packets may be lost when peers change parents. When packets are uniquely numberd, forwarding loops can be tolerated to certain extent, which make FTR easy to implement. A multi-point re-transmission mechanism is critical to a high packet delivery rate. It can be used in both data-driven and network-driven schemes at the expense of overhead comparable to (usually less than) swarm-based schemes.

8. Conclusions and open issues

In this paper, we provide a comprehensive and in-depth survey of P2P live video streaming schemes from an algorithmic perspective. We identify a series of design choices critical to system performance, evaluate schemes by their design choices, and identify the impact of design choices on system performance. Valuable lessons have been learned and some of the most important are summarized below.

Centralized and recursive schemes can support a small to medium group size, typically less than a thousand. For an arbitrary group size, a fully distributed algorithm should be used. Centralized schemes have the smallest communication overhead, and are not necessarily less scalable than recursive schemes because the latter have a long convergence time in the presence of peer churn. Both centralized and recursive schemes can have a tree cost, although recursive schemes can achieve a lower tree cost
than centralized schemes. Unless having special objectives, such as to maximize the propagation tree’s throughput (e.g., Overcast), centralized schemes are preferred over recursive schemes.

Almost all the fully distributed schemes first construct a neighboring graph, which can be structured or unstructured. Structured schemes use DHTs to provide a routing mechanism, which can be used to build short and balanced trees. However, DHTs are not a good fit for P2P live video streaming applications because peers are dynamic, their out-degrees are bounded, and a low tree cost is often preferred.

Unstructured distributed schemes can be swarm-based or tree-based. Swarm-based schemes are robust against peer churn and can achieve a packet delivery rate close to 100% at the expense of long playback delays (typically more than one minute in large systems). Swarm-push schemes have a smaller playback delay than swarm-pull schemes, but avoiding duplicated chunks is a big challenge. The packet delivery rate in swarm-based schemes are more influenced by the upload bandwidth in the system and the playback delay than by the scheduling algorithm, but the scheduling algorithm can make a difference under the same conditions. Algorithms that prioritize rarest and latest chunks and consider neighbors’ workload usually work good.

Tree-based schemes, including centralized, recursive, and distributed schemes, can achieve a much shorter playback delay than swarm-based schemes. If a re-transmission error control mechanism is used, the playback delay is several to tens of seconds; otherwise, it is less than a second. The packet delivery delay is mainly impacted by the error control mechanism. Without a re-transmission error control mechanism, the packet delivery rate is usually less than 95%. If a multi-point re-transmission error control mechanism is used, tree-based schemes can also achieve a packet delivery rate close to 100%. Therefore, tree-based schemes with a multi-point re-transmission mechanism are preferred over swarm-based schemes. In tree-based schemes, a fast tree-repairing mechanism is typically used to reduce the number of packets that need to be recovered by the error control mechanism.

Unstructured distributed tree-based scheme can be data-driven or network-driven. Data-driven schemes, including swarm-based schemes, typically have a high tree cost. All the swarm-based schemes and many data-driven tree-based schemes construct a random neighboring graph for its favorable properties. Some network-driven schemes mirror the neighboring graph to the substrate Internet to reduce the tree cost. In network-driven schemes, peers can select nearby peers as parents to reduce the tree cost, but peers need to switch parents more frequently, which increases the packets that need to be recovered by the error control mechanism.

Despite two decades of research, a number of issues remain open. We only discuss two open issues regarding the objectives of distributing packets to peers and reducing the tree cost. Many applications, such as video conferencing, remote education, and online gaming (action games), requires a playback delay of less than a second, which means re-transmission error control mechanisms cannot be used. One promising direction is to use MDC and build multiple interior node-disjoint trees. However, current MDC schemes have a high coding overhead and but it is hard to build interior node-disjoint trees that are robust against peer churn. Another promising direction is to use network coding. This method does not require interior node-disjoint trees, but practical network coding schemes are hard to design.

The second open issue is to reduce the Internet traffic generated by fully distributed P2P live video streaming schemes. Network-driven tree-based schemes can achieve a significantly lower tree cost than data-driven schemes, but the tree cost is still significantly higher than IP multicast trees. To reduce the tree cost, peers should construct a neighboring graph with low-cost edges and build degree-bound minimum weight trees on the neighboring graph. However, if peers only select nearby peers as neighbors, the neighboring overlay is prone to partitioning and has a large diameter and high clustering coefficient. These properties have negative impacts on the packet delivery rate and playback delay. Building a robust degree-bound minimum weight spanning tree is a challenge.

References


Xiangyang Zhang received his B.E. degree and M.E. degree in Electrical Engineering from Harbin Institute of Technology, China, in 1992 and 1995, respectively. He received his Ph.D degree in Computer Science from Queen's University, Canada, in 2012. Between 1995 and 2007, he was with Huawei Technologies, where he was a senior engineer and research manager in the data communication division. His research interests are in the fields of multimedia communication, peer-to-peer networks, Internet infrastructure, and telecommunication networks.

Hossam Hassanein is with the School of Computing at Queen's University working in the areas of broadband, wireless and variable topology networks architecture, protocols, control and performance evaluation. Dr. Hassanein obtained his Ph.D in Computing Science from the University of Alberta in 1990. He is the founder and director of the Telecommunication Research Group (TR Lab) http://www.queensu.ca/~trl in the School of Computing at Queen's. Dr. Hassanein has more than 350 publications in reputable journals, conferences and workshops in the areas of computer networks and performance evaluation. He has delivered several plenary talks and tutorials at key international venues, including Unconventional Computing 2007, IEEE ICC 2008, IEEE CCNC 2009, IEEE GCC 2009, IEEE GLOBECOM 2009, ACM MSWiM 2009 and IEEE Globecom 2009. Dr. Hassanein has organized and served on the program committee of numerous international conferences and workshops. He also serves on the editorial board of a number of international journals. He is a senior member of the IEEE, and is currently chair of the IEEE Communication Society Technical Committee on Ad hoc and Sensor Networks (TC AHSN). Dr. Hassanein is the recipient of Communications and Information Technology Ontario (CITO) Champions of Innovation Research award in 2003. He received several best paper awards, including at IEEE Wireless Communications and Network (2007), IEEE Global Communication Conference (2007), IEEE International Symposium on Computers and Communications (2009), IEEE Local Computer Networks Conference (2009) and ACM Wireless Communication and Mobile Computing (2010). Dr. Hassanein is an IEEE Communications Society Distinguished Speaker (Distinguished Lecturer 2008-2010).