A Survey on P2P Content Delivery Mechanisms

Nazanin Magharei
Department of Computer & Information Science
University of Oregon
nazanin@cs.uoregon.edu

Contents

1. Introduction

Most of the Internet applications are designed for delivery of content (data) between end-systems, *i.e.*, content delivery. The design and evaluation of content delivery mechanisms has been a popular area of research in the network community, as a significant portion of the Internet traffic consists of the traffic generated by some of these applications.This high traffic demand can be due to the following reasons: *(i)* high bandwidth requirement of such an application, *e.g.*, video-conferencing, *(ii)* large amount of data being delivered, *e.g.*, software-update, or *(iii)* large number of participating end-systems, *e.g.*, BitTorrent [1].

A content delivery application requires to distribute a certain type of content based on a communication model among a group of participants. Therefore, content type and communication model are the two orthogonal dimensions which determine the goals and challenges in the design of a content delivery mechanism. Content type can be broadly classified into elastic, *e.g.*, file and streaming, *e.g.*, video. On the other hand, communication model determines the flow of content between interested end-systems and can be one-to-one (*e.g.*, file download), one-to-many (*e.g.*, live event broadcasting) and many-to-many (*e.g.*, video conferencing).

During the past decade, the following three major architectures have been used to support delivery of content over the Internet: *(i) unicast or client-server*, *(ii) IP multicast*, and *(iii) peer-to-peer*. The traditional unicast or client-server architecture allows a single server to exchange content to clients in a one-to-one communication model. However, this architecture does not scale with the number of clients due to the limited server's resources (*i.e.*, bandwidth and CPU). Many applications require one-to-many or many-to-many communication model. Multiple unicasts is not scalable and efficient for that purpose which motivates IP multicast architecture. By relying on the core of the network, IP multicast allows delivery of a single copy of the content from a single server to any number clients in a convenient and efficient fashion. The limited availability of IP multicast due to its deployment difficulties has motivated a new architecture called peer-to-peer (P2P) which pushes the delivery functionality to the edge of the network. This architecture has received a great deal of attention for support of content delivery over the Internet. In P2P, participating peers form a P2P overlay and forward the content through the overlay links. Each peer acts as a client, which receives the content as well as a server, that forwards the content to other participating peers. The growing interest in P2P for content delivery is due to two major reasons: *(i)* This architecture does not require any special support from the network, which leads to its ease of deployment and low cost. *(ii)* P2P is self-scaling, as the total resources (*e.g.*, bandwidth, CPU) organically increase with the number of participating end-systems.

P2P offers new opportunities to support various applications but introduces new challenges. It improves the performance of a wide range of content delivery applications by allowing an efficient search for objects (*i.e.*, data files, storage or other parties) [2], [3], [4], [5] in one-to-one content delivery applications, a scalable and robust file distribution among a large number of end-systems [1] in one-to-many elastic content delivery, and the delivery of high quality stream to a large group of interested peers [6], [7] in one-to-many streaming content delivery. The key challenges of P2P comes from relying on a set of autonomous end-systems for delivery of content that may fail or willingly leave at any time, and might have heterogeneous and asymmetric access link bandwidth.

In this survey, we focus on P2P content delivery mechanisms. To systematically cover the design space of all content delivery applications, we identify the main components of P2P one-to-one, one-to-many and many-to-many applications. Further, we present the classification of existing approaches for each P2P application based on its components. For each approach, its various components, design choices and challenges are identified and discussed in great details. In addition, we compare and contrast the different approaches for each P2P application and elaborate on some case studies.

The rest of our report is organized as follows: in Section 2, we discuss the taxonomy of content

Delivery Architecture	Communication Model	Content Type			
		File	Stored Streaming	Live Streaming	
				Non-Interactive	Interactive
Unicast (Client-Server)	One-to-One	WWW	$[YouTube]$	Live Radio	Customer-service support chat
IP Multicast	One-to-Many	SW update	Tele-teaching	Live event broadcasting	X
	Many-to-Many	X	X	X	Tele-conferencing
P2P	One-to-One	$[KaZaa, Gnutella]$	Audio playback	Webcam sharing	Chat/$[Skype]$
	One-to-Many	$[BitTorrent]$	$[RedCarpet]$	Live event broadcasting	X
	Many-to-Many	X	X	X	Tele-conferencing

Table 1. Taxonomy of content delivery over the Internet.

delivery over the Internet. Section 3 and 4 present various types of content and communication model along with their specific characteristics, respectively. We proceed on existing delivery architectures by elaborating on unicast, IP multicast and P2P, along with some of their relevant issues in Section 5. In the next two sections of 6 and 7, we survey P2P one-to-one and one(many)-to-many content delivery applications, respectively. Section 8 discusses some of the related topics to P2P content delivery, *i.e.*, incentive mechanisms and network coding. Finally, Section 9 concludes the report.

2. Classification of Content Delivery Mechanisms

Requirements of content delivery applications depend on type of the content and the communication model. These two orthogonal dimensions can be employed to categorize content delivery mechanisms over the Internet. Content type can be *elastic* such as file distribution or *streaming* such as video streaming. The type of the content directly affects the design choices and goals of a content delivery mechanism. The other dimension is communication model which determines the pattern of content delivery. Communication model can be one-to-one (*e.g.*, File download), one-to-many (*e.g.*, live event broadcasting) and many-to-many (*e.g.*, video conferencing).

Machinery of the content delivery mechanism can be placed at the core of the network or the edge. Unicast and IP multicast delivery architectures rely solely on network core for delivery of content, whereas, peer-to-peer (P2P) architecture push the content delivery functionality at the edge.

Table 1 illustrates the taxonomy of content delivery over the Internet. The first column summarizes various delivery architectures as mentioned above which are Unicast or client/server, IP multicast and P2P. The second column shows different communication model supported by each architecture. The next columns are divided based on the type of the content and each box is an application class or an instance of application in the bracket for the corresponding row and column.

3. Content Type

Content delivery mechanisms can be divided into two categories, based on the type of content being delivered which are *elastic* and *streaming* content delivery.

An application for delivery of elastic or non-streaming content is often called *file distribution application*. It involves the distribution of (usually large) files from one sender to one or more receivers [8]. On the other hand, streaming applications refer to sending real-time streaming content over the Internet from one or multiple senders to a number of receivers.

3.1. File/elastic Content

In elastic content delivery, a file is usually stored at a sender and is delivered to receivers. There is no timing issue in delivery of file and typically the objective is to download/receive the entire file and then use it. Elastic content delivery mechanisms offer a reliable service while minimize the total delivery time. In such mechanisms, a sustained delivery rate is not an issue. In fact, neither the order of the packet arrivals, nor the time between delivery of each packet are important.

3.2. Streaming Content

In streaming content delivery, the content is streamed from a single or multiple sources to the receiver(s). Receiver can start playing as soon as a few packets are received. Stream has an inherent playback rate (*e.g.*, MPEG 1.4 Mbps) which depends on the coding rate. Streaming content delivery has *timing constraint*, that is once playback started/initiated, packets should be delivered before their playtime, otherwise an interruption will happen.

Streaming content can be categorized into *stored streaming* and *live streaming* [8].

3.2.1. Stored Streaming. Stored streaming is referred to streaming a pre-stored/cached audio or video from a source to interested receivers [9], [10], [11], [12]. An example application of stored streaming is On-demand streaming of stored movies or songs [13]. In stored streaming, the entire content is available and stored which gives control to both the source and receivers. Source can deliver the stream at a higher rate and receiver can choose the playtime (*i.e.*, VCR functionality which is fast forward, pause and rewind).

In stored streaming applications, receiver can delay its playback and starts buffering to absorb any variation of bandwidth and minimize probability of late arrival of packets. This flexibility results in a *loose timing constraint* for stored streaming applications.

3.2.2. Live Streaming. In live streaming, content is generated on the fly and can be delivered as soon as it becomes available. Therefore, the average sending rate is equal to the stream rate and source cannot send faster. Moreover, advancing playtime is not feasible in this context, as the future content is not available. Live streaming can be non-interactive or interactive.

Live Non-interactive Streaming Live non-interactive streaming refers to the synchronized distribution of live content from a single source to receivers. An example is broadcasting a live sporting event. In this context, a receiver can delay its playback and starts buffering. However, the delay should be small due to the live nature of the content. Therefore, live non-interactive streaming has a *relatively* loose timing constraint.

Live Interactive Streaming Live interactive streaming involves a group of users who use audio/video to communicate with each other in real time. Essentially, users interact with each other by exchanging timely data that is generated on the fly. Some examples of live interactive streaming are IP telephony, video conferencing and distributed interactive gaming applications. In this context, the distinction between the role of source and receivers is diminished as each user is able to generate content (speak or move) at any time and send it to other users. In live interactive streaming, receiver cannot delay its playback as it loses the interaction with other participants. This results in a *tight* timing constraint for such application.

3.3. Comparison

Content delivery mechanisms depending on the type of the content differ in some fundamental aspects and have various sets of requirements regarding delay, bandwidth and scaling. The distinctions between

Content Type	Application	Example	Time-constraint	Limited content availability	Scalability
Elastic/file	File-distribution	BitTorrent	X	X	High
Stored streaming	Video-On-Demand	Skycrapper	Loose	X	Medium
Live non-interactive streaming	Live sport event broadcasting	PPLive	Relatively loose	✓	High
Live interactive streaming	Video conferencing	VSee [16]	Tight	✓	Low
	IP telephony	Skype	Tight	✓	Low

Table 2. Comparing characteristics of content delivery applications.

these mechanisms lead to different goals and challenges in the design of such applications. The main differences between them can be pointed out as follows:

- *Content availability:* In file distribution and stored streaming applications, the full content is available at source, while in live streaming (whether interactive or non-interactive), the availability of the future content is limited. This limited availability of content in live streaming applications makes the design of such applications more challenging.
- *Timing constraint:* File distribution applications are not sensitive to end-to-end delay and do not have any timing constraint. On the other hand, streaming applications are sensitive to end-to-end delay; packets that incur a sender-to-receiver delay above a certain amount are essentially useless. The level timing constraint which determines the amount of allowed buffering at the receiver, is different among types streaming applications. Live interactive streaming applications have a tight timing constraint and thus, very demanding about end-to-end delay. For instance, in voice over IP applications, the end-to-end delay of more than 200ms will make a voice conversation unsatisfactory [14]. On the other hand, for non-interactive streaming applications, such as live video broadcasting, the delay tolerance is typically higher, due to lack of interactions among users. Between live and stored applications, the timing constraint is tighter in the former. In live streaming, the shorter the end-to-end delay is, the more lively the stream is perceived by receivers (referred to as liveliness in [15]). In stored streaming, liveliness is simply irrelevant because the stream is already pre-recorded.
- *Playing time:* In stored streaming applications, users can watch the video or listen to the audio at any arbitrary time. While, in live streaming applications (either non-interactive or interactive), all participants have a loosely synchronized viewing/listening time.
- *Scalability:* Ability to scale to a large group is often more of an issue for file distribution and live non-interactive streaming applications compared to the other two streaming applications. Huge live events have lots of interested users watching them live and simultaneously. Whereas, for stored streaming this is not common to happen. Live interactive applications that involves multiple users such as video conferencing are often used by small to medium sized groups due to the nature of conferencing.

Table 2 presents characteristics of various content delivery applications along with an instance of each class of applications.

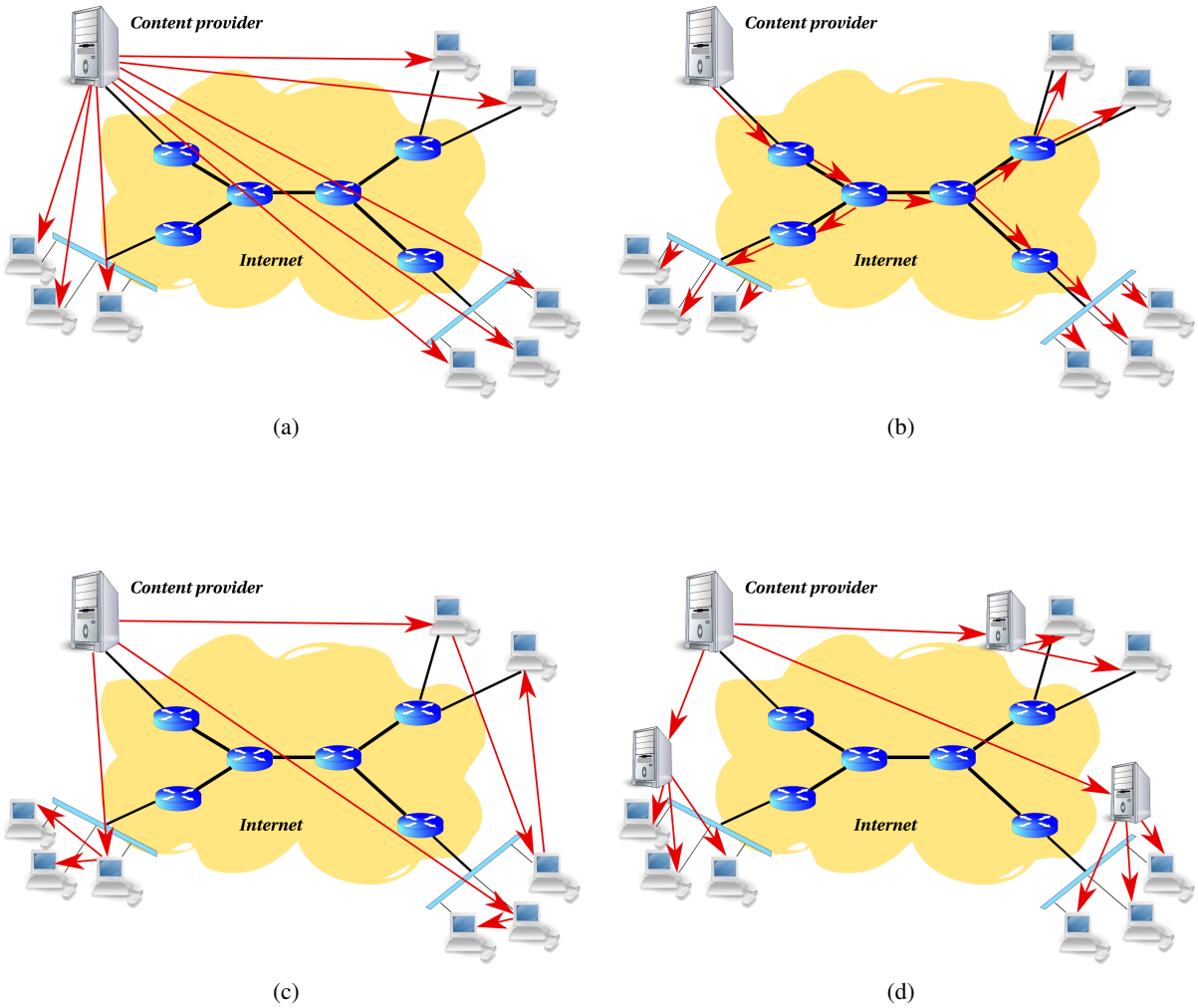


Figure 1. Various architectures for content delivery over the Internet- 1: Unicast/Client-server, 2: IP Multicast, 3: P2P, 4:CDN

4. Communication Model

Communication model determines the pattern of content delivery between participants. There are three communication models including one-to-one, one-to-many and many-to-many. In one-to-one model such as uploading/downloading a file, content is exchanged between two parties. One-to-many model, refers to a communication model in which the content flows from one end to a group of interested receivers, simultaneously, *e.g.*, live event broadcasting. In the third model which is many-to-many, content is generated by a number of participants and each generated content is simultaneously delivered to all participants. The many-to-many communication model usually delivers live interactive streaming content. In such a model, multiple sources interact with each other which can consists all the participants (*e.g.*, teleconferencing, often small scale) or a subset of participants (*e.g.*, watching a live online debate, usually large scale).

5. Delivery Architecture

In this section, we present existing architectures supporting the delivery of content over the Internet. Existing delivery architectures are unicast (client-server), IP multicast and P2P.

5.1. Unicast (Client-server)

Unicast or client-server is a traditional architecture for supporting content delivery over the Internet. It relies on a single server in a one-to-one communication model as depicted in Figure 1(a). When a client wants to exchange some content, it contacts a well-known server, and the server delivers the requested content to the client (*e.g.*, playback video or audio, webcam) or receives the content from the client (*e.g.*, upload a file).

Client-server cannot scale with the number of clients due to the limited server bandwidth. For instance, if the content is a live video with bitrate of 300 Kbps then with server access link bandwidth of 1 Gbps, this architecture can only support up to approximately 3,000 simultaneous clients. Moreover, the server is the single point of failure which results in a low resiliency as failure of the server (*i.e.*, problems in the server or its connectivity to the Internet) shuts down the whole content delivery to all clients.

5.2. IP Multicast

Many applications require one-to-many or many-to-many communications. Multiple unicast is not scalable and efficient as described in previous section. IP multicast was designed to reduce the total traffic injected into the network [17], [18], [19] in a scalable fashion. To utilize network resources efficiently and reduce network traffic, it simultaneously distributes a *single* copy of a packet to all the interested receivers. Multicast routers organize participating end-systems into a multicast group and duplicate the packet and deliver it to the participating end-systems. Figure 1(b) shows an IP multicast delivery architecture. [19].

IP multicast relies on the core of the network or IP layer. According to end-to-end argument [20], a functionality should be pushed to higher layers of the network if possible unless implementing it at the lower layer can achieve significant performance benefits that outweigh the cost of additional complexity. Deering in [21], argued that due to the significant performance achievement multicast should be implemented at the IP layer. In essence, routers participating in multicast, must know how to process and route multicast packets. Moreover, multicast routers must be capable of setting up and tearing down multicast sessions.These requirement from the network introduce a major burden in the way of the deployment of IP multicast in the Internet.

5.2.1. Research Topics. There are many research topics on IP multicast. Significant studies were performed on multicast routing []. In this report, we cover research topics that are related to supporting higher level functionalities in IP multicast, *i.e.*, congestion control and heterogeneous receiver support, reliability and synchronized user participation. Given the group membership management and delivery is performed by network, supporting these functionality is a challenging task in IP multicast.

Congestion Control and Heterogeneous Receiver Support: Sending to receivers with a wide variety of bandwidth capacities, latency characteristics, and congestion in the multicast is challenging. When two hosts communicate via unicast, it is relatively easy for senders to adapt to varying network conditions. The Transmission Control Protocol (TCP) provides reliable data transport, and can adapt to network variation. This is done in TCP, through a feedback loop (*i.e.*, receiving acknowledgments to detect losses) and correspondingly adjusting the sending rate.

However, with multicast delivery architecture; either one-to-many or many-to-many, there exists multiple receivers. If all receivers return loss reports simultaneously, the sender is easily overwhelmed with

feedback messages, which is known as the *implosion problem*. Many mechanisms were proposed to deal with this issue, through *shared learning* or *representative* [22], [23], [24], [25]. In shared learning, upon loss detection at a receiver, it starts a random timer. If it receives a loss report sent by another rec, it stops the timer. Otherwise, it multicasts a report when the timer expires. In representative [26], [27], [28], [29], [30], Some receivers may be designated as representative for sending congestion signals. However, all these mechanisms come with the cost of introducing extra delay in feedback. Feedback delay directly contributes to the responsiveness of congestion control schemes, the longer the delay the less the responsiveness.

The other issue affects the scalability, is *drop-to-zero problem* (also known as loss multiplicity problem) [26]. The problem arises when receivers use packet losses as congestion signals and the source uses these signals to regulate its transmission rate without proper aggregation. When packets are lost on multiple paths independently, receivers downstream of these paths will all send loss reports to the source resulting in multiple rate drops at the source. Some mechanisms proposed receiver suppression [26] to deal with this issue.

Another problem is that receiver's heterogeneity can vary widely due to the wide range of access link bandwidth capacities and dynamic fluctuation of connection bandwidth due to transient network conditions. If a sender adapts its send rate (and streaming quality) based on the loss rate of its worst receiver(s), then it can not satisfy all receivers [].

Congestion control or strategies for supporting heterogeneity in IP multicast can be classified into *sender-based or single-rate* [26], [31], [32], [33] and *receiver-based or multiple-rate* [34], [35], [36]. The sender-based is similar to what TCP does, the sender uses a single transmission rate and infers congestion from feedback collected from receivers. The sender-based is more suitable for for delivery of elastic content, where the primary goal is to reliably deliver content to all receivers and long delay for some receivers is tolerable. This strategy requires additional mechanisms to deal with implosion control and drop-to-zero problems as discussed above.

For streaming content delivery which are sensitive to delay but flexible to some quality reduction, a receiver-based approach is proposed [35], [36], [37], [38]. In this approach, streaming is encoded into layers and transmitted onto different multicast groups. Receivers can adapt their quality by joining groups for various layers based on current network condition.

Early proposed solutions for receiver-based, such as RLM [35], RLC [36], PLM [37] and RLS [38] incorporate a fixed layer transmission rate. Some following solutions such as FLID-DL [39], [30] and STAIR [40], still use fixed layer sending rates, but are more careful about the join and leave operations by receivers. Newer solutions such as MLDA [41] and [42], [43], allow the source to adjust the layer sending rates. The capability of adjusting the rate at both sides allows for more adaptability to network conditions. The receiver-based approach requires receivers to coordinate when join and leave a multicast group, causing more overheads and dependency on the underlying multicast routing protocols. Such approach are complicated in design.

Reliability: Some of the challenges involved with enabling reliable multicast transport are the same as those described before for supporting for heterogeneous receivers, and solutions are similar also.

There are two main classifications for supporting reliability in multicast. One approach is to use *sender-initiated* reliability, where the sender is responsible for detecting lost packets and subsequently re-sends it. Other approach is *receiver-initiated*, in which case the receiver is responsible for detecting lost packets and requesting them to be re-sent. In designing a reliable multicast scheme that is highly scalable, there are two issues. First, a sender-initiated scheme will require the sender to maintain state of each receiver. This may result in a state explosion. Second, there is feedback or loss report implosion as in the previous section.

Many proposed solutions avoid the implosion problem by using negative acknowledgments (NACK) and further NACK suppression [44], [22] similar to shared learning as described above. In NACK suppression,

NACKs are multicast so that all receivers detect that a NACK has already been sent. Receivers delay their NACK for a random time, hoping to receive a NACK for the same packet from some other receiver.A limitation of NACK suppression is that the timer delay calculations become complex with huge overhead with a large group of receivers.

Another approach for reliability is local repair which allows any receiver that receives a packet reply to a NACK request [22]. Because the receivers use a timer-based NACK suppression to minimize the number of receivers that respond, this approach has the same drawbacks as NACK suppression when the receiver set becomes large.

Polling is a sender-initiated technique to prevent implosions [45], [46]. The sender allows only a set of random receivers to request a re-transmission. Therefore, the sender can sample feedbacks and control the traffic coming from receivers. However, in a large group, obtaining an appropriate sample without causing implosion is impossible.

Although many reliable multicast protocol were proposed, none of them are optimal in solving all the involved challenges. Scalability is the significant concern, and it implies designing a "network friendly" multicast protocol that detects and avoids congestion as it provides reliable delivery for a large group of receivers.

Asynchronous Participation: IP multicast offers an inherently synchronous service. Delivery of content to a large number of asynchronous users with the goal of maximizing efficiency (*i.e.*, minimizing the network cost and redundant traffic) is another topic of research in this domain. Users' requests for elastic or streaming content are likely to be asynchronous. For example, in tele-teaching or file update applications, users might want to stream the content or download an update at different times.

Some studies proposed using periodic broadcasting [10], [47], [48], [49] for streaming to asynchronous users. In periodic broadcasting, segments of the streaming content (with increasing sizes) are periodically broadcasted on different multicast channels, and asynchronous users join one or more channels to receive the content.

Stream patching [50], [51], [52], [53] is another technique for supporting asynchronous participation. Patching [54] allows asynchronous users' join to catch up with an ongoing multicast session by requesting the missing portion through server unicast.

[52] proposed a merging technique to deal with this issues. Clients merge into a larger and larger multicast session repeatedly, thus reducing both the server bandwidth and the network link cost.

A different approach for delivery of elastic content through IP multicast, is the use of periodic multicasting the encoded content as was done over broadcast disks [55] using IDA [56], and more recently using the Digital Fountain approach which relies on Tornado encoding [57], [58]. These techniques enable users to reconstruct the original content of size N segments using any subset of N segments from a large set of encoded segments. These techniques are not able to accommodate real-time constraints of streaming content due to the necessity of encoding/decoding each N segments to construct each N segments of the content.

5.2.2. Limitations. Unfortunately, despite the tremendous research studies on IP multicast, its practical deployment issues have prevented it from being widely deployed [59]. We briefly describe some of these issues:

- IP multicast requires routers to maintain per-multicast-group state, which introduces high complexity and scaling constraints at the IP layer. The routers were originally designed to have a simple stateless architecture to be able to handle heavy traffic efficiently. To support IP multicast, each router should admit the multicast groups on which it receives traffic from and handle the join and leave requests from the end-systems for their groups. These new functionalities add extra overhead and complexity to routers.

- IP multicast requires infrastructural-level changes which slows down its deployment. Essentially, for the full deployment and widely available multicast server, IP multicast routers need to be installed at all levels of the network (from backbone to edge routers). This presents a significant cost and investment to ISPs.
- IP multicast does not provide solutions for many important higher level functionalities such as congestion control, reliability, multicast group management, multicast address allocation, loose synchronization among clients, and accommodating bandwidth heterogeneity of end-systems.
- IP multicast is vulnerable to Denial of Service and flooding attacks from unauthorized malicious users. For example, unauthorized users can flood any multicast group, with arbitrary content [60].

5.3. P2P

Due to the deployment issues of IP multicast, researchers have shifted their focus to peer-to-peer (P2P), where the multicast-related functionalities are moved away from routers towards end-systems or peers [61], [62], [63], [64], [65], [66]. In P2P, group management, content replication and routing are implemented on the application level without requiring any special support from the network. This is done by constructing an overlay, consists of unicast connections between participating peers on top of the current IP network. Peers collaborate with each other to replicate and forward content using the overlay. Figure 1(c) shows an example of 7 peers forming a P2P overlay for delivery of content.

5.3.1. Advantages and Disadvantages. The motivation behind incorporating P2P for content delivery over the Internet comes from its ability to address many issues associated with previous delivery architectures and its desired characteristics:

- ***Ease of deployment:*** Since P2P does not need any special support from the network (*e.g.*, maintaining group state and content forwarding) or any modification to the current Internet infrastructure, it can be easily deployed and adapted to a specific application.
- **Scalability:** P2P maintains the stateless nature of the network by requiring peers to keep states and perform additional complex tasks which results in its high scalability.
- **Support for higher level functionalities:** P2P potentially simplifies higher level functionalities (*e.g.*, congestion control, asynchronous participating and supporting heterogeneity of peers) by leveraging unicast solutions.

There are two disadvantages of P2P compared to IP multicast *(i)* less efficient usage of network resources and *(ii)* longer end-to-end delay in delivery of content. An end-system in P2P, has little or no knowledge about the underlying network topology, thus resulting in performance penalty in terms of longer end-to-end latency and lower efficiency compared to IP multicast. It is a significantly complex task to construct a P2P overlay in which some overlay links do not traverse the same physical link. As a result some duplicate traffic on physical links is unavoidable. Further, delivery of the content to a peer involves traversing the content through other peers, which potentially leads to an increase in the latency.

5.3.2. Two Flavors of P2P. There are two flavors of P2P architectures based on the deployment level, *i.e.*, *Infrastructure-level* and *End-user level*.

Infrastructure-level is commonly referred to as Content Delivery Networks (CDN). In CDN, there exists a dedicated set of well-provisioned servers which are strategically placed around the Internet [67], [68], [69], [70]. According to the applications' requirement, these servers form an overlay with specific properties.End-users directly connect to one of the servers and receive the content through that connection. CDN was started by Sandpiper [68] and Mirror Image [69] and followed by Akamai [70] and Digital Island [71]. The overlay formed in this paradigm, is stable and reliable as it uses a set of dedicated and well-maintained servers. The high costs of server setup, bandwidth, and administration makes CDN

feasible only for large content providers who expect high demand. The cost is often not justifiable for average distributors with unpredictable demand or smaller budgets, such as schools, small companies, and individuals who want to deliver a particular content over the Internet.

In end-user-level which is commonly referred to as P2P, all participating end-users collaborate in content replication and forwarding [63], [72], [73], [64], [65], [74], [75]. P2P relies on all participating peers for delivery of content. Participating end-users (peers) form a P2P overlay and forward the content through the overlay links to reach to all peers.

P2P compared to CDN has some advantages, *i.e.*, higher scalability, robustness and lower cost. In P2P, available resources scale with the number of participating end-users. Essentially, the more peers join the session the more resources become available as P2P leverage the bandwidth resources of all participating peers. P2P is highly robust in the face of network failure or attacks because of the lack of a central point of failure. Moreover, P2P is low cost, as it relaxes the need for any kind of costly infrastructure or a set of over-provisioned, well-maintained and resilient server(s).

Design of a P2P content delivery application due to its unique features is challenging. Essentially the design of such an application involves the following challenges: *(i)* P2P relies on a wide range of autonomous peers that may easily fail or willingly leave at any time. Maintaining the connectivity of such an overlay is challenging, *(ii)* Accommodating heterogeneity and asymmetry of peers bandwidth is difficult, and *(iii)* Effectively utilizing the resources (*i.e.*, bandwidth) of individual peers to ensure scalability, without requiring a central (set of) server(s) is challenging. We primarily focus on the design of P2P content delivery applications due to the above challenges and characteristics, however, most of the discussions are also valid in the context of CDN.

5.3.3. Membership Management (Peer Discovery). To construct and maintain an overlay in a highly dynamic P2P network, there should be a mechanism to allow peers to locate other participating peers and keep track of their presence in the session. In fact, overlay construction greatly depends on the membership management and peer discovery method *i.e.*, which peers are currently in the session, and what are their characteristics such as geographical location, available bandwidth, network delay and uptime. This membership information might be maintained in a central server or it might be stored across all participating peers. Different methods for membership management have been proposed and incorporated by existing P2P applications. The common design goals of such methods include one or multiple of the following: scalability, simplicity and ease of deployment, load balancing, unbiased (random) peer discovery, efficiency, reliability, accommodating heterogeneity and QoS-awareness. The methods that we describe, are centralized, embedded, flooding, random walk, gossip-based and distributed hash table (DHT) based.

Centralized

The simplest method for membership management and peer discovery is to maintain a centralized directory of all participating peers in a dedicated entity called central server [76], [77], [78], [7]. The central server is typically located at a well known location (DNS name or IP address) and manages the membership information of all participating peers. The central server keeps the arrival and departure of all peers in the session and it may maintain some peer information such as available bandwidth and outgoing degree.

When a peer joins the streaming session, it notifies the central server of its arrival and membership information along with a peer discovery request. Upon receiving a peer discovery request, the central server selects the most suitable peer(s) from the list of members for the requesting peer, according to some application specific criteria (*i.e.*, closeness or uptime). Participating peers periodically send a heartbeat message to the central server and renew their membership state. Each peer upon departure, may send a message to the central server to clear its membership information. If the information of a peer has not been updated within a threshold, its membership

information will be deleted.

The centralized method is easy to implement and simple to deploy. Moreover, relying on the global information for peer discovery has the benefit that efficient overlays can be built. However, maintaining global membership information might not be scalable for live P2P streaming, due to the large number of participating peers and high churn. The centralized method also poses reliability issues as it is vulnerable to a single point of failure.

Embedded

In the embedded method, whenever a peer joins or need a parent, it probes existing peers in the overlay, starting from a random peer or the source, until it finds a suitable peer in the overlay [15], [79].The embedded method distributes the membership management load over all the participating peers and also eliminates the single point of failure. However, one drawback of this method is that when the number of peers is large, a joining peer may need to probe many peers before finding an appropriate peer (*i.e.*, locating its position in the overlay), which causes long delays in joining the session.

Flooding

Flooding is another method incorporated for peer discovery and membership management. In this method, each peer maintains a list of participating peers [2]. A new peer joining the session has to contact an existing peer; this peer forwards the discovery request message to all of its list except the one that sent the message. Each peer receiving the message will do the same until the maximum number of allowed forwarding hops has been reached. Peers receiving the discovery request message reply to the requesting peer, thus growing peer's list. In order to keep the list of participating peers updated and to maintain its size constant, these discovery messages have to be sent periodically by existing peers. The flooding method introduces high overhead which grows exponentially with the number of hops a message can be flooded.

Gossiping

Many recent studies [80], [81], [82], [83], [73] have adopted a gossip-based method for membership dissemination and peer discovery. A gossip-based or epidemic broadcast method operates as follows [84], [85], [86], [87], [88]: When a peer wants to broadcast a message, it selects t peers from its neighbor set (list of known participating peers) at random and sends the message to them; upon receiving a message for the first time, each peer repeats this procedure. The key property of gossip-based methods is that the neighbor set of each peer or the gossip targets represent a *uniform random* sample of all peers in the session. To achieve this uniform random selection, ideally, each peer should be able to select random gossip targets from the entire participating peers (*i.e.*, its neighbor set should include all the participating peers). Unfortunately, this is not a scalable solution, not only due to the high memory costs associated with maintaining full membership information about all participating peers, but also due to the cost of ensuring that such information is up-to-date. To overcome this scalability issue, several existing methods rely on a *partial random view* instead of a *full view*, which still allows a gossip-based method to work without losing its properties [88]. In a partial view, the neighbor set of peers consists of a small subset of participating peers from which peers can select their targets to relay gossip messages.

Narada [73] incorporates a full view gossip-based method for membership management. Every peer maintains a membership list of all other participating peers. This list should be updated when a new peer joins or an existing peers leaves the session. In order to propagate changes in the session's membership, each participating peer periodically generates a refresh message and exchanges its membership list with its neighbors. Participating peers upon receiving a refresh message update their membership list. Finally, if the state of some peers has not been refreshed within a certain threshold, their entry is removed from lists. Clearly, this full view membership

management has limited scalability.

DoNet [80], Pulse [83], mTreebone [89] employ an existing gossip-base method called SCAlable Membership Protocol, SCAMP [90], [91], to distribute membership messages by relying on the partial view of the session.Each peer in SCAMP maintains two separate lists, a *neighbor set* consists of a subset of participating peers that it selects targets for gossip messages, and an *InView* consists of peers from which it receives gossip messages (*i.e.*, peers that contain this peer in their neighbor set). When a new peer joins, it sends a subscription request to an arbitrary participating peer. When a peer receives a new subscription request, it forwards the new peer information to all members of its neighbor set. When a peer receives a forwarded subscription, it adds the new peer information in its neighbor set with some probability. If it doesn't keep the new subscriber, it forwards the subscription to one of its neighbors at random. If a peer decides to add the subscription to its neighbor set, it sends a message to the new subscriber telling it to update its InView. To accommodate churn, peers periodically send refresh messages to all off their neighbors. If a peer does not receive a refresh message for a long time, it assumes that it has become isolated, and it will resubscribe through a random neighbor. To leave the session, a peer will send to some of the peers in its InView a replace request containing a random peer from its neighbor set. The peer that receives this request, will replace in its neighbor set, the departed peer with the new one. To the remaining peers in its InView, the departed peer will send a departure notification to remove itself from their neighbor set.

Delivery of redundant messages, assured by the nature of the gossiping, provides reliability in case of failures or high loss rate in the network. Moreover, gossip-based method is highly scalable as it distributes the load among all peers and it has been shown that the load on each peer increases logarithmically with the peer population [92], [88], [90]. However, gossip-based method is not efficient in propagating membership information, as the inherent redundancy of gossiping produces duplicate network traffic.Another limitation of gossip-based method is its lack of support for heterogeneity of peer's bandwidth. In gossiping, peers' load and thus the size of neighbor set is uniformly distributed across all participants, however, in a heterogeneous environment, peers with higher bandwidth should have proportionally larger neighbor set. In addition, gossiping may not be suitable for QoS aware peer discovery. This is because a randomly selected peer in a large network is unlikely to have the desired QoS characteristics (*e.g.*, delay or bandwidth). For the purpose of QoS aware peer discovery, it is desirable if peers that are potential good parents are selected with higher probabilities.

Random Walk

The other method used for membership management and peer discovery is random walk which gives a random view of the participating peers in the session. Chunkyspread [93] employs a biased-random walk method called SwapLinks [94]. SwapLinks uses random walks to build unstructured graphs, and to do random peer discovery over those graphs. The random-walk method in SwapLinks has control over both the load placed on each peer, and the number of times each peer is discovered/selected. SwapLinks tries to accommodate heterogeneity of peers' bandwidth by statistically controlling the size of the neighbor set of each peer *i.e.*, peers with larger bandwidth proportionally have larger neighbor set. The idea is that peers with larger bandwidth should have more neighbors to select parents from, while peers with smaller bandwidth should have proportionally fewer neighbors.

DHT-based

Another distributed method for peer discovery and membership management is based on DHT [95], [96], a common P2P searching technique, usually for locating peers that store a desired object (*e.g.*, file) in P2P networks [97], [98]. In DHT, each peer is assigned an ID by hashing its own IP address, and each object is also associated with a key from the ID space. The peer

with an ID equal to the hashed key is responsible for storing the object's location. A query for the object is routed through several peers in the DHT to the peer responsible for storing the object. Therefore, there is a routing table maintained at each DHT peer based on which peers route queries.

To maintain membership information and facilitate peer discovery some of the P2P streaming protocols employed the DHT-based method [99], [95]. For instance, RITA uses landmark vectors of peers as objects to store information of peers in the DHT, such that information of peers that are physically close to each other are stored near each other in the DHT. Therefore, a peer finds information of close-by peers in an efficient and scalable way.

Studies [97], [98] show that query messages are routed through only $O(log(N)$ peers for each lookup, and each peer only needs to maintain $O(log(N)$ states in its routing table. The routing load is evenly distributed across all peers in the DHT.

Other methods

Previous distributed methods provide a random view of the session to each peer, however, QoS-aware peer discovery is desirable for some of the P2P streaming applications. To achieve that, Dagstream [100] employs a separate service for membership management and peer discovery called RandPeer [101]. RandPeer is essentially a centralized service for membership management that allows join and leave for peers and enables look up of random peers or look up based on some given QoS metrics. Peers register their information with the RandPeer service and send periodic refresh messages to RandPeer. RandPeer, clusters participating peers based on their QoS characteristics such as geographical location. To discover another peer, each peer sends a look up request to RandPeer, which will then return a random peer from a specific cluster. RandPeer uses a distributed trie data structure to organize the membership information, which is in turn mapped to an underlying distributed hash table (DHT).

RanSub [102] is used in Bullet [103], to facilitate the distribution of membership information across all participating peers. RanSub assumes the presence of an underlying overlay tree for membership management and peer discovery. It periodically, distributes random subsets of participating peers throughout the tree using collect and distribute messages. Collect messages start at the leaves and propagate up the tree along the path to the root. Collect messages sent by a peer to its parent are both random and uniformly representative of all participant of the sub-tree rooted by that peer. Distribute messages start at the root and travel down the tree to distribute uniformly random subsets of participating peers to all peers. The contents of the distribute set are constructed using the collect set gathered during the previous collect phase. With these messages, each peer periodically knows about a random subset of participating peers. One drawback of RanSub is that it is not efficient and has large overhead. Specifically, if a peer does not need to discover other peers, it must participate in membership management process and continue to learn about other peers. A method in which any peer, at any time of its own choosing, can discover some participating peers randomly or with specific QoS criteria without burdening other peers, has less overhead.

Discussion: Distributed methods of membership management such as embedded, DHT-based, flooding and gossip-based, are more scalable than centralized method in terms of the number of states that the server should maintain. However, the centralized method is the easiest to implement, can respond to requests quickly and can be configured for application specific requirements such as QOS-aware or heterogeneity-support. Table 3 summarizes some of the properties of the above methods of membership management and peer discovery.

Method	Server States	Peer States	Implementation	Scalability	QOS	Heterogeneity	Single point of failure
Centralized	$O(N)$	$O(1)$	Easiest	Low	✓	✓	✓
Embedded	$O(1)$	$O(Log(N))$	Difficult	High	✓	✓	X
Flooding	$O(1)$	$O(1)$	Medium	Medium	X	X	X
Gossip-based	$O(Log(N))$	$O(Log(N))$	Medium	High	X	X	X
Random-Walk	$O(Log(N))$	$KO(Log(N))$	Medium	High	X	✓	X
DHT	$O(1)$	$O(Log(N))$	High	Medium	✓	✓	X

Table 3. Characteristics of different methods of membership management. N is the peer population and K is the proportional degree of peers in Swaplinks.

6. P2P One-to-one Applications

P2P one-to-one file sharing applications are extremely popular. They are used by millions of users and present a large fraction of Internet traffic [104]. The functionality of these applications is similar to the client-server's. In client-server, the server is a well-known entity that the client directly contacts. On the other hand, in one-to-one P2P applications, the other party, locations of the file or resources is unknown. Therefore, P2P overlays provide mechanisms to discover stored file, resources or the other party. The main components of P2P one-to-one applications are *(i)* constructing an overlay and *(ii)* search/query delivery. P2P one-to-one applications based on the shape of the overlay and search mechanism can be classified into *structured* and *unstructured*.

Structured approach [2] impose constraints both on the shape of the overlay and placement of object (*e.g.*, file, resources) to enable efficient search. On the other hand, in unstructured approach such as Gnutella [2] and KaZaa [3], the shape of the overlay and placement of objects are arbitrary.

6.1. Unstructured

The most popular P2P one-to-one applications are unstructured [105]. In these applications, peers connect to each other in a random way, the object location is arbitrary and there is no guarantee for the success of a search query. Unstructured approach is completely decentralized. It requires only local maintenance and are robust to failures. Unstructured approach is good for building highly dynamic systems with the goal of anonymity and minimal centralized overlay. However, performing search is not efficient and produces lots of network traffic.

The main component of unstructured P2P one-to-one applications is the delivery of search queries over a random P2P overlay. Search in unstructured approach, can be *blind* or *informed*. In a blind search, peers hold no information about the location of the object and try to propagate the search query to a sufficient number of peers to satisfy the request. On the other hand, in informed search, peers leverages some localized information of the object location to perform search. The information can range from forwarding hints to the exact object locations. In this section, we initially discuss blind search mechanisms and further elaborate on informed search mechanisms.

6.1.1. Blind Search. A blind search tries to send queries to a sufficient number of peers to satisfy a search request. In many P2P networks, objects are replicated at many peers. Therefore, a blind search consumes an excessive amount of bandwidth in both flooding the queries and receiving the responses.

Random Breadth-First Search (BFS): Flooding is one of the basic unstructured searching methods. Gnutella is the most famous unstructured P2P system that uses flooding to deal with queries. To locate an object, a peer sends a query to all its neighbors, which in turn propagate the query to their neighbors, and so forth until the query reaches all the peers within a certain hop distance of the query initiator. This method is robust but inefficient as it generates lots of redundant traffics.

Modified-BFS [106]: In this variation of BFS method, peers choose only a ratio of their neighbors to which the queries are forwarded. This method is more efficient in terms of the redundant queries forwarded, compared to BFS, but still a lot of redundant traffic is generated.

Iterative Deepening [107]: This method tries to address maximum hop count selection problem in BFS by using successive BFS with increasing hop counts. A peer starts a flood with a small maximum hop count and waits to see whether the search is successful or not. If successful, then the peer stops flooding, otherwise, the peer increases the maximum hop count and another flooding trial begins. This method can significantly reduce the message overhead.

***k*-Random Walk [107], [108]:** To reduce the number of duplicate messages walking the P2P network is proposed instead of flooding. In random walk, a query is forwarded to a randomly chosen neighbors. A query message walks through the network until it hits or reaches to a maximum number of hops. Random walk greatly reduces the number of redundant messages. One difficulty with walking is that it incur substantially more latency than flooding. [107] proposed using k walkers in parallel to decrease this latency. They showed that 16 to 64 walkers typically achieves good performance comparable to flooding. Comparing to BFS method, k random walk reduces the duplicate messages by more than one order of magnitude as shown by [107]. However, the success ratio is low in this method. The other issues associated with random walk discussed in [109], are *(i)* inefficiency of choosing a random neighbor at each step, and *(ii)* long delay for response by overloaded peers which results in an increase of the total latency in search. [109] proposes a dynamic topology adaptation protocol to solve the second problem by putting a peer's capacity into consideration when a query is to be forwarded.

Super Peers [110], [3], [111]: In this method, a super-peer network is organized by connecting high bandwidth peers or topologically close peers [111] to each other and each super-peer is connected to a set of ordinary peers. A super-peer operates as a centralized server to a set of ordinary peers. Each ordinary peer connects to one or several super-peers. For each query initiated from an ordinary peer, the ordinary peer sends the query to one of its super-peers and receives query results from it. Every super-peer maintains the contents of all its attached ordinary peers and then answers queries on behalf of them. Queries are flooded or walked among super peers to satisfy a search by an ordinary peer. As queries are forwarded among super peers that have relatively higher bandwidth, searches become much more efficient.

6.1.2. Informed Search. Informed search utilizes some information about object location to perform a search. The information can range from forwarding hints to the exact object locations. The placement of this information can be centralized or distributed. In a central placement [4], a central known server keeps and maintains all the exact object locations. The distributed placement can be pure or hybrid. In pure [106], [112], [113], all peers maintain some portion of the information, while in hybrid [114], certain peers are responsible for keeping the information of object locations.

Intelligent-BFS [106]: This method is an informed version of the modified-BFS. Peers store query-neighbor-ID for recently successful search queries. A peer upon receiving a query, identifies all queries similar to the current one, according to a query similarity metric; it then chooses to forward to a set of neighbors that have returned the most results for these queries. This method tries to increase the hit ratio rather than efficiency in terms of message reduction.

Directed Flooding and Biased Random Walk: In directed flooding, each peer sends queries to a subset of its neighbors that hits more search results than other neighbors. In biased random walk, rather than forwarding incoming queries to randomly chosen neighbors, each peer selects a neighbor that has the highest probability of having the query results. Compared with pure flooding and random walk, directed flooding and biased random walk decrease the number of forwarded queries and improve the hit ratio. To intelligently select one or some neighbors, a peer must maintain statistics on its neighbors. Statistical information of a peer can be collected based on its basic information (degree, capacity, availability,

latency, bandwidth, etc.) and the contents of it. [115] suggests that queries are forwarded to high-degree peers.

Adaptive Probabilistic Search (APS) [112]: Each peer performs k different walks for search in a probabilistic fashion rather than random. Each peer keeps a local index consists of one entry for each object it has requested per neighbor. This entry shows the relative probability that this neighbor chosen as the next hop in the future requests for the specific object. Index values are updated based on feedback from walks. If a walk succeeds/fails, the probability of the peers on the walk's path are increased/decreased. To achieve that, the feedbacks traverse the reverse path back to the initial requester either after a failed or successful walk. APS is bandwidth efficient, robust in presence of high dynamics.

Local Indices (LI) or r-hop Replication [116], [109]: The idea in LI is to reduce the number of peers that process a query. Each peer knows the objects stored at all peers within a certain hop count r and can answer queries on behalf of them. A search is performed in a BFS fashion. This method has high hit ratio, however, it is not efficient since it needs to maintain the indices at each peer, specially in a highly dynamic environment.

Keywords-based Query Route table [117]: This method tries to avoid flooding in Gnutella. Each peer maintains a query route table by hashing file keywords and regularly exchanging them with their neighbors. This method may reduce the bandwidth consumption of flooding.

Routing Indices (RI) [113]: Objects (*e.g.*, documents) fall into a various set of categories. Each peer maintains a routing table entry which includes an approximate number of objects from every category that can be retrieved from each neighbor. A peer forwards the query to neighbors with the highest "goodness" value. The goodness value of each neighbor can be derived from the number of successful queries of objects from every category.

Guide Rules and Interest-based Shortcuts [114], [118]: This method is based on the assumption that if a peer has a particular object that one is interested in, then it is more likely that it will have other objects that are also of interest [114]. Base on this assumption, peers form an interest-based overlay on top of the P2P overlay. Each peer starts the search by first looking at its neighbors in the interest-based overlay. This method can improve the performance of flooding-based search. However, when interest of users constantly changes, this method could decrease the search performance.

6.2. Structured

Due to the inefficiency and high bandwidth demands of search in unstructured approach, structured approach has been proposed [97], [119], [98], [74]. Overlay topology and mapping of objects (or its location) to peers are tightly controlled in structured approach. Such an approach implements a distributed hash table (DHT) as a substrate, in which object location is placed deterministically at the peers with ID corresponding to the object's unique key. Structured topologies can be tree [120], [98], ring [97], d-dimension torus [75] and butterfly [121], [122]for their DHT structure.

In structured approach, each peer maintains a small routing table consists of its neighboring peers' ID. Lookup queries for object keys are forwarded across overlay paths to peers in a progressive manner, with the peer ID that are close to the key in the ID space. This method of search results in an efficient, scalable and guaranteed search. However, this approach is complex, requires high maintenance overhead specially in highly dynamic networks, unable to support range and keyword queries [109]. Therefore, solutions based on structured approach are not deployed for scalable one-to-one P2P applications.

Various DHT-based solutions have proposed different key/peer mapping scheme, query forwarding policy and routing table formation and maintenance. We cover some of the most popular solutions and their extensions.

Tapestry [74] and Pastry [98], are similar to Plaxton and implement a tree-like overlay structure. Their goal is to improve the capability to detect, circumvent and recover from failures through maintaining

periodically updated cached content. Tapestry maps peers and keys into strings of numbers, and forwards lookups to another peer by correcting a single digit at a time. To support this forwarding, a peer needs to maintain, for each prefix of its own ID, of peers that match the prefix but differ in the next digit. Tapestry tries to build proximity-aware overlay with the cost of additional complexity.

Pastry [98] routes queries with key k to a peer with ID numerically closest to the key. Pastry uses a prefix-based forwarding scheme. Each peer p maintains a leaf set L, which is the set of $|L|/2$ peers closest to p and larger than p, and the set of $|L|/2$ peers closest to p and smaller than p. The correctness of this leaf set is the only thing required for correct forwarding; correct forwarding occurs unless $|L|/2$ peers with adjacent IDs fail simultaneously. The leaf set is conceptually similar to Chords successor list.

[123] presents a topology-aware version of Pastry. At each hop Pastry presents multiple equivalent choices to route a request. By choosing the closest (smallest network delay) peer at each hop, they try to minimize network delay. However, at each step the possibilities decrease exponentially, so delay is mainly determined by the last hop, usually the longest.

Chord [97] organizes peers into a ring shaped overlay called chordal ring. Object location is implemented by associating a key with each object and storing the key/object pair to the first peer whose identifier is equal to or follows the key in the identifier ring. Each peer maintains a routing table with information of about $O(logN)$ other peers. A peers routing table contains the IP address of a peer halfway around the ID space from it, a quarter-of-the-way, and so forth in powers of two. Peer forward a query for key k to the peer in their routing table with the highest ID less than k. There are some variants of Chord, such as Viceroy [124] and Multi-Ring [125].

CAN [75] uses a d-dimensional Cartesian coordinate space to implement the DHT abstraction. The coordinate space is divided into zones and each peer is responsible for a zone. Each peer is responsible for object's keys whose coordinates fell into the corresponding zone. Two peers are neighbors if their zones share a $d-1$ dimensional hyperplane. Peers maintain a routing table of all their neighbors. Queries are forwarded along an approximate straight path in the coordinate space. Upon receiving a query, a peer forwards it to the neighbor closest in the coordinate space to the peer storing the key.

[126] proposed using landmarks to bin peers into M groups. Each peer assigns itself to the closes landmark by measuring its round-trip time (RTT) to them. The authors then apply this binning technique to CAN, to construct a locality-aware overlay. In this binning-CAN scheme, the peer id space is partitioned into M equal-size portions, one portion corresponding to each group. Therefore, for a lookup, typically short topological hops are taken while the lookup message routes within a group; and then longer topological hops are taken when the message reaches the boundary of a group.

[127] proposed a hybrid approach in which queries are forwarded by flooding with a low maximum hop count. Queries for popular objects will be satisfied easily with the small flood. If not many results are found, then the lookup is done using a DHT which only stores rare objects.

7. P2P One(Many)-to-Many Applications

The goal of One(many)-to-many P2P applications is to deliver content to a group of participating peers which is similar to the goal of IP multicast applications. The main components of one(many)-to-many P2P applications is overlay construction and content delivery. Peers form an overlay with some specific properties (*i.e.*, shape and connectivity) and distribute the content through the connections in the overlay. In such applications, each peer receives the content from one or multiple other peers referred as *parents*, and might forward the content to other peers called *children*.

The main problem of one(many)-to-many P2P applications is how to deliver content to all interested peers. For this an overlay is constructed among peers which can be a single tree, multiple trees or a mesh. Further, content (data packets) are delivered to all peers on top of the overlay, which can be performed in push or pull fashion.

Several research studies and proposed solutions for P2P one-to-many content delivery exist that suggest different approaches for overlay construction and content delivery. Based on the characteristics of the overlay (*i.e.*, shape of the overlay) and the way the content is mapped to the overlay, existing solutions can be broadly classified into four approaches: *(i) single-tree-based*, *(i) multiple-tree-based*, *(ii) swarm-based*, and *(iv) hybrid*. [1]

A simple way of delivery of content over a P2P network is to organize peers into a single source-rooted tree and push the content on top of that; typical examples include ESM [73] and Yoid [64]. In this approach, each peer has only one parent and receives the whole content from that parent. The single-tree-based approach is vulnerable to churn, a leave or failure of a peer, particularly one close to the source may cause disruption in delivery of content to all the descendants of that peer. The tree thus has to be repaired frequently, which adds extra costs and overhead. Moreover, this approach has some other (generally known) drawbacks such as limited scalability due to inability to utilize all peers' bandwidth and inability to accommodate heterogeneity of peers' bandwidth.

Multiple-tree-based approach is an enhancement to the single-tree-based designed for delivery of streaming content. It builds multiple trees, each of which delivering a particular part of the streaming content. CoopNet [76] and Chunkyspread [93] are proposed protocols based on the multiple-tree-based approach. In multiple-tree-based approach, the streaming content is divided into sub-streams or descriptions and then each sub-stream is mapped to a particular tree. While this approach can overcome some limitations of single-tree-based approach, the multiple-tree overlay is more complex to be constructed and maintained.

Swarm-based approach constructs a mesh over all participating peers and incorporates swarming on top of that for delivery of the content [1], [7]. In this approach, each peer has multiple parents from whom it pushes the content and multiple children that serve the content to. The content is divided into equal sized blocks. Each peer explicitly informs its children of the block availability information and periodically requests blocks from its parents.

An alternative approach is a hybrid one which sits in the middle of the multiple-tree and swarm-based approaches. A hybrid approach is basically a mesh-tree or pull-push approach in which peers receive part of the content from their parent in a tree structure and pull the rest of the content from their mesh parent(s). Bullet [103] and mTreebone [89] are two protocols that incorporate such an approach.

7.1. Single-tree-based Approach

The single-tree-based approach is by far the most popular, obvious and intuitive way to support P2P one(many)-to-many content delivery applications [128]. This approach is commonly referred to as *application-level multicast*. It aims to construct a source-rooted tree across all peers. Content is delivered over the tree, with each packet being disseminated using the same overlay. In the context of many-to-many applications, a shared tree might be constructed or a separate tree for each peer as a source.

Content delivery in single-tree-based approach is push-based, that is, when a non-leaf peer receives a packet, it forwards copies of the packet to its children. In the single tree-based approach, all packets follow the same tree overlay, therefore, it is critical to ensure that the overlay is optimized to offer good performance to all peers.

In protocols adapted single-tree-based approach, to join the session, a new peer initially contacts a central server or bootstrap node to find at least one existing peer. This existing peer serves either as a parent for the new peer or an initial contact point for the new peer to run the protocol's tree construction algorithm to find its parent. The way the parent is selected (*i.e.*, the tree is constructed) differs from

1. Due to the significant similarities between approaches for one-to-many and many-to-many P2P content delivery, for the rest of this survey, we cover both simultaneously. However, some of these approaches are suitable for one-to-many applications only. In cases when some design choices are applicable for one of them we explicitly mention it.

one protocol to another. Further, to maintain the content delivery tree, such a protocol must be able to detect any failure and partitioning, as peers join and leave the session [129]. In particular, if a peer fails or leaves, the overlay should be repaired quickly, otherwise the delivery of the content gets disrupted to all lower level peers in the sub-tree rooted at that peer. Protocols adopted different methods for tree maintenance, some relies on a central server to maintain the tree, while others form a control topology and let peers communicate over that [130]. Moreover, as network condition may change over time, some protocols provide adaptation mechanisms to improve the quality of the content delivery tree.

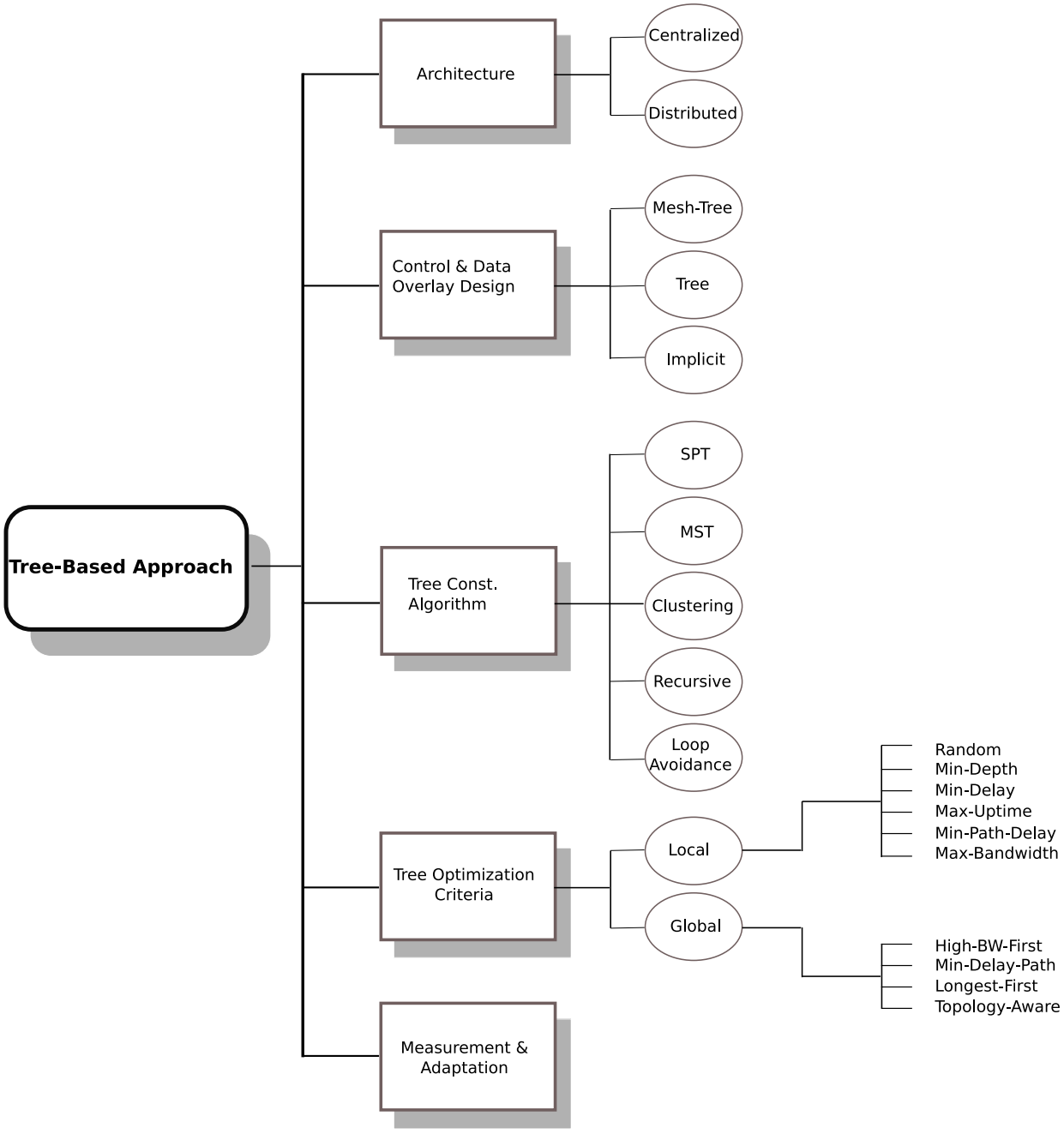
In general, the basic components of a single-tree-based P2P protocol as shown in Figure 2, consist of: *(i)* a centralized or distributed architecture for overlay construction and maintenance [130], [131], *(ii)* a tree-first, mesh-tree, or implicit design for control and data overlay, *(iii)* an algorithm for tree construction, *(iv)* performance criteria for tree optimization, and *(v)* an adaptation mechanism for coping with network dynamics. To provide a systematic insight into the design of different single-tree-based P2P protocols, we broadly categorize them according to the above components.

7.1.1. Architecture: Centralized, Distributed or Hybrid. Single-tree-based P2P protocols can differ in the architectural approaches adopted for the overlay construction and maintenance. The frequent joining and leaving of peers, failures, and the change of network conditions all make the content delivery tree in an extreme dynamic situation. Therefore, the approach adopted for the overlay construction and maintenance is a key issue in single-tree-based protocols. This determines the centralized or distributed architecture for construction and maintenance of the overlay tree. A centralized approach refers to performing all the functionalities regarding to tree construction, optimization and maintenance at a central server. The central server collects measurements from all peers and guides the participating peers to set up an efficient source-rooted tree. The main advantages of a centralized architecture are in its simplicity and quickness in computing the optimal tree, its reliability in preventing tree partitions and loops, as well as its efficiency in repairing the tree in case of peers departure or failure. However, the centralization limits the scalability and poses reliability issues as it is more vulnerable to a single point of failure.

In a distributed architecture, the load of construction and maintenance of the tree is distributed across all participating peers. Participating peers are responsible for finding their own or another peer's optimal position in the tree. Therefore, it is relatively more robust to failures as there is no single entity in the session, that its failure affects the entire group.Moreover, the distributed architecture is more scalable due to relaxing the need for a central server as a potential bottleneck. Although intuitively one might think that the distributed architecture for tree construction and maintenance better fits large-scale content delivery applications, there are still incentives for a centralized architecture [132], [77]. A fully distributed architecture causes excessive overheads and is not as optimal, efficient and quick in building optimal tree. Further, a synchronized communication among the participating peers and thus, fast decision-making process are hard to achieve with distributed architecture [77]. There is a trade-off between simplicity and practicality of a centralized architecture versus robustness and scalability of a distributed architecture.

7.1.2. Control & Data Overlay design: Mesh-tree, Tree or Implicit. In distributed architectures, to facilitate the tree construction and maintenance, participating peers can be organized into two overlays: control and data overlay. Peers on the control overlay may probe each other to measure some performance metric and periodically exchange refresh messages to identify and recover from "ungraceful" departures. The data or content delivery overlay is usually a subset of the control overlay and identifies the paths for delivery of content from source to all peers. The data overlay is a tree, while the control overlay might have a separate overlay mesh structure where peers have higher connectivity or it may share the same structure as the data overlay. There are three different designs for control and data overlay topology: tree-first, mesh-tree, and implicit.

Tree-first: In the tree-first scheme, participating peers are organized into a tree for content delivery



(a)

Figure 2. An overview of the tree-based approach components along with the design choices of each component.

by selecting their parents from known peers (in distributed or hybrid architectures) or by connecting to a peer assigned by the central server (in centralized architectures). Peers on the tree may establish and maintain additional control connections to other participating peers to enhance the robustness of the tree in dynamic environment. These additional connections along with the content delivery tree form a control overlay mesh. Examples of protocols using this scheme are Yoid [64] and HM [72]. This scheme

is simple and scalable due to the low communication overhead. It also gives direct control over the content delivery tree. However, this scheme requires running additional algorithms for loop avoidance and detection along with the partition detection to ensure that the overlay is indeed a connected tree [64].

Mesh-tree: In the mesh-tree scheme, the first step is constructing a mesh for control traffic and then a tree for data traffic. It is common for participating peers to first organize themselves into a mesh as the control overlay. The mesh is usually optimized for some performance criteria and dynamically adjusted to accommodate the underlying network changes. For instance, if a peer arrival or departure occurs, or some mesh connections experience congestion, the mesh may be reconstructed to adapt to the changes. The second step is the construction of a source-rooted tree for content delivery on top of the mesh. Participating peers run a well-known routing algorithms (*e.g.*, DVMRP [133]) on the control overlay compute the data tree in a distributed way. In the mesh-tree scheme, the quality of the tree depends on the quality of the mesh. Therefore, the challenge is to construct an efficient mesh, that is, each link in the mesh, has a good property defined by the protocol's specific optimization criteria (*e.g.*, . minimum delay or high bandwidth), while maintaining the mesh with low overhead (*i.e.*, each peer has a small number of neighbors in the mesh). One of the most popular protocols uses this scheme is Narada [73]. Protocols employ mesh-tree scheme are efficient for small peer population, but do not scale well beyond a few tens of peers [134]. The Mesh-tree is more complex than the tree scheme. However, it provides more resilience to failure of participating peers and simplifies tree construction and maintenance as loop avoidance and detection are built-in mechanisms in well-known tree construction algorithms such as DVMRP.

Implicit: Implicit performing control and content delivery overlay at the same time is also possible and is called implicit scheme. The implicit scheme creates only a control overlay among the participating peers. The data tree is implicitly determined by the defined set of packet-routing rules which take advantage of the specific properties of the control overlay to build loop-free paths. Participating peers are usually arranged in clusters and the optimization works towards putting the peers with a relatively close performance criteria (*e.g.*, topologically close peers or latency-wise close peers) in the same cluster, while respecting upper and lower bounds of the cluster size. NICE [79] and Zigzag [15] are popular representatives employ this scheme. The advantage of this scheme is its flexibility and scalability as each peer need to maintain information about only a small subset of participating peers. However, the resulting tree might not be optimal.

7.1.3. Tree construction algorithm. A tree construction algorithm typically involves a solution to a graph theory problem. That is, given a certain graph $G = (V, E)$ (*i.e.*, V represents all peers and E characterizes the overlay connections), and certain constraint on each peer (*i.e.*, peer degree), the problem involves the construction of a source-rooted tree covering all peers that tries to optimize a performance criteria; *e.g.*, minimizing the delay/hop-count from source to each peer, minimizing the total number of hops, or the cumulative end-to-end delay of forwarding content to all peers. Common algorithms for tree construction are as follows:

Shortest Path: The aim of this algorithm is to construct degree constraint minimum diameter spanning tree. A Shortest Path Tree (SPT) algorithm such as Dijkstra constructs a tree in which the path from source to each peer has the minimum cost, where cost is the protocol specific optimization criteria such as hop count or delay. An SPT or one of its variants is commonly used by various proposed tree-based protocols such as TAG [135].

Minimum Spanning Tree: This algorithm tries to build a low cost tree without considering degree constraint of peers. In a graph with a cost associated with each edge, a minimum spanning tree (MST) is a tree with minimum sum of total cost connecting all peers. MST is commonly used by centralized approaches such as ALMI [132] and HBM [136] to construct a low cost tree that is not rooted at any particular peer and all peers use the same tree to distribute their content. Cost can be captured by RTT measurement between two connected peers or hop count measurement depending on the protocol's

specific optimization criteria. MST usually contains peers with high degree which will be overloaded. Finding a degree-constraint minimum spanning tree is NP-complete [137]. Therefore, in practice many single-tree-based protocols do not seek to find the minimum cost spanning tree, rather turn to construct a near-optimal spanning tree that has bounded peer degree.

Clustering: Clustering algorithm divides peers into different clusters based on some desired criteria (*e.g.*, topologically close). In each cluster, a peer as a head is responsible for receiving the content from outside of its cluster and sending that to all the members of the corresponding cluster. Some protocols such as ZigZag [15] and NICE [79] deployed such a clustering algorithm for tree construction. Clustering algorithm has low computation overhead, however, the resulting tree might not be optimal and some peers might have very high degree.

Recursive: Recursive algorithm mostly used in distributed protocols, where each peer upon joining or rejoining recursively traverses the existing tree from source to find its appropriate parent and position in the tree. To find a suitable parent, peers can traverse the tree and repeat the top-down process in various ways such as breadth-first-search, depth-first-search or by some redirection hints from another peer (*e.g.*, A peer first contacts the source, chooses the best peer among the source children, and repeats this top-down process until it finds an appropriate parent). The position of each peer is decided based on desired criteria of the protocol. For instance, when the criteria of tree construction is building a minimum depth tree, each peer traverses the tree from source in breadth-first-search fashion until it finds an existing peer with empty slot to accept it as a new child. SpreadIt [138] and Overcast [139] use recursive algorithm to build their desired tree.

Loop avoidance: Some existing solutions adapt loop avoidance algorithm to construct a tree, in which each peer with some methods detects whether choosing a specific parent creates a loop or not. One method for detecting loops is through the *root path* [64]. The root path is the set of peers in the path from source to each peer which is analogous to AS path in BGP. If a peer finds that the root path from its potential parent contains itself, it should choose another parent otherwise a loop will be formed in the overlay. This method in incorporated in Yoid [64].

7.1.4. Tree optimization criteria. In the single-tree-based P2P applications, depending on the design goal(s) of the proposed protocol, constructed tree can be optimized for various local or global criteria. Such criteria directly affect the resulting shape of the tree. For instance, a protocol that aims in constructing a tree in which high bandwidth peers are located at the top levels, results in a tree with low depth. Local criteria are the ones that are optimized for each peer independently,whereas, global optimization criteria of tree affects the overall shape of the tree and its characteristics. Global optimization might require some tree reconstructions and preemptions. Single-tree-based protocols might be optimized for one or multiple of these criteria. Next, we explain these local and global optimization criteria.

- **Local optimization criteria**
  - *Non/random:* There is no specific optimization criteria, and tree is constructed at random. This criteria leads to the most efficient with the lowest overhead tree construction. However, the resulting tree might have a large depth [140].
  - *Max-Bandwidth:* Each peer tries to find a parent with maximum available bandwidth to ensure good quality and fast delivery of the content. Overcast [139] employs this optimization criteria with the goal of maximizing the bandwidth of the path from the source to all peers.
  - *Min-Delay:* Each peer finds a closest parent in terms of end-to-end delay. The intuition behind this criteria is to potentially minimize the delay from source in a light-weight fashion.
  - *Min-Path-Delay:* Peers try to find a potential parent with minimum delay from source through the overlay tree to achieve smaller buffer size and better liveliness. Clearly, this criteria has large overhead.

  - *Min-Depth:* Peers find a parent with minimum depth in the tree. This criteria potentially reduces the likelihood of bottleneck among upstream connections and achieve better robustness. The reason is that a smaller number of ancestors reduces the probability of late arrival of the packets due to the congestion occurrence in any upstream connections and it also decreases the probability of disruption in delivery of content due to departure of any of the ancestors.
  - *Max-Uptime:* To decrease the likelihood of parent departure and disruption in delivery of the content, peers try to find a parent with maximum uptime or session time.
- **Global optimization criteria**
  - *High-Bandwidth-First:* Peers with higher outgoing access link bandwidth are placed in the top levels of the tree [141]. In such a tree, peers are organized from high to low levels in a non-increasing order of their outgoing access link bandwidth; that is, peers do not have more bandwidth than any peer higher up in the tree. This criteria allows later arriving peers to preempt the positions of existing peers with smaller bandwidth. It can achieve a minimum depth tree at any time but needs frequent preemptions and reconnections between peers to maintain such a globally ordered hierarchies. The departure of a peer may impose very high overhead on the descendants to reconstruct the tree with such a criteria.
  - *Minimum-Delay-Path:* This criteria tends to minimize the delay of each path from source to any peer. The shortest path tree construction algorithm can globally achieve that by computing a shortest path tree over some existing edges.
  - *Longest-First:* This criteria builds resilient trees by minimizing the number of interruptions in delivery of content resulted from departures of peers. It puts the longest-lived peers in higher level of the tree; the intuition behind this is that when the peers' lifetime follows a heavy-tailed distribution, the older peers generally tend to stay longer than newer peers [140].
  - *Topology-aware:* Tree is constructed and maintained in a way that peers that are closer to each other in terms of underlying network topology connect to each other. This criteria can potentially reduce the delay between peers and decrease duplicate network resource usage [135].

7.1.5. Overlay-based Measurement and Adaptation. A tree-based P2P application must continually adapt itself to the variation of network environment and to a new set of participating peers due to the joining of new peers or departure of existing peers. Internet is a highly dynamic environment and prone to unpredictable congestion, partitions and flash crowds. Thus, an efficient tree at some point may become very inefficient after a period of time. Moreover, the arrival or departure of peers might result in a completely different efficient and optimal tree. Therefore, protocols should be able to discover network variations by means of frequent measurements, adjust to a new set of participating peers and adaptively alter the tree to reflect any changes.

Many measurement techniques to capture overlay connections' characteristics have been proposed [142], [143], [144], [145], [126]; they usually send light-weight messages to estimate the desired criteria such as end-to-end delay or available bandwidth between participating peers. In these techniques, the delay can be derived by RTT (round-trip time) and a 10-KB TCP probing are used for available bandwidth estimation. Unfortunately, available bandwidth is highly fluctuating as it depends on both bottleneck link bandwidth and cross traffic which is highly variable in the Internet. Therefore, frequent verifying the validity of a measured available bandwidth causes high overhead. Some techniques [146], [147] try to estimate the bottleneck bandwidth which is stable and accurate. Bottleneck bandwidth is the upper bound for available bandwidth. While bandwidth measurement is an active area of research [148], [149], accurate results generally require the transfer of a large amount of data to gain confidence in the results and due to its high variability over short time periods it's frequent measurements leads to a high overhead.

After detecting any changes, tree-based protocols adapt the tree overlay to reflect those variations and improve optimality of the tree. Many tree-based protocols incorporate adaptation to the tree in a local way

Protocol	Architecture	Control & Data Overlay Design	Tree Construction Algorithm	Optimization Criteria	Adaptation	Degree Constraint	Membership Management	Goal
ALMI [132]	C	NA	MST	Global Minimum-Total-Delay	✓	✓	Centralized	Minimizes total delay to all peers
AMCAST [150]	C	NA	MST	Global Minimum-Total-Delay	X	X	Centralized	Minimizes total delay to all peers
BTP [150]	D	Tree-first	Recursive	Local Min-Delay	✓	X	Embedded	Minimizes delay of delivery path to each peer
HBM [136]	C	NA	MST	Global Minimum-Total-Delay	✓	✓	Centralized	Minimizes total delay to all peers
Narada [73]	D	Mesh-first	SPT	Global Minimum-Delay-Path	✓	X	Gossip	Minimizes delay of delivery path to each peer
NICE [79]	D	Implicit	Clustering	Local Min-Delay	✓	X	Embedded	- Maximizes scalability by lowering overhead on peers - Minimizes delay between peers
OMNI [151]	D	Tree-first	SPT	Global Minimum-Delay-Path	✓	✓	Embedded	Minimizes delay of delivery path to each peer
Overcast [139]	D	Tree-first	Recursive	Local Max-BW	✓	✓	Embedded	Maximizes bandwidth from source to peers
ProBaSS [152]	D	Tree-first	Recursive	Local Min-Delay	X	✓	Embedded	Minimizes delay between peers
RITA [99]	D	Tree-first	Loop Avoidance	Local Min-Delay	✓	✓	DHT-based	- Minimizes delay between peers - Efficient & scalable adaptation mechanism
Scattercast [61]	D	Mesh-first	SPT	Global Minimum-Delay-Path	✓	X	Centralized	Globally minimizes delay from source to peers
SpreadIt [138]	D	Tree-first	Recursive	Local Min-Delay	X	✓	Embedded	Minimizes delay between peers
TAG [135]	D	Tree-first	Recursive	Global Topology-aware	✓	X	Embedded	Minimizes stress & stretch
TBCP [153]	D	Tree-first	Recursive	Local Min-Delay	X	✓	Embedded	Minimizes delay between peers
Yoid [64]	D	Tree-first	Loop Avoidance	Local Min-Delay	✓	✓	Random-walk	- Minimizes delay between peers - Minimizes loss rate
ZigZag [15]	D	Implicit	Clustering	Local Min-Delay	✓	X	Embedded	- Maximizes scalability by lowering the overhead on each peer - Minimizes delay between peers

Table 4. Taxonomy of solutions for tree-based P2P applications.

by allowing peers to dynamically change to a better parent instead of globally change the tree structure. For example, Overcast [139] estimates the available bandwidth between a peer and its potential parent. In addition, each peer periodically reevaluates its position in the tree by measuring the bandwidth to its current siblings, parent and grandparent. Based on the new measurements peers may switch parent.

Note that, excessive adaptations make the tree overlay unstable which is specially an issue in the context of streaming applications, due to the potential disruption of streaming delivery to some of the participating peers.

Table 4 compares a number of tree-based protocols with respect to various characteristics described in this Subsection 7.1.

7.1.6. Case studies. In this section, we cover some of the protocols mentioned in Table 4, to get a better understanding of how they perform. We look at Narada [73], ALMI [132], OverCast [139] and NICE

[79] and ZigZag [15].

Narada [73]

Narada is one of the earliest tree-based P2P protocols for many-to-many streaming applications. It incorporates a distributed mesh-first scheme for tree construction, maintenance and a gossip-based protocol for membership management. Peers initially form a well-connected overlay mesh as a control infrastructure. Further, they run the DVMRP routing algorithm to construct a spanning source-rooted tree over the mesh for delivery of the content. Peers through a gossip-based membership management protocol keep an updated list of all the existing peers to detect overlay partitions and to select the best paths for delivery of the content. When a new peer joins the session, it retrieves a list of existing peers from a bootstrap node and sends a neighbor-request to them and further, establishes one or more mesh connections at random. Once a peer connects to the mesh, it starts exchanging periodic refresh messages with its neighbors. These refresh messages are propagated through the mesh to all the other participating peers.

The goal of Narada is to construct a shortest path tree for each peer as a source (source-specific) in which the delay from source to each peer is minimum. This tree is a spanning tree of the mesh. Therefore, the quality of the tree depends on the quality of mesh connections. Since in the join process, random connections are formed, initially the mesh is not efficient. To construct an efficient mesh and also adapt to the variation of network conditions, periodic adjustments are made to the mesh connections. Peers continuously probe other random participating peers, and may add or drop connections depending on the perceived gain in *latency* in doing so. This periodic probing also helps them detect a participating peer failure which further will be propagated throughout the mesh.

Overcast [139]

Overcast tries to construct a tree in which the bandwidth from source to each peer is maximized. Overcast places peers as far away from source as possible without sacrificing bandwidth. The tree is constructed with the recursive algorithm in a distributed fashion. Peers upon joining contact the source and choose the source as their parent. A series of rounds will begin in which they decide where on the tree their position is. In each round, peers measure their available bandwidth to their parent and to their parent's children by considering the time to download a 10KB data. If the bandwidth through any of the children is about as high as the bandwidth to their current parent, then that child becomes their new parent and a new round commences. In case, there are multiple suitable potential parents (bandwidth differences within 10%), the child with minimum number of hops reported by traceroute will be chosen as a new parent.

To adapt to network variation and optimize the tree, peers periodically reevaluate their position in the tree by measuring the available bandwidth between themselves and their parent, grandparent and all siblings. Based on the new measurements, peers may switch parent. To maintain the tree in a distributed way, peers keep an ancestor list to reconnect to the tree in case of a parent departure. When a peer detects that its parent is unreachable, it will connect to its grandparent or continue to move up its ancestry until finds a reachable peer.

ALMI [132]/HBM [136]

ALMI and HBM aim to build a Minimum-Total-Delay tree in a centralized tree-first fashion. In both ALMI and HBM, participating peers are organized into a tree, which is formed as a minimum spanning tree (MST) using delay as the metric. The only difference between ALMI and HBM is that ALMI builds the tree using only partial knowledge of inter-peer delay measurements, whereas, HBM requires the measured delay information between all peers (full knowledge).

A new peer will join the tree by contacting the bootstrap node. The bootstrap node then refers the

new peer to a randomly-selected participating peer. The bootstrap node ensures the efficiency of the tree by periodically calculating a minimum spanning tree based on the measurement updates received from all the participating peers. To collect measurements the bootstrap essentially instructs each peer to periodically send probes to a constant number of other peers and measure the round trip delay. This delay measurements serve as the costs used to periodically re-calculate a new minimum spanning tree.

NICE [79]

NICE focuses on implicitly building a Min-Delay tree in a distributed fashion while maintaining the overlay with low overhead. It organizes peers into multi-level hierarchy of clusters with bounded maximum and minimum size. While constructing the hierarchy, nearby peers (in terms of end-to-end delay) are mapped to the same cluster. Cluster-mates keep detailed state about each other. A peer that has the minimum maximum distance to all other peers in a cluster is the leader of that cluster. The source-rooted tree for content delivery is implicitly defined from the hierarchy. A leader of each cluster in each level is responsible for communicating with higher levels and delivery of the content to its cluster-mates. In each cluster, the control overlay is a clique, while the content delivery overlay is a star.

Upon joining, a new peer must locate the nearest cluster to itself. This is done in a recursive way, such that the new peer first contacts the highest level peer and then requests from it a list of peers in the next lower level. The new peer should probe each of the them and selects the closest one. Further, the closest peer will inform the new peer the list of its cluster-mates in the next-lower level. With this method, the new peer can iteratively find its level 0 cluster. To maintain the hierarchy and thus the tree, peers periodically send heartbeat messages to their cluster-mates. Leaders of each cluster should also keep state about their higher level cluster-mates. They are also responsible for maintaining proper cluster size and thus apply the splitting or the merging algorithm when needed. To adapt to network variations and changes due to peers arrival and departure in NICE, each peer in any level, periodically probes all cluster leaders of its level to identify the closest peer to itself in other clusters. If there is a closer cluster, it leaves the current cluster and switches to the new cluster. Leader selection also periodically should be recalculated and if necessary, a new leader will be elected.

By clustering the participating peers, the control overhead is effectively reduced, since a peer must maintain state for a limited number of its cluster-mates. This lets NICE scale better than a mesh-tree protocols (*i.e.*, Narada). However, in NICE, the some peers requires to have an unbounded bandwidth (*e.g.*, leaders of higher levels) and a very high overhead. Moreover, the large disruption time in case of cluster leader failure is another issue in NICE.

ZigZag [15]

ZigZag is very similar to NICE. It focuses on a large scale P2P streaming application with the goals of low delay from source to all peers, quick recovery from failures, and small overlay maintenance overhead. It tries to address some of the issues of NICE. ZigZag does so by splitting the role of the leader over two different entities, the head and the foreign-head. When a peer wants to join, it submits a request to the source, then this peer traverses along the tree downward from the source until finds the closest cluster to join by probing each. On each cluster, the foreign head is responsible for getting the content and transmitting it to all its non-head cluster-mates, whereas, the head is responsible for overlay maintenance. Thus, unlike NICE, the control overlay in ZIGZAG does not infer a content delivery tree.

Zigzag has been shown to have a better performance than NICE due to its method of constructing tree: *(i)* the worst case degree of NICE is $O(logN)$, while it is bounded by a constant in Zigzag and *(ii)* failure recovery in Zigzag is more efficient that that of NICE, *i.e.*, NICE requires $O(logN)$ peers to reconnect to the overlay while overhead in ZIGZAG is upper

Protocol	Max-Path length	Max-Degree	Avg.- Control Overhead	Peer Population
ALMI	$O(N)$	Proportional	$O(N)$	Small
BTP	$O(N)$	Unbounded	$O(N)$	Medium
HBM	$O(N)$	Proportional	$O(Max - Degree)$	Medium
TBCP	$O(N)$	Proportional	$O(Max - Degree)$	Medium
Narada	$O(N)$	Unbounded	$O(N)$	Medium
Nice	$O(log(N))$	$O(log(N))$	$O(K)$, Max:$O(Klog(N))$	Very large
ZigZag	$O(log(N))$	$O(K)$	$O(K)$, Max:$O(Klog(N))$	Very large
Overcast	$O(N)$	Proportional	$O(N)$	Large
Yoid/HMTP	$O(N)$	Proportional	$O(K)$	Medium

Table 5. Comparing characteristics of typical P2P content delivery applications. K is the cluster size for Nice and ZigZag and maximum degree for Yoid

bounded by a constant. Both NICE and Zigzag require that most of the peers have a guaranteed minimum outbound bandwidth,ı.e. an integer multiple of the stream bandwidth. Moreover, both of these complex protocols have to deal with cluster splits and merges, leader selections and peer departures. They are not optimized for a high rate of churn, joining the session requires $O(log(N))$ messages, where N is the number of participating peers, and disruptions in the tree due to peer failures can take up to 30 seconds to deal with. To maintain the structure of trees they require lots of control traffic (*i.e.*, periodic heartbeat messages between peers in each cluster). Moreover, the number of peers that partition from the tree and need to reconnect due to a failure is significantly large (*i.e.*, $O(logN)$ in Nice and a $O(k^2)$ in ZIGZAG).

7.1.7. Comparison and Discussion. The choice of mesh-tree, tree-first or implicit seems to influence the properties of a single-tree-based protocol the most. Therefore, we conclude this section by briefly summarizing the advantages and disadvantages of mesh-tree, tree-first and implicit designs, with respect to each other. Further, we focus on some of the major characteristics of existing tree-based protocols in general.

General Comparison of mesh-tree, tree-first and implicit protocols: The first advantage of the mesh-first protocols is their superior robustness. The probability of overlay partitioning due to peer failure or departure is low, since these overlays consist of multiple or redundant connections between peers. In addition, there is no need to reconstruct a path (as is the case in a tree-first designs) since alternate paths already exist. Finally, the mesh-tree protocols utilize the already existing tree construction algorithms which relaxes the need for any extra loop detection and avoidance methods and results in an optimal tree.

The main drawback of mesh-first protocols (*e.g.*, Narada) is high control overhead, since every peer needs to keep information on state of all the other participating peers. Every time a peer joins/leaves, all the participating peers have to be notified, leading to considerable control overhead. Although this is an explicit design choice, trading overhead for greater robustness, it limits the scalability of these protocols.

The main advantage of tree-first protocols is that they scale better to a larger population than mesh-first protocols, since each peer has to be aware of only a small number of other participating peers. Their main weakness is non-optimal tree and partitioning susceptibility. Tree overlays are more susceptible to partitioning, since departure of any (non-leaf) peer can lead to the tree partitioning and communication breakage.

On the other hand, implicit protocols (*e.g.*, NICE) has the highest scalability compared to previous designs due to lowering the control overhead on each peer. However, their main weakness is that they are not optimized for a high rate of churn, joining the session requires $O(log(N))$ messages, and disruptions in the tree due to peer failures can take up to 30 seconds to deal with [77].

Discussion on characteristics of single-tree-based protocols: The single-tree-based protocols reviewed in this section, vary widely in their goals, designs and performance evaluation metrics. To compare these protocols, we focus on some of the important evaluation metrics includes scalability, stability, delivered quality, delay of delivery of the content, conformity to degree constraint and efficiency of the protocol. To evaluate each of them we introduce some performance metrics as follows: average path length, average control overhead and out-degree.

- **Scalability:** The scalability of a protocol is limited by factors such as control overhead, amount of available and utilized resources and average depth of the overlay as it determines the average latency of packet arrival to peers. Table 5 summarizes the scalability and target size of peer population for various protocols under peer population column.
- **Stability:** In the single-tree-based protocols, when a parent leaves, it causes all of its descendants to become disconnected from the overlay and stop receiving the content. Disconnections are perceived as disruption in delivery of content, resulting in poor performance. Disconnected descendants will need to find new parents and reconnect to continue receiving the content. Average path length is an indicator of protocols' stability. In tall trees, departure of a high level peer affects the performance of a large number of descendants peers. on the other hand, short trees minimizes the damage caused by peer departure as shown in [140].
- **Quality/total delivery time:** In the context of streaming applications stream quality is important, whereas, in the context of file distribution applications, total delivery time is a performance metric. Delivered quality to each peer directly depends on the stream quality received and further sent by its ancestors along the path to the source. Total delivery time also depends on delivery time to the ancestor of a peer. In other words, both the delivered quality to each peer and total delivery time are affected by any loss or bandwidth fluctuation in the upper level of the tree. Therefore, probability of experiencing low quality or large total delivery time is proportional to the number of upstream connections or ancestors. Path length is a metric that captures the number of upstream connections and thus, can translate to the probability of experiencing low stream quality or large total delivery time.
- **Delay:** In the context of live P2P streaming, the delay for delivery of the streaming content from source to each peer should be minimized to receive the content as live as possible. The worst case delay from source to peers also determines the buffer size needed at each peer. Protocols that construct their tree with the goal of minimizing the delay from source to each peer can achieve the lowest delay. Average path length or hop count can *loosely* indicate the delay as it only shows the number of hops and not the latency experienced in each hop.
- **Conformity to degree constraint:** Whether protocols can apply the bandwidth-degree condition for streaming purpose or not is an important metric in the streaming context. Without a proper out-degree for each interior peer (proportional to its out-going access link bandwidth and an integer multiple of the stream bandwidth) delivered quality drops. In one hand, if the out-degree of a peer is larger than the $\frac{outbw}{STR\_BW}$, its child peers cannot receive the stream with the rate of STR_BW. On the other hand, if the out-degree of an interior peer is smaller than the mentioned ratio its out-going bandwidth can not be fully utilized and protocol can not scale very well. Maximum out-degree dictated by a protocol determines its conformity to the bandwidth-degree condition.
- **Efficiency:** The efficiency of the protocol can be defined as the amount of non-data traffic that it generates. Average control traffic is an indicator of protocols' efficiency.

To evaluate each of above metrics we introduce some performance metrics as discussed above: path length, out-degree, and average control overhead. Table 5 shows a summary of the performance of the various protocols in terms of these performance metrics. These metrics can be easily analyzed for the different protocols [134].

Path length is measured by the number of hop count through the content delivery tree and is an indicator for the stability of the tree and perceived quality at individual peers. From Table 5, it can be observed that the maximum path length can be very long ($O(N)$) for mesh-tree or tree-first protocols such as Narada and Yoid, where as for those protocols which use implicit scheme to construct the tree it increases only logarithmically with peer population.

In the context of streaming applications, the *maximum out-degree* of a peer in the tree overlay, shows whether these protocols (can) apply the bandwidth-degree condition for streaming purpose or not. Protocols (*e.g.*, Yoid, ALMI, HBM, HMTP) that allow peers choose their degree, can apply bandwidth-degree condition and choose a proportional degree. Protocols with unbounded degree such as Narada and BTP result in poor performance[2]. The Implicit protocol NICE can have a max-degree of $O(log(N))$ which clearly can not satisfy bandwidth-degree condition. ZigZag has an intrinsic bound for maximum out-degree per peer which is the size of cluster rather than proportional to peer's bandwidth. The maximum degree in Overcast, is related to peer's available bandwidths resulted in a proportional degree.

Average control overhead is the overhead introduced by the protocol in the form of control traffic to maintain the tree. In general, control traffic are all the non-data messages that are required to maintain, improve and repair the tree. It includes refresh messages exchanged by peers, probing traffic and other active measurements performed during the tree construction, maintenance and optimization process. The average overhead per node is often used as an indicator of the scalability and efficiency of the protocol. As shown in Table 5, the highly scalable protocols with implicit design (*e.g.*, NICE and ZigZag) have a non-linear bound on their control overhead and proportional to the size of the clusters. Mesh-tree protocols such as Narada have the highest overhead as each peer has to maintain the state for the entire participating peers. The same is for centralized protocols (*e.g.*, ALMI) in which each peer has to actively probe a partial set of other participating peers.Overhead in Overcast and BTP, is also unbounded as the degree is not bounded (although it is proportional in Overcast).

7.2. Multiple-tree-based approach

More recently, multiple-tree-based approach [77], [76], [95] for P2P streaming have been proposed to increase the overall resiliency and efficiency of the single-tree-based approach. In the single-tree-based approach, the intermediate peers deliver the content, while the leaf peers only receive the content. Therefore, the upload bandwidths of the leaf peers are not utilized for content delivery. Multiple-tree protocols such as CoopNet [77] and Splitstream [95] overcome this inefficiency by constructing multiple trees and distributing part of the content in each tree.Any peer could be an interior in one or multiple of the multicast trees, and contribute in delivery of the streaming content.

The stream is split into smaller sub-streams or descriptions (if encoded with MDC or LC) with equal bit rate. In such an approach, multiple source-rooted trees with desired properties are formed and each sub-stream is simply pushed to a particular tree by source. Each participating peer receives sub-streams from specific parent and serves multiple children. Each peer decides a proper number of trees to join based on its incoming access link bandwidth. The out-degree of each peer determines the number of children that it can support. Participating peers might contribute (*i.e.*, be *fertile* and have children) in all the trees that they have joined or in a subset of them. The policy adapted for allocation of out-degree or distribution of the number of children of each peer across various trees can affect the performance of such an approach.

Similar to the single-tree-based approach, the multiple-tree-based approach can be fully centralized [77] or it can have a distributed architecture [93]. Besides the architectural choice, other components

2. Although Narada defines a notion of maximum degree on the mesh, occasionally it is required to relax this constraint to allow new peers to join the mesh. Otherwise, in some cases, new peers will suffer a long latency before they can find a participating peer with spare slots.

of multiple-tree-based approach include optimization criteria and algorithm for construction of each tree, maintaining each tree and adaptation mechanism to improve trees in case of changes in network conditions. These components and their design choices are basically similar to those of the single-tree-based approach discussed earlier in Subsection 7.1. However, the new component in multiple-tree-based compared to single-tree-based approach, is the *out-degree allocation policy*. In the single-tree-based approach, depending on the out-degree of peers, they are either intermediates and have children or leaves without any children, whereas in the multiple-tree-based approach, peers can allocate their out-degree across various trees. In this section, we focus on this new challenge and component of the multiple-tree-based approach which is the out-degree allocation policy.

7.2.1. Out-degree Allocation Policy. In the multiple-tree-based approach, each participating peer can be fertile and have children in trees that it has joined. Based on each peer's out-degree ($outdeg$), the peer can be fertile in all of its joined tree or a sub-set of them. The new challenge in multiple-tree-based approach, is how to allocate *slots* within each peer's out-degree into various trees. Existing multiple-tree-based protocols propose various out-degree allocation policies as follows:

- *Random*: The motivation for random allocation policy is building diverse trees with low overhead. In random trees, while each peer may have upto $outdeg$ children, the distribution of the number of children of a peer across different trees is random without any constraint. Therefore, within the constraint imposed by $outdeg$, peers can accept children in any trees. The random policy is incorporated in Chunkyspread [93].
- *Interior-disjoint*: Each peer can be fertile in only one tree and in the rest of the trees is a leaf. CoopNet [77] applied this policy to allocate peers' contribution across various trees. This results in the trees with minimum depth which have desirable properties such as minimizing the number of affected peers at each peer departure and minimizing the propagated effect of bandwidth fluctuation from upstream connections to lower levels as described in Subsection 7.1. Moreover, interior-disjoint policy increases the stability of trees and simplifies tree maintenance in presence of churn, as the departure of a peer only partitions one tree.
- *Balance*: To have balance trees in which the number of peers that each tree can accommodate is equal, resources should be equally divided among trees; that is the sum of allocated slots by peers in each tree should be roughly equal. Therefore, the choice of which tree a certain peer should be fertile and how many slots it should allocate at depends on the number of slots at each tree. CoopNet tries to build balance trees.
- *One-child-per-Tree*: Peers have one child in each tree, which is analogous to minimum breadth trees [154]. These trees are stable and easy to manage but result in very long trees that suffer from long delay for content delivery and maximizing the probability of bottleneck propagation from upstream levels to lower levels.
- *Local-Balanced*: Every peer allocates equal number of slots into its fertile trees [154].

Protocols can apply more than one of the above policies to allocate peers' slots across various trees. For instance, CoopNet [77] tries to build balance and interior-disjoint trees by letting a peer to be fertile in only one tree which has the minimum number of fertile peers. They further, compare the performance of interior-disjoint balance allocation policy with random policy and showed that the interior-disjoint balance policy performs significantly better because it is able to construct shorter and also more diverse trees. However, building interior-disjoint trees is complex specially in heterogeneous environment in which peers have various outgoing access link bandwidth or correspondingly out-degree. Moreover, authors in [154], through analysis and simulations, discussed that interior-disjoint policy imposes some limitations on tree reconstruction in presence of churn. In fact, they showed that limiting the number of fertile trees for each peer although results in shorter trees, it increases the reconnection attempt failures for

disconnected peers through ancestor departure. Random allocation policy minimizes the probability of experiencing reconnection attempt failure for a disconnected peer.

7.2.2. Case studies.

CoopNet [77]

CoopNet focuses on organizing participating peers into multiple diverse trees to minimize the effect of churn and effectively utilize available resources in the streaming session. The goal of tree construction is to maintain multiple stable minimum depth trees with interior-disjoint and balance policy. CoopNet relies on existence of a central server for overlay construction and management. Streaming content is encoded with MDC and divided into descriptions. Each description is simply pushed through a separate tree. A newly joining peer first contacts the central server and informs its available out-going and in-coming bandwidth to determine its out-degree and the number of trees that it wants to join, respectively. The server first decides the tree in which it is going to be fertile; in all other trees the peer is a leaf node to achieve stability in presence of churn. The new peer is added as fertile peer to the tree that has the minimum number of fertile peers to keep the population of fertile peers balanced among different trees. To maintain minimum depth trees, the new fertile peer is placed as a child for the peer with the lowest depth that can accommodate a new child or has a child that is a leaf. In the latter case, the new peer replaces the leaf peer and the preempted leaf should rejoin the tree similar to a new leaf. When a fertile peer of a tree departs/fails, each one of its children as well as the sub-tree rooted at it are partitioned from the original tree, and should report the departed peer and contact the central server to rejoin the tree. Peers in such a partitioned sub-tree, initially wait for the root of the sub-tree to rejoin the tree. If the root is unable to join after a certain period of time, individual peers in a partitioned sub-tree independently contacts the server to rejoin the tree.

CoopNet, employs an adaption mechanism based on MDC, in which the central server periodically gathers participating peers reception information, and feed this information into the MDC optimizer to adapt to the network dynamics and group size. Thus, MDC continuously adapts to the incidence of packet loss in the network, with more redundancy added when packet loss is frequent and vice versa.

In CoopNet, a tree can always accept a new internal node. However, in presence of churn, a tree could become *saturated* and thus unable to accept any new leaf node. This occurs when a tree loses a fraction of its internal nodes within a short period of time which reduces the number of leaf nodes that it can accommodate. In this case, the number of internal nodes at different trees becomes imbalanced, where spare slots for leaf nodes are available on other trees but they can not be used to resolve the problem of the saturated tree.

Splitstream [95]

Splitstream is very similar to CoopNet and advocates the use of multiple-trees for live streaming applications. However, unlike CoopNet, Splitstream constructs the overlay in a distributed fashion, on top of the Scribe protocol [162] which is in turn based on Pastry [98], a DHT P2P substrate.Using Scribe relaxes the need for a resourceful central server. However, constructing interior-disjoint trees in a distributed fashion adds extra complexity and control overhead. For instance, in case a peer that has reached its out-degree limit receives a join request from a child, it should disconnect one of its existing children and accept the new one. The disconnected child then seeks to locate a new parent in multiple extra steps which might not be successful without violating the interior-disjoint policy. Therefore, in Splitstream, a peer might be assigned more children than it can handle, trying to avoid that either requires sacrificing interior-peer-disjointness or leads to a long disruption in receiving the streaming content for some peers.

Chunkyspread [93]

Chunkyspread is another multiple-tree-based protocol which tries to build multiple trees with random allocation policy in a fully distributed fashion.The stream is divided into various sub-streams with equal bitrate. Each sub-stream is pushed over a separate tree, while trees are random and not necessarily node-disjoint. To facilitate the construction of multiple-trees, Chunkyspread forms a well-connected random mesh by a continuously running distributed SwapLinks algorithm that described in Subsection 5.3.3. SwapLinks is able to control the degree of each peer, and Chunkyspread uses this to give peers with higher outgoing access link bandwidth proportionally higher peer degree to potentially have more children.

Trees are constructed through a loop avoidance mechanism as described in Subsection 7.1. Essentially, each peer tries to find a parent for each sub-stream without forming a loop. Chunkyspread avoids and detects loops by using bloom-filter in the data packets. Peers advertise the bloom filters they receive for every sub-stream to their neighbors. A given peer does not select a neighbor as a sub-stream parent if the peer itself appears in the neighbor's received bloom filter. Loop avoidance and detection add extra overhead and complexity for Chunkyspread.

Each peer has an accepted range for the number of children that it can serve and a target out-degree. Individual peers by considering the target out-degree, the current number of children and per-sub-stream bloom filters advertised by their neighbors, determine which neighbor would make appropriate parent for each sub-stream. To prevent from overloading a peer, each peer periodically checks to see if it has an overloaded parent, and an underloaded neighbor, and if so attempts to switch parents. Chunkyspread does not perform any adaptation to changes in network conditions such as fluctuation of available bandwidth or increase in loss rate and as long as parents' load is in their accepted range, children switch parents only to improve relative latency between sub-streams.

Upon failure or departure of a participating peer, only the immediate connected children try to find an appropriate parent for the corresponding sub-stream. During this recovery period, all the descendants of the departed peer are also disconnected from the corresponding tree.

CoolStreaming [163]

The new version of Coolstreaming, employs a multiple-tree-based approach for streaming of live content. Similar to Chunkyspread, the multiple-trees formed in Coolstreaming, are random and the streaming content is divided into sub-streams with equal bitrate. Membership management is through a gossiping mechanism. Peers initially form a random mesh by discovering each other through gossip messages. Each peer tries to find a parent for each sub-streams from the set of its neighbors in the mesh. Once a peer select a parent for a particular sub-stream, parents continue pushing the sub-stream to the peer.

Similar to Chunkyspread, trees are constructed through a loop avoidance mechanism but in a different way. Coolstreaming avoids loops by checking the latest timestamp available at neighbors for each sub-stream. If the largest timestamp available at a neighbor for a specific sub-stream is larger than the one available at the peer, the neighbor is closer to the source in the path of the corresponding sub-stream and can be a potential parent for that sub-stream. Among the potential parents for a particular sub-stream, each peer tries to select the one that is not considerably lagging behind other neighbors in terms of the largest available timestamp for any sub-stream.

In Coolstreaming, a parent will not voluntarily drop a child, therefore, it is upto the children to dynamically monitor the incoming bandwidth of their parent and trigger any parent switching if necessary. Parent switching is performed when a sub-stream pushed by a parent is lagging behind other sub-streams at the child peer or other sub-stream among neighbors. Such peer adaptation and switching can potentially cause stream disruption and the instability of the overlay topology.

Protocol	Architecture	Tree Construction Algorithm	Optimization Criteria	Allocation Policy	Adaptation	Degree Constraint	Membership Management	Goal
Chunkyspread [93]	D	Loop Avoidance	Local Random & Min-Path-Delay	Random	✓	✓	SwapLinks	- Improving relative delay of sub-stream delivery - Simple overlay construction & maintenance
CoolStreaming [163]	D	Loop Avoidance	Local Random	Random	✓	✓	Gossip	- Simple overlay construction & maintenance
CoopNet [77]	C	Recursive	Local Min-Depth	Interior-disjoint & Balance	✓	X	Centralized	Building stable short trees
DAGSter [78]	C	Recursive	Global Min-Depth	Random	✓	X	Centralized	Building stable trees
Splitstream [95]	D	Recursive	Local Min-Delay	Interior-disjoint & Balance	X	✓	Scribe	Building stable short trees

Table 6. Taxonomy of multiple-tree-based P2P streaming protocols.

7.2.3. Comparison and discussion. The above multiple-tree-based protocols mostly differ in the choice of allocation policy allocation, existence of an adaptation mechanism and distributed or centralized architecture. Table 6 presents a taxonomy of various multiple-tree-based protocols. To compare these protocols, we mainly focus on the effect of allocation policy on the performance of these protocols. The effects of the other components (*i.e.*, choice of architecture and adaptation) are similar to those of single-tree-based protocols that discussed earlier in Subsection 7.1.

Authors in [77], compare the performance of interior-disjoint balance allocation policy with random policy and showed that the interior-disjoint balance policy performs significantly better because it is able to construct shorter and also more diverse trees. As discussed before, short trees have desirable properties such as higher stability, reliability and shorter delay. However, building interior-disjoint trees is complex specially in heterogeneous environment in which peers have various outgoing access link bandwidth or correspondingly out-degree[3]. Another drawback of interior-disjoint policy shows up in environments with dynamic peer participation. Authors in [154], through analysis and simulations, discussed that interior-disjoint policy imposes some limitations on tree reconstruction in presence of churn. In fact, they showed that limiting the number of fertile trees for each peer although results in shorter trees, it increases the probability of reconnection attempt failures for peers disconnected from a tree through ancestor departure. Essentially a tree could become *saturated* and thus, unable to accept any new leaf node [164]. This occurs when a tree loses a fraction of its internal nodes within a short period of time which reduces the number of leaf nodes that it can accommodate and results in reconnection attempt failures. In this case, the number of internal nodes at different trees becomes imbalanced.

Random allocation policy as used in Coolstreaming and Chunkyspread, results in longer trees that are less stable and reliable. However, this policy minimizes the probability of experiencing reconnection attempt failure for a disconnected peer, as any peer with empty slot can accept children in any tree [154]. Maintaining and constructing random trees are also easier with less overhead compared to other policies.

7.3. Swarm-based Approach

An alternative approach for P2P one-to-many content delivery is swarm-based. In this approach, participating peers form a connected mesh over which they incorporate swarming (*i.e.*, pull-based) content delivery. In this approach, each peer receives content from multiple parent and serves content to multiple

3. Note than CoopNet argues that centralization simplifies the construction and maintenance of node-disjoint trees, while maintaining them in a distributed fashion as in Splitstream, adds extra complexity and overhead.

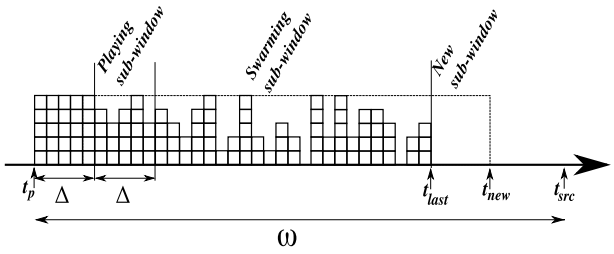


Figure 3. Buffer state at an scheduling event in a peer

children. In contrast to the tree-based, the swarm-based approach does not need to construct and maintain an explicit overlay structure for delivery of the content to all peers. Rather than constantly repairing a tree for delivery of the content in a hugely dynamic P2P environment, swarm-based approach uses the availability of content to guide the content flow. In such an approach, there is no pre-defined mapping of the content to connections, the content mapping to each parent is dynamic and locally decided at each peer.

Swarming content delivery enables participating peers to contribute their resources (*i.e.*, outgoing bandwidth) more effectively which in turn improves the utilization of available resources among peers, and leads to a better scaling property for the swarm-based approach. Swarming content delivery couples push content reporting with pull content requesting. Based on the availability information of the content at each parent, each peer as a child requests different blocks from parents. The blocks requested by each peer from a parent are determined by a *block scheduling* algorithm based on the available content and bandwidth from the parent. Block scheduling is the key component of the swarm-based approach, it aims to utilize available bandwidth from individual parents in order to maximize the received quality in the streaming context and minimize the total delivery time in the file distribution context.

There exists a few recent studies that have proposed a swarming-based P2P protocol which incorporate a variety of scheduling schemes, construct random or biased mesh and consider naive or intelligent source. Each of these protocols tries to achieve certain goals and maintain specific properties. To classify existing/possible swarm-based protocols for live P2P we first bring an overview of swarm-based P2P approach and further, discuss the main components of the swarm-based approach which are:*(i)* overlay mesh construction and maintenance, *(ii)* block scheduling scheme, and *(iii)* source behavior.

7.3.1. Overview. In the swarm-based approach, participating peers form a connected mesh over which they incorporate swarming (*i.e.*, pull-based) content delivery. In General, each peer discovers other participating peers, through a centralized bootstrapping or a distributed mechanism as described in Subsection 5.3.3. Further, each peer tries to establish connections to a sub-set of other participating peers. A peer can select random neighbors or the ones with specific criteria (*e.g.*, the ones that are topology-wise or delay-wise closer to the peer). Each peer has multiple parents that receives content from and multiple children that provides content to. Note that the pairwise connections in the swarm-based approach, can be used for content delivery in both bidirectional or unidirectional fashion.

Pull-based content delivery is the key component of the swarm-based approach. Content is divided into equal sized blocks. As a parent, each peer progressively reports its new or available blocks to all of its child peers. Based on this availability information at each parent, each peer as a child determines a sub-set of blocks that should be requested (*i.e.*, pulled) from each parent. The requested blocks from individual parents are determined by a *block scheduling* algorithm at each child. Each parent simply delivers requested blocks by individual children in the provided order and at the rate that is determined by the available bandwidth between parent and the corresponding child.

7.3.2. Mesh construction and maintenance. An overlay mesh with the desired properties of simplicity, robustness and efficiency can be constructed in various ways. Existing mesh construction methods, try to achieve all or some of the above properties. We now describe methods of mesh construction along with their negative and positive properties.

- *Random*: In a random mesh overlay, each peer randomly selects its neighbors from the pool of known peers. The random mesh overlay is simple to construct with low overhead. Moreover, it is highly robust to partitions. However, random mesh overlays generally lack locality awareness, trying to improve the locality awareness in random mesh lead to network partition, which can not be prevented or even detected [165], [100].
- *Biased*: Each peer chooses the peer which has specific criteria as neighbor among the ones it is aware of. The criteria for neighbor selection can be the one with minimum delay or topologically close. While this overlay might be network efficient, it generally does not guarantee connectivity [165].
- *Best-Random-Parents*: A peer might choose some neighbors at random and the rest biased. This overlay provides a near-efficient mesh with probabilistic connectivity properties [165].
- *Loop-free or Directed Acyclic Graph (DAG)*: A DAG enforces a partial order among the peers and forbids loops. It has the following property: In a directed acyclic overlay with one source, if each peer (except the source and its direct children) has at least k neighbors, then the removal of any $k-1$ non-source peers does not cause any remaining peers to be disconnected from the source.

Another dimension in the mesh construction, is the relationship between connected peers. Essentially, the relationship between connected peers can be *bidirectional* or *unidirectional*, that is a mesh can be *directed* or *undirected*. In a directed mesh, connected peers have a parent-child relationship and the flow of the content is from a parent to a child. On the other hand, in the undirected mesh, the relationship between connected peers is bi-directional; there is no parent-child relationship between each two connected peers. Mesh overlay in Bittorrent [1], is undirected, whereas, in the context of streaming applications, authors in PRIME [7], showed that directed overlays exhibit better performance than undirected overlays.

7.3.3. Block scheduling. The key component of swarming-based content delivery is a *block scheduling* scheme at individual peers which determines the sub-set of blocks that should be pulled from each parent. Block scheduling can be performed periodically or in an event-driven fashion. In the periodic scheduling, the block scheduling function is invoked once per Δ seconds and in each scheduling event it requests required packets from all parents. In the event-driven scheduling, the invocation of the scheduling function is triggered by an event when a parent delivers all of the requested blocks and becomes idle. In the occurrence of such an event, the scheduling function invokes and determines blocks that should be requested from any idle parents. Note that in contrast to the periodic scheduling, the event-driven scheduling does not send synchronized requests to all parents.

Design goals and flexibilities for block scheduling are different based on the content type. For elastic content, block scheduling should increase the diversity of blocks to minimize the download time. On the other hand, for streaming content, block scheduling should ensure in-time delivery of blocks along with increase in diversity of them. This motivates a different scheduling mechanism for elastic and streaming types of content.

P2P 1-to-many File Distribution Applications: Block Scheduling In the context of swarm-based elastic content delivery [1], [2], block scheduling leverages the elastic nature and the availability of the entire file at the source to distribute different blocks of a file among participating peers, enabling them to actively contribute their outgoing bandwidth through swarming.

In this context, block scheduling should ensure the diversity in a neighborhood, so that each peer always has a useful block to offer. More specifically, in a swarm-based P2P content delivery, once a peer receives a block, it can send it to its children who request for this block. A naive block scheduling

can result in low utilization of bandwidth, as most of the available blocks at a peer are not useful for their children (*i.e.*, they have already received those blocks). There are generally to block scheduling algorithms in the context of file distribution applications:

- *Random:* Each peer selects missing blocks that are available among its parents at random. This algorithm relatively ensures diversity of blocks in each neighborhood. However, this algorithm, can not prevent the last block problem as verified by [172]. Peers who receive most of the blocks must wait for a long time to obtain the last block, but the last block is only available in few of the peers.
- *Rarest-first:* Each peer selects blocks that are rarest among parents. This algorithm maximizes the minimum number of copies of any given block in each neighborhood which results in an improvement of the diversity and eliminating the last block problem. The rarest-first algorithm minimizes the total downloading time as each peer has always a suitable block to offer to its children.

P2P 1-to-many streaming Applications: Block Scheduling In the context of P2P streaming applications, blocks of streaming content are progressively generated over time and must be delivered in a timely fashion. To effectively incorporate swarming into P2P streaming and absorb any out-of-order delivery of blocks that are mainly caused by swarming, each participating peer requires to maintain ω seconds worth of buffering. This implies that peers' should delay their playout time Ω seconds behind source playout time (Figure 3). This delay continuously provides ω seconds worth of content that can be used by peers for swarming. Further, to ensure in-time delivery of blocks, each block should be delivered to all peers within ω seconds from its generation time.

Block scheduling should accommodate the following three requirements: *(i)*: the in-time delivery of each block to individual peers (*i.e.*, timing requirement), *(ii)* the diversity of available blocks among peers in order to enable effective swarming (*i.e.*, availability or diversity requirement), and *(iii)* the utilization of available bandwidth from parents to the peer in order to maximize the delivered quality. Moreover, block scheduling should cope with limited availability of content in the context of streaming, which could in turn limit the degree of diversity in available blocks among peers and adversely affect swarming content delivery. In essence, swarm-based P2P streaming should be "timing-aware" and properly leverage the demand between timing and diversity in the context of streaming.

The block scheduling function is invoked once per Δ seconds and in each scheduling event, it considers blocks within its current window of ω seconds (buffer) that should be pulled from parents in the current interval. The timestamp of the blocks in the current buffer falls within the following range [t_p+Δ, t_p+Δ+ω] where t_p is the peer's playout time.. Figure 3 depicts a view of blocks with relevant timestamps (buffer state) for a peer at an scheduling event. t_p, t_{src}, t_{last} and t_{new} denote peer's and source's playout times, the largest available timestamp, and the largest reported timestamp in this scheduling event. Block scheduling has two steps as follows: *(i) Block selection*, that determines the required blocks that are requested from parents. *(ii) Block assignment*, that allocates selected blocks to specific parents who can provide them.

Block Selection: Block selection algorithm should determine the missing but required blocks that still have sufficient time to be pulled from parents while considering the available blocks among parents. The block selection algorithm takes into account the aggregate incoming available bandwidth of the peer to determine the total number blocks that can be pulled during each interval or the block budget (*i.e.*, $\frac{AGG\_BW*\Delta}{PktSize}$). The design of a block selection algorithm has the following two dimensions: *(i)* the relative priority (*i.e.*, the order of requesting missing blocks) of various parts of the window, and *(ii)* the choice of *random* or *rarest-first* strategy to select a subset of blocks that are missing but available among parents. As proposed in [173], these two dimensions of design space for block selection motivate the following eight *candidate* selection algorithms:

- *Rare* or *Rand*: This algorithm selects all the blocks from the entire window using a rarest-first [80], [83] or random [1], [174] strategy, respectively. By enforcing random/rarest-first strategy across the entire window, this algorithm maximizes the diversity among delivered blocks to different peers.

Moreover, such an algorithm implicitly addresses in-time delivery (or timing) of blocks since the number of opportunities to request a block is equal to the number of scheduling events that the block has appeared within the window. Therefore, blocks that fall into the left side of the window in Figure3 (earlier timestamps) are more likely to have been requested in one of the previous scheduling events.

- *PRare* or *PRand*: This algorithm explicitly addresses the timing requirement by first requesting all the missing blocks that are in danger of being delayed beyond the deadline (*i.e.*, the blocks in the range of [t_p+Δ, t_p+$2*\Delta$]), and then using the remaining block budget to select rare/random blocks from the rest of the window [t_p+$2*\Delta$, t_{new}] [175][4].
- *NRare* or *NRand*: This algorithm explicitly addresses the diversity of blocks by first requesting all the blocks that are just entered the window (in the range of [t_{last}, t_{new}]), and then using the remaining budget to request a rare/random subset of blocks from the rest of the window [t_p+Δ,t_{last}].
- *NPRare* or *NPRand*: This is a hybrid scheme [7] that gives priority to blocks just entered the window, then the ones that are in danger of being delayed, and finally uses any remaining budget to request a rare/random subset of blocks from the the rest of the window. Therefore, this scheme explicitly addresses both timing and availability.

The output of the block selection algorithm is an ordered list of required blocks that are available among parents and should be requested.

Block Assignment: In this step, selected blocks should be mapped to requests from individual parents considering the blocks available at each parent. During each Δ seconds, each parent can send a limited number of blocks to the child that can be estimated by the available connection bandwidth from the parent to the child as $\frac{bw(i)*\Delta}{Blk_Size}$. The goal of block assignment is to utilize the incoming bandwidth of each peer during each interval to maximize the delivered quality. Therefore, it should fully utilize the available bandwidth of each parent by assigning appropriate number of blocks to each parent.

When a selected block is available only at one parent, the block is assigned to the associated parent. However, a selected block might be available in more than one parent. In this case, the block can be assigned based on two different ways:

- *Random:* Among the possible parents, assign the selected block to a random parent[174].
- *Min-Ratio:* Assign the selected block to a parent whom a smaller fraction of its block budget has been assigned so far [173], [80].

The second assignment policy tends to balance the number of assigned blocks to individual parents proportional to their block budgets.

7.3.4. Source behavior. In swarm-based approach, the uncoordinated requests from source's children can directly affect the performance. In the context of elastic content delivery, a disproportionate distribution of blocks among source's children might happen. Studies [177], [172] found that the distribution from the source becomes a bottleneck in the replication process. That is, a naive source could result in some of the blocks not being replicated at all while some others replicated multiple times. This phenomenon is referred to as starvation can significantly increase the total download time.

In the context of streaming content delivery, the maximum available quality is determined by *(i)* the aggregate available bandwidth from source to all of its children, and *(ii)* the rate of delivery for new blocks from source to its children which is limited to the stream bandwidth. For example, in the environment with limited source bandwidth (*i.e.*, roughly equal to the stream bandwidth), if the same block is requested by multiple children of source, the rate of delivery of new blocks might be significantly lower than the aggregate bandwidth from source and thus the stream bandwidth. In contrast, if all blocks are unique, the rate of new blocks is equal to the aggregate bandwidth from source. Therefore, if source's access

4. [176] designs a similar selection scheme in which it probabilistically selects a missing block close to deadline.

Protocol	Membership Management	Overlay	Heterogeneity Support	Block Selection	Block Assignment	Source Special Behavior	Degree constraint
AnySee [178]	Flooding	Directed Biased, Location-Aware	X	$Rare$	*Min-Ratio*	X	X
Bitos [176]	C	Undirected Random	X	$PRare$	$Random$	X	X
BitTorrent [1]	C	Undirected Random	X	$Rare$	*Random*	✓	X
Chainsaw [174]	Centralized	Undirected Random	X	$Random$	*Random*	✓	X
DAGStream [100]	RandPeer	DAG, Location-Aware	X	X	X	X	X
DoNet [80]	Gossip	Directed Random	X	$Rare$	*Min-Ratio*	X	X
GridMedia [179]	D	Random	✓	$Rare$	$Random$	X	X
PRIME [7]	Centralized	Directed Random	✓	$NPRand$	*Min-Ratio*	✓	✓
Pulse [83]	Centralized	Undirected Random	X	$Rare$	*Random*	X	X

Table 7. Taxonomy of swarm-based P2P streaming protocols.

link bandwidth is equal to (or larger than) the stream bandwidth, it can deliver the full quality stream to the session if its aggregate bandwidth is used properly. Due to the independent block scheduling by individual children of source, duplicate blocks may be requested from source which leads to the waste of source's bandwidth and decrease in the rate of delivery of new blocks to the overlay which in turn decreases the maximum available stream quality in the session.

Some of the existing protocols incorporate a source with special behavior in which source swaps any requested block with a block that has not been delivered at all or has the minimum number of delivered copies [172], [7].

7.3.5. Case Studies. Table 7.3.4 presents a classification of the major swarm-based P2P content delivery protocols based on different aspects of the swarm-based approach. In this section, we describe some of the major proposed protocols in more details.

PRIME [7]

PRIME organizes participating peers into a randomly connected uni-directional overlay mesh. Each peer tries to maintain certain number of parents and also serves a specific number of children proportional to its incoming and outgoing bandwidth, respectively. In PRIME, membership management and peer discovery are centralized. That is, upon arrival, a peer contacts a bootstrapping node to receive a set of peers that can potentially serve as parents. The bootstrapping node selects a random subset of peers in response to an incoming request for parents. All the parent-child connections in PRIME, are congestion controlled. To accommodate heterogeneity, the content is encoded with MDC which enables each peer to maximize its delivered quality by pulling a proper number of descriptions.

Individual peers periodically report their newly available blocks to their children and request specific blocks from individual parents. Each peer monitors the aggregate incoming bandwidth from all parents and slowly adapts the number of requested descriptions (or its target quality) with its aggregate incoming bandwidth from all parents. PRIME incorporates the $NPRand$ and $Min - Ratio$ for block selection and assignment, respectively. Essentially, the scheduling scheme is periodically invoked to determine a set of blocks that should be requested from each parent. The scheduler identifies the blocks with the highest timestamp that have become

available among parents since the last request. Further, the missing blocks that are close to begin played (in the first Δ seconds of the window) are selected, and the rest of the blocks are randomly selected. Next, the scheduler assigns blocks to the corresponding parent(s), while trying to fully utilize the parents' bandwidth. The block assignment in PRIME, tries to balance the load among parents and request blocks with more than one supplying parent from the parent with lowest fraction of its bandwidth budget utilized so far. Source has a special behavior in PRIME, that swaps a requested block with the one that has been delivered the least.

DoNet [80]

Early version of DoNet employs a swarm-based approach to support P2P streaming applications. DoNet requires each peer maintains a partial sub-set of other peers in the session, and participates in a continuously running gossiping algorithm (*e.g.*, SCAMP) for membership management. Newly joining peers contact a bootstrap node to obtain an initial set of potential partners and further, run the gossiping algorithm to discover new partners. DoNet constructs a bi-directional random mesh in which all peers have the same in- and out-degree regardless of their incoming or outgoing accesslink bandwidth.

The content delivery in DoNet is similar to PRIME, except that peers exchange the whole buffer map with their partners rather than newly available blocks. The scheduler is invoked periodically to determine the set of blocks that should be requested from each partner. DoNet employs a $Rare$ block selection scheme. The scheduler determines the potential suppliers of blocks starting from those with only one potential supplier, then those with two, and so forth. Among the multiple potential suppliers, the one that has the lowest fraction of its bandwidth utilized is determined. Source in DoNet, does not perform any specific coordination.

Chainsaw [174]

Chainsaw organizes peers into a bi-directional random mesh in a centralized fashion. A new peer contacts a bootstrap node and obtains a set of neighbors. A peer attempts to maintain a specified minimum number of neighbors without considering its outgoing or incoming bandwidth. Peers never refuse a connection request from any peer.

Block scheduling in Chainsaw, is event-driven that is the scheduling scheme invokes whenever a neighbor's outstanding list of blocks is finished or a parent notifies of availability of a new block. This event-driven scheduling can potentially reduce the playout delay introduced by periodic scheduling, as each peer can request a block as soon as it becomes available at any parent. However, notifying neighbors whenever a new block becomes available and further, per-block individual requests incur a very high overhead. Chainsaw applies a $random$ scheme to select blocks for requesting from neighbors, in case there exists more than one wanted blocks. The source has a specific behavior in Chainsaw, that the source maintains a list of blocks that have never been delivered before. When it receives a request for a block that is not on the list, the source ignores the requested block, sends the oldest block on the list instead. This behavior ensures that at least one copy of every block is delivered quickly, and the source's bandwidth does not utilize for sending duplicate copies of blocks.

Pulse [83]

Pulse is another swarm-based P2P protocol for streaming live content. It incorporates a mechanism similar to BitTorrant for rewarding peer participating and discouraging peers from contributing an insufficient amount of resources. It build a bi-directional random mesh in which all peers have the same in-degree while their out-degree is variable. In Pulse, peers do not have a synchronized playout time, therefore, peers can request blocks only from neighbors with overlapping buffer. The scheduling algorithm runs every constant period of time (called epoch) and it is based on an incentive mechanism similar to the one used in BitTorrent. In each scheduling event, a peer chooses a fixed number of potential parents from its neighbor set which

upload most data to it during the last period and have overlapped buffer with the peer. Packets exchange with these parents can be mutual, if both sides have blocks the other one needs. If the peer has still available bandwidth, it will select more peers from its neighbor set by using a *history* parameter which indicates the quality of previous exchanges with other peers. These latter set of peers will be served with less priority and serving them introduces some altruism in the session, and allows high bandwidth peers to contribute more of their capacity to the session. The block selection schemed used by all peers is based on a rarest scheme and blocks are assigned to random peers.

7.3.6. Comparison and discussion. Except for BitTorrent, most of the swarm-based protocols are designed for streaming. Therefore, for the comparison we focus on swarm-based streaming protocols. They mainly differ in the choice of block selection and assignment, source behavior and mesh construction as shown in Table 7.3.4.

Block selection: Among possible block selection algorithms, authors in [173], showed that the ones that prioritize newly available blocks with largest timestamps among parents (*i.e.*, $N*$ algorithms) exhibit a significantly better performance than other algorithms in the resource constraint environments. This implies that in a timing-aware swarming content delivery, the availability of newly generated blocks which further, increases the diversity of blocks for swarming, is more important than addressing timing requirement. In summary, the main drawback of other selection algorithms is their inability to diffuse newly generated blocks in order to ensure the availability of wider range of content which leads to a higher degree of block diversity among peers. [173] observed that any selection algorithms can provide good quality to participating peers by adding sufficient amount of excess resources (*i.e.*, source bandwidth or peer bandwidth) or increasing buffer size at each peer.

Block assignment: [7], [173] examined the effect of $Random$ versus $Min-Ratio$ block assignment policies. It illustrated that the $Min-Ratio$ policy that tends to balance the number of assigned blocks to individual parents proportional to their block budgets in each scheduling event, leads to a better utilization of available bandwidth of parents which exhibits the better performance compared to the $Random$ policy.

Source behavior: Two studies [7], [172] examined the effect of source behavior on the performance of a swarm-based protocol. Through simulations and experiments, they showed that the performance of a P2P streaming protocol (or even non-streaming) directly depends on the source behavior in a resource constraint environments with limited source bandwidth. Essentially, children of source operate in an uncoordinated manner and independently request the same blocks. Therefore, when source bandwidth is not significantly more than one stream bandwidth and source behaves similar to any ordinary parent, a single copy of the stream can not be diffused to the session in a timely manner. A special behavior in source, is required to effectively utilize source outgoing bandwidth and increase the rate of delivery of new blocks to the session. In summary, source bandwidth is a precious resource and it is important not to waste it on duplicate blocks until all blocks have been diffused to the session at least once.

Mesh construction: One of the desirable characteristics of swarm-based approach is its simple overlay construction with low maintenance. Therefore, simplicity in the construction of a mesh is an important metric. Random mesh overlay clearly has the most simple with low maintenance construction [100]. Any mesh overlay with specific properties such as biased connectivity or DAG needs special construction mechanism and extra maintenance overhead to keep the properties of the overlay.

The other important metric is robustness of a mesh overlay that significantly affects the performance of the swarm-based approach. The more robust and resilience to transience of peers a mesh overlay is, the higher the performance and delivered quality to individual peers will be. Connectivity is an inherent characteristics of random mesh overlays [165], whereas in overlays with biased connectivities the resiliency decreases. DAG overlay has the highest resiliency with the guaranteed connectivity property. This good connectivity property helps constructing and maintaining a biased DAG overlay (*e.g.*, locality

aware DAG). However, in DAG overlays, to prevent any loops, some of the peers can not have any children and their outgoing bandwidth can not be utilized which further, limits the scalability. Overlays with biased connectivity are highly vulnerable to partitions [165].

The last metric that can potentially affects the network-wise performance of the swarm-based approach is its efficiency and being location-aware. However, constructing a mesh overlay that is location-aware lowers the connectivity and simplicity of the mesh.

7.4. Hybrid Approach:Mesh-Tree

Hybrid approach is a combination of the above approaches and is suitable for one-to-many communication model. A hybrid approach can be achieved by combining push and pull for content delivery [103] or tree and mesh for the content delivery path [89]. In protocols incorporates mesh-tree hybrid approach, participating peers receive part of the content through a tree overlay and the rest of the content is pulled from mesh parents. Two existing protocols that incorporate the hybrid approach are Bullet [103] and mTreebone [89], which we further describe in detail.

7.4.1. Case Studies.

Bullet [103]

Bullet is designed for elastic content delivery. In Bullet, peers are organized into an overlay tree, which can be constructed and maintained by any of the existing tree construction mechanisms [79], [73], [139], [162]. Each Bullet peer, starting with the source of the tree, transmits a *disjoint* set of blocks to each of its children, with the goal of maintaining uniform representativeness of the content across all participating peers. The level of disjointness is determined by the bandwidth available to each of its children. Bullet then employs a distributed peer discovery to enable peers to quickly locate multiple parents capable of transmitting missing blocks of the content to the peer without global knowledge. Bullet incorporates RanSub [102] to periodically disseminate the changing, uniformly random subsets of global state to each participating peer. Each peer uses this information to request missing blocks from other participating peers. RanSub distributes the content availability information of random subsets of participating peers through the tree using *collect* and *distribute* messages. Collect messages start at the leaves and propagate up the tree, leaving state at each peer along the path to the source. Distribute messages start at the source and travel down the tree, using the information left at the peers during the previous collect round to distribute uniformly random subsets to all participants. Each peer as a parent creates distribute sets for each child by compacting collects sets from that child's siblings and its own distribute set. The results is a state information of a random subset of participating peers representing all peers in the tree, except for those rooted at that particular child.

In essence, during each interval, a peer receives a summarized partial view of the session's state at that time. Upon receiving a random subset, a Bullet peer may choose a peer with the minimum amount of shared blocks compared to its own available blocks to be its perpendicular (mesh) parent. This is done only when the peer has sufficient slots based on its bounded in-degree (in Bullet it is 10) to request for another parent (parents with poor performance may be removed). Once a peer has chosen the potential perpendicular parent, it sends a peering request containing its Bloom filter. The new perpendicular parent will transmit blocks not present in the Bloom filter to the child peer. The child peer will refresh its Bloom filters at each of its perpendicular parents, periodically. Along with the fresh bloom filter, a child will also assign a pre-determined fixed portion of the sequence space to each of its perpendicular parents to reduce the likelihood of receiving duplicate blocks. A duplicate block, however, may be received when a parent (through the tree structure) recovers a block from one of its parents and relays the

block to its children (and descendants). Less than 10% of all received blocks are duplicates in their experiments.

mTreebone [89]

mTreebone is another mesh-tree hybrid protocol which relies on constructing a tree with only stable peers as a backbone and pushing the streaming content over this backbone. These stable peers, together with other participating peers, are further organized a random mesh overlay, which facilitates accommodating churn and fluctuation of bandwidth in the backbone. The core of mTreebone is constructing a tree-based backbone, referred to as *treebone*. This backbone consists of only a subset of peers which are stable. Other non-stable peers are attached to the backbone as outskirts. The streaming content is pushed through the treebone and eventually reach the outskirts. To improve the resiliency and efficiency of the treebone, participating peers are organized into a mesh overlay. A gossip-based membership management algorithm is exploited for peers to periodically exchange their status. The mesh neighbors periodically exchange their content availability information. However, a peer is schedule to pull content from neighbors in the mesh only if an disruption in delivery of the content occurs in the treebone.

mTreebone identifies stable peers through their session length (peers with higher age tend to stay longer). If the session length of a peer exceeds a certain threshold, it will promote itself as a treebone peer and further can accept children. Upon joining, each peer obtains a partial list of participating peers from source, at least one of which is in the treebone. The new peer attaches itself to one of the treebone peers and locates mesh neighbors using the list. To optimize latency of the treebone, if a treebone peer has more children than a peer closer to the source in the treebone, a swap of them occurs to reduce the average depth of the treebone. Moreover, each treebone peer tries to move upward in the tree by periodically checking whether there are peers closer to the source than its parent that can accept a child. If so the peer leaves its original parent and attach itself to the closer peer as a child. Without any disruption in the delivery of the streaming content, peers keep receiving the whole content from their single parent in the treebone. In case of a departure of any treebone peers or bandwidth reduction in any upstream treebone connections, affected children try to request missing blocks from their mesh parents.

7.5. Discussion

In the above P2P content delivery approaches, each peer can receive the content from one or multiple parents. Therefore, we can broadly classify the above approaches into *single-parent* and *multiple-parent* categories. The single-tree-based approach falls into single-parent category as each peer has a single parent for receiving the content, while all the other approaches are multiple-parent that peers receive the content from multiple parents. In this section, we first present the limitations of the single-parent or single-tree-based approach[5] and further discuss the similarities and differences of approaches fall into the multiple-parent category.

7.5.1. Single-Parent vs. Multiple-parent category. Single-parent protocols have some inherent drawbacks compared to protocols fall into the multiple-parent category as follows:

- In any efficient (i.e. low-depth) single-tree-based protocol, a small number of interior peers carry the burden of forwarding the content to the rest of the peers, while most peers are leaves and do not contribute their bandwidth. In the context of streaming a video content, this problem is more of an issue as many peers may not have enough outgoing bandwidth required for an interior peer to streaming the whole content to multiple children. This limits the scalability and thus the number of

5. We use the term single-parent and single-tree-based interchangeably through this section.

participating peers.

- The other issue with single-tree-based approach is that the received bandwidth and thus quality to each peer is limited by the minimum bandwidth on the path from the source. Any loss or bandwidth fluctuation in the upper levels of the tree reduces the bandwidth available to peers lower in the tree and affects their delivered quality. A number of techniques have been proposed to recover from losses and hence improve the available bandwidth in a tree [180], [57], [129]. However, fundamentally, the delivered stream quality (or the download time) to any peer is limited by the available bandwidth from that peer's single parent and the delivered stream quality (or the total download time) in the parent[89]. In other words, the delivered quality (or the total download time) down an overlay tree is guaranteed to be monotonically decreasing. Therefore, in a large overlay tree, participating peers in lower depth (near the leaves) are likely to experience more losses and late packet arrivals, thus, lower quality or larger total download time.

- The single-tree-based approach has poor resilience to churn. As each peer only receives the content from its single parent, the failure of its parent causes interruption in receiving the content. Moreover, the departure of a single peer, particularly one close to the source, may result in the disruption in receiving the content for all of the descendants of the departed peer until the tree is repaired. This dramatically impacts the overall reliability of the single-tree-based approach and may cause scalability concerns under churn as a potentially large number of peers (all descendants) attempts to rejoin the tree which brings extra overhead and may prevent the tree structure from being optimal.
- In the context of streaming, the residual outgoing bandwidth of interior peers can not be used in the single-tree-based approach. This lowers the utilization of available bandwidth among interior peers which further affects the scalability of the single-tree-based approach [140]. For example, if an interior peer has an outgoing bandwidth of 600 Kbps, and the stream is encoded at 250 Kbps for a single tree, then the peer with two children has a residual bandwidth of 100 Kbps that is unused. In contrast, multiple-parent approaches can effectively utilize the residual outgoing bandwidth of individual peers due to a smaller content mapping granularity to each connection, *i.e.*, block level in the swarm-based and sub-stream level in the multiple-tree-based.

7.5.2. Multiple-Parent Category: Similarities and Differences. In the multiple-parent category, each peer has multiple parents that receive the content from and multiple children that delivers the content to. It is widely accepted that only multiple-parent approaches can lead to high utilization of all peers bandwidth, since they allow all participating peers to contribute their bandwidth and participate in delivery of the content [93]. The approaches fall into the multiple-parent category are multiple-tree-based, swarm-based and hybrid approaches that have a great deal of similarities as follows:

- While these approaches use different overlay construction mechanisms, the overall shape of their resulting overlays is very similar. More specifically, the superimposed view of multiple diverse trees or mesh-tree is in fact the same as a mesh.
- The content delivery in both approaches enable individual peers to receive different pieces of the content from different parents. At the local view, each peer receives content from multiple parents and sends content to multiple children. At the global view, in all of the corresponding approaches, the collection of edges used for the delivery of a single block from source to all participating peers form a source-rooted tree that we call *delivery tree* [164].
- These approaches require participating peers to maintain a loosely synchronized playout time that is sufficiently behind source's playout time. This requires a specific seconds worth of buffering at each peer which accommodates the diversity of different path from source in multiple-tree approach, and out-of-order block arrival in the swarm-based and also hybrid approaches. The amount of buffering

depends on the maximum hop count from source to different participating peers through the overlay which is a function of peer population and peer degree.

The key difference between the swarm-based, multiple-tree-based and hybrid approaches is how the delivery tree of individual blocks is formed, in other words, how the content is mapped to parents. This difference determines the performance of these approaches as will be discussed in the next subsection.

Mapping of the content has two dimensions of *frequency* and *granularity*. The frequency of mapping can be *event-driven* or *periodic*. The frequency of mapping in the multiple-tree-based protocols is event-driven. The incident that triggers the change of the mapping in the event-driven mapping case, can be departure of a parent as in CoopNet [77] or it can be adaptation to network condition changes such as fluctuation in bandwidth which is employed in CoolStreaming [80] and Chunkyspread [93]. On the other hand, in swarm-based protocols such as PRIME [7], the mapping of content to parents performed periodically.

The granularity of mapping can be *(i)* a list of blocks with flexible size, or *(ii)* a pre-determined description or sub-stream which is firm in bitrate. The block-granularity mapping is flexible in the sense that the number of blocks mapped to each parent can be adjusted based on the estimated available bandwidth from the associated parent. On the other hand, the sub-stream-granularity mapping requires a guaranteed lower bound bandwidth for the parent's connection and expects that the parent's connection bandwidth is sufficient to deliver the sub-stream at the proper bitrate.

Generally, when the mapping is event-driven the granularity has to be firm or description/sub-stream-based as in multiple-tree-based protocols. The reason is that the duration of the mapping is not known as a priory, therefore, the mapping can not be a limited list of blocks. On the other hand, in periodic mapping, content is mapped to each connection for the duration of a fixed interval. Therefore, based on the available bandwidth of the connection, a limited number of blocks can be mapped to the connection/parent. Essentially, in multiple-tree-based protocols, the mapping is long term which relies on the assumption of persistent network conditions. On the other hand, in periodic mapping cases, the design assumptions lies on the variation of network conditions and the protocol tries to adjust the mapping accordingly in each interval.

7.5.3. Performance comparison. Comparing the P2P content delivery approaches depends on the desired performance metrics such as maximum and consistent perceived quality in the context of streaming or total download time in the context of file distribution, high bandwidth utilization, low control overhead and small buffer size (delay) for streaming under various network environments. Ensuring each of the above metrics can achieve some of the goals of a P2P application as follows:

- Maximizing the delivered quality and its consistency are counted to be an inherent performance metric of streaming applications as it determines the user satisfaction and resiliency of the approach. This is analogous to minimizing the total downloading time in the context of elastic content delivery.
- A high bandwidth utilization can increase the scalability of the P2P content delivery approaches by allowing more peers join the session.
- An efficient and scalable approach should have low percentage of control overhead for overlay construction, maintenance and content delivery.
- Finally, a small delay from source to all peers is guaranteed the liveliness in the context of live streaming applications.

Besides, a P2P content delivery approach should perform well and efficiently under various network characteristics such as churn, heterogeneity of peers' bandwidth, resource constraint, bandwidth fluctuation and group size.

There exists few performance comparison studies that compare specific protocols from the above approaches in the context of streaming applications. We discuss their findings along with inherent limitations and strengths of the streaming approaches fall into the multiple-parent category as follows:

(i) Perceived quality: The delivered quality to individual peers is an important aspect of P2P streaming. Authors in [164], compare a swarm-based protocol (*i.e.*, PRIME) and a non-adaptive multiple-tree-based protocol (*i.e.*, CoopNet [77]) and showed that swarm-based approach can exhibits a superior performance and receive higher delivered quality across a wide range of network environments. A similar comparison study [181] is performed between Chainsaw [174] and SplitStream [95] and showed that the swarm-based protocol (*i.e.*, Chainsaw) generally yields a higher quality.

In the multiple-tree-based approach, the event-driven long term mapping of sub-streams to connections adversely affects the delivered quality to individual peers under dynamic network environment (*i.e.*, peers' available bandwidth fluctuates). When the bandwidth of a connection of a parent to its child is less than the sub-stream bandwidth, the blocks for that sub-stream can not be "streamed" at a proper rate to the child and subsequently to all the peers that are receiving that sub-stream through a path from this parent (descendants of the particular parent in the corresponding sub-stream tree). In other words, the effect of a low bandwidth connection extends to all the downstream peers on the particular tree. In non-adaptive multiple-tree-based protocols, *i.e.*, CoopNet, this severely affects the perceived quality at individual peers, while in adaptive multiple-tree-based protocols, *i.e.*, ChunkySpread [93], this decrease in bandwidth triggers the event of changing the mapping. However, switching or changing of the mapping adds extra complexity and overhead specially in a highly dynamic network environment. In contrast, in the swarm-based and mostly hybrid approaches, the periodic mapping of blocks to connections makes these approaches inherently adaptive and enable them to maximize the delivered quality by effectively utilizing the available resources despite fluctuation in bandwidth. In particular, when a connection has low bandwidth, its children and descendant peers can still receive their required blocks through alternative paths from other parents. This minimizes the impact of a low bandwidth connection on the connected child peers and all descendants.

Dynamics of peer participating can also adversely affect the perceived quality of individual peers. Some peers may find themselves disconnected or experience a temporary disruption in receiving the streaming content. The time that it takes for a disconnected peer to rejoin the overlay, the number of affected peers by a single peer departure and the amount of available buffer determine the perceived quality of peers under dynamic network and consequently the resiliency of a protocol. All approaches fall into multiple-parent category, are resilient and can cope with peer dynamics due to not relying on single parent to receiving the content from. However, the multiple-tree-based approach is more vulnerable to departure of a peer than a swarm-based or hybrid approach. As the departure of any interior peer in a particular tree results in a disruption of the streaming to all the descendants of that peer in the particular tree. Authors in [181], investigate the resiliency of multiple-tree-based and swarm-based approaches and find that swarm-based approach has higher stability than multiple-tree-based approach.

(ii) Bandwidth utilization: The utilization of outgoing bandwidth of participating peers determines the amount of available resources which controls the number of peers that can be accommodated. In the multiple-tree-based approach, the sub-stream-granularity mapping of content to connections dramatically decreases the bandwidth utilization of peers. When the bandwidth of a particular connection is more than the sub-stream bandwidth but less than two times of it, the residual of the connection bandwidth can not be utilized. For instance, if the available bandwidth of a parent to one of its child is 100 Kbps and the sub-stream rate is 70 Kbps, 30 Kbps residual bandwidth of the parent can not be utilized. This limits the ability of multiple-parent approach to utilize the total resources and limits its scalability. In contrast, in the swarm-based and mostly hybrid approaches, the the block-granularity mapping enables them to effectively utilize the available resources despite fluctuation in bandwidth. These approaches can fully utilize available bandwidth of all connections by adjusting the number of blocks requested in each interval.

(iii) Control overhead: The overhead of overlay maintenance and content delivery can affect the scalability and efficiency of a P2P streaming protocol. A multiple-tree-based protocol should deal with

tree construction and maintenance overhead while in a swarm-based protocol overlay construction is relatively simple with low overhead. However, periodic content reporting and requesting adds extra control traffic in swarm-based protocols.

(iv) Delay: In the context of live streaming, a P2P approach should minimize the delay for delivery of the streaming content from source to individual peers to make the session as live as possible. The amount of delay also determines the buffer size required at each peer. The multiple-tree-based approach has a smaller delay in streaming the content from source to individual peers. The delay in a multiple-tree-based protocol is proportional to the maximum hop count from source to each peer across various path. On the other hand, in the swarm-based approach, due to the nature of swarming and reports/requests the delay is longer. In fact, there is an efficiency-delay tradeoff in swarm-based approach [175]: if peers choose to request any newly available block from the parents, the overhead will be excessive; periodical requests containing a list of reported blocks reduces the overhead, but increase the delay. A well-designed hybrid protocol might be able to reduce the control overhead compared to swarm-based approach and decrease the delay.

8. Related Topics

Despite popularity of P2P content delivery applications, there are some issues that has not completely addressed yet and remained as open issues specially in the context of streaming content delivery. Dealing with incentive mechanism and incorporating network coding into streaming applications are the ones that we discuss here.

8.1. Incentive and fairness

So far we have made an implicit assumption that peers can and are willing to cooperate in delivery of the streaming content. However, in P2P networks, this is not always the case. Several studies [184], [185] have shown that users of P2P networks tend to be selfish and try to benefit from the P2P network without contributing as much resources in return. An extreme example of uncooperative behavior is the ”free-rider”, where a peer only consumes the bandwidth without contributing any. Therefore, a proper incentive mechanism that encourages peers to contribute and upload as much as they can is critical in P2P streaming applications. In the absence of such a mechanism, the performance of the P2P streaming application can seriously degrades or it can be variable and unpredictable.

Providing incentives in highly dynamic P2P networks where it is difficult to identify peers and obtain the information about their past behavior in order to predict their future performance, can be a particularly challenging task. P2P file sharing applications adopted various incentive mechanisms, based on payment [186], or reciprocity, to encourage peers to contribute [187], [186], [188], [189], [190], [191]. However, designing incentive mechanisms for P2P streaming applications is more challenging than P2P file sharing applications due to the unique features of streaming applications *i.e.*, real-time constraints and bandwidth requirement.

Payment-based mechanisms dictate that the service recipients simply pay the service providers for resources they consume. [186] is one of the first studies that considered payment-based mechanisms in P2P file sharing applications. This study uses a game theoretical model to study the potential benefits of incorporating payment-based incentive mechanisms into P2P file-sharing applications. Various payment-based mechanisms have been proposed in the context of P2P file-sharing. However, these mechanisms seem highly impractical even in P2P file sharing applications, since they require an infrastructure for accounting and micro-payments.

In reciprocity-based mechanisms, peers maintain histories of past behavior of other participating peers and use this information in their decision making process. The reciprocity mechanism can be based on

indirect reciprocity or *direct reciprocity*. In indirect mechanisms, the decision of peer X about Y is based on the contribution of peer Y to the whole P2P network. In contrast, in direct-reciprocity mechanisms, peer X decides how to treat Y based only on the contribution of peer Y to X in the past. In general, indirect-reciprocity mechanisms are vulnerable to collusive behavior such as false accusation and false praise [192] that do not arise in direct-reciprocity mechanisms.

Based on the time duration needed for reciprocation, reciprocity-based mechanisms can be further divided into *reputation-based* and *instantaneous* mechanisms. Reputation-based mechanisms rely on the history of contribution of a peer to the P2P network. In general, peers that tend to contribute more obtain higher reputations and may be prioritized when they download. For instance, Kazaa [3] incorporates an reputation-based indirect reciprocity mechanism in which peers earn reputation scores by uploading, and highly reputed peers receive preferential treatment in their downloads. Overall, reputation-based incentive mechanisms are feasible in the context of file-sharing applications, because the total time to download a file can often be long providing sufficient time to collect enough credits or build reputation. Further, peers downloading a file can tolerate slow download rates for a period of time. However, due to the strict time and bandwidth constraints and intolerance to long delays, the reputation-based mechanisms might not be trivially extended to P2P streaming applications.

Instantaneous direct reciprocity mechanisms relax the need for maintaining long-term state information, in the form of reputation. This simplifies the design and improves the robustness of the mechanism against collusive behavior. BitTorrent adopts a form of direct reciprocity by embedding a tit-for-tat strategy [193] to incentivize peer contributions. In particular, peers preferentially upload to peers from whom they are able to download at a fast rate in return. Studies found much lower levels of free-riding in BitTorrent network compared to other P2P file sharing applications. However, measurements and analysis has demonstrated that the BitTorrent protocol can still be manipulated by misbehaved peers in their favor and the fairness properties of BitTorrent is questionable [194], [195]. Although, the tit-for-tat strategy or its extended versions [196], [172] work well in the context of file sharing,it cannot be trivially extended to the context of P2P streaming because of the timeliness and the high bandwidth requirements involved. Moreover, the direct-reciprocity incentive mechanism requires direct interactions between each pair of peers which might have some implications on the properties of the overlay structure for P2P streaming applications.

Another flavor of incentive mechanisms is to encourage peers to contribute as much bandwidth as they have rather than focusing on a bit-for-bit fair sharing where a peer receives as much bandwidth as it contributes. For that, authors in [197], [198], [199], consider taxation schemes in which bandwidth-rich peers try to compensate the resource-poor peers by contributing more bandwidth to the system and proportionally receiving higher bandwidth. The intuition is that with a bit-for-bit mechanism, peers with very high upload capacity (*i.e.*, behind Ethernet) that are capable of contributing much more than the source rate, can not contribute all of their resources, while peers with low upload capacity (*i.e.*, behind asymmetric connections such as DSL and cable modem) are precluded from receiving more than their upload capacities. The basic mechanism in this approach, is a contribution-aware framework where peers receive different levels of bandwidth based on the overall instantaneous upload bandwidth available in the system as well as the amount of resources the peer contributes. This contribution-aware mechanism is of a flavor of indirect-reciprocity mechanisms and relies on trusting the participating peers about their announced instantaneous contributions to the system.

8.2. Network coding

A traditional assumption in many P2P applications, is that each peer performs as a switch in the sense that it does nothing to the received blocks but replicating and forwarding. Network coding is a new tool in information theory that breaks this assumption by letting a peer functions as an encoder in the sense

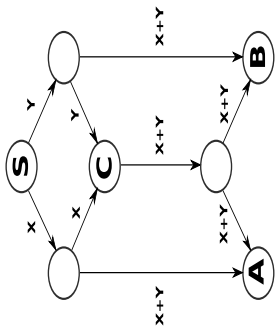


Figure 4. A butterfly network [205], [206]

that it receives blocks, encodes and then forwards them [201]. The fundamental insight in network coding is that if data can be encoded in intermediate peers and all the links are error-free then optimal utilization of the resources of a system can be achieved [201], [202], [203], [204].

With a simple network coding, every transmitted block is a linear combination of all or a subset of the blocks available at the sender (similar to XORing multiple blocks). Further, the encoded blocks can be recombined to generate new linear combinations, enabling peers to generate encoded blocks without having the full file. The original information can be reconstructed after receiving enough linearly independent blocks. Some P2P file sharing applications incorporate network cording. The benefit of network coding with respect to improving throughput can be best presented in the "Butterfly" example, as shown in Figure 4. In this figure, each link has the capacity to send one block and the source S wants to maximize the throughput to both A and B. With network coding, maximum throughput can be increases from 1 to 2. Node C can encode its received blocks x and y into $x + y$ where $+$ can be a liner combination of both blocks. This results in an effective end-to-end throughput of 2. Several studies have investigated the practicality of network coding in the P2P file sharing applications show that it is indeed applicable [207], [208], [209]. An example is Avalanche [208], which shows that network coding may improve the overall performance of P2P file sharing applications by more than $2 - 3$ times. The intuition is that, with network coding, all blocks are treated equally, without the need of some complex schedulings (*e.g.*, distribute the "rarest block" first). Authors in [205], shows that the performance of network coding from the point of view of coding complexity and real-world coding in the context of P2P file sharing applications is acceptable when one uses a small number of blocks, in the order of less than a thousand.

The practicality of network coding in the context of P2P streaming applications has yet to be explored. Due to the distinguishing properties of streaming, additional issues and challenges are introduced when applying network coding into P2P streaming applications. In the context of streaming, given the synchronized playback of all participating peers and the continuously generating new content by source, the window of interest for all participating peers is moving and not all the blocks can be encoded at once. In fact, the buffer at each peer has to be updated over time to remove obsolete blocks. This is different from file sharing where the buffer is allocated for all the blocks of the file during its download time. Chou [202] has suggested that a stream can be divided into generations. Clearly, the coding efficiency is improved with a longer generation, but the startup delay can be increased with longer generations. Moreover, the combination of video coding (*e.g.*, MDC) and network coding for supporting heterogeneity of bandwidth can be even more complex. Overall, applying network coding over small windows of the streaming content may compensate the effect of a naive block scheduling mechanism and reduces the risks of missing deadline.

9. Conclusion

In this report, we survey the design of P2P content delivery mechanisms which has been in top 10 most active area of research in networking. We have looked at the roots and rationale behind incorporating P2P for content delivery. Compared to IP multicast, P2P is easy to deploy, scalable and can support higher level functionalities. However, P2P incurs longer delay and is less efficient. Relaxing the need for network infrastructure modification, along with providing the ability to easily apply any application-specific requirements have resulted the serious consideration and development of P2P architecture for content delivery. P2P offers new opportunities to support various applications but introduces new challenges. The key challenges comes from the combined effect of content type and communication model.

P2P has gained a tremendous amount of attention from researchers for supporting different content delivery applications. These includes video conferencing, live event broadcasting, file distribution, IP telephony, on-line chat and file download.

Although P2P is considered as an active area of research over the last decade, still there are many open issues specially in the context of P2P streaming applications. Design of a P2P streaming application for scalable delivery of a high quality stream is challenging due to its unique characteristics and requirements. This survey offers a comprehensive reference and captures various existing approaches, their design goals, strengths and weaknesses along with some of the open issues in this field.

[1] B. Cohen, "Bittorrent." [Online]. Available: http://www.bittorrent.com

[2] Gnutella, "http://www.gnutella.com."

[3] Kazaa, "http://www.kazaa.com."

[4] Napster, "http://www.napster.com."

[5] Emule, "http://www.emule-project.net."

[6] B. Alfonsi, "I want my iptv: Internet protocol television. predicted a winner," in *IEEE Distributed Systems Online*, 2005.

[7] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in *INFOCOM*, 2007.

[8] J. Kurose and K. Ross, *Computer Networking, A top-Down Approach Featuring the Internet*. Addison Wesley, 2002.

[9] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "A peer-to-peer on-demand streaming service and its performance," in *ICME*, jul 2003.

[10] K. Hua and S. Sheu, "Skyscraper broadcasting: A hybrid broadcasting scheme for metropolitan video on demand systems," in *ACM SIGCOMM*, Cannes, France, 1997.

[11] S. Sheu, K. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video-on-demand systems," in *International Conference on Multimedia Computing and Systems*, 1997.

[12] S. Ramesh, I. Rhee, and K. Guo, "Multicast with cache (mcache): An adaptive zero-delay video-on-demand service," in *INFOCOM*, 2001.

[13] J.-F. Paris, S. Carter, and D. Long, "A hybrid broadcasting protocol for video on demand," in *Multimedia Computing and Networking*, San Jose, CA, jan 1999.

[14] R. Zimmermann and L. S. Liu, "Active: Adaptive low-latency peer-to-peer streaming," in *SPIE/ACM Conference on Multimedia Computing and Networking*, New York, NY., mar 2005.

[15] D. A. Tran, K. A. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *INFOCOM*, 2003.

[16] "www.vseelab.com."

[17] S. E. Deering, "Multicast routing in internetworks and extended lans," in *ACM SIGCOMM*. ACM, aug 1988.

[18] S. Deering and D. Cheriton, "The pim architecture for wide-area multicast routing," *ACM/IEEE Transactions on Networking*, 1996.

[19] M. et al., "Periodic broadcast and patching services - implementation, measurement, and analysis in an internet streaming video testbed," in *ACM Multimedia*, 2001.

[20] J. Saltzer, D. Reed, and D. Clark, "End-to-End Arguments in System Design," in *ACM Transactions on Computer Systems*, 1084.

[21] S. Deering and D. Cheriton, "Multicast routing in datagram internetworks and extended lans," *ACM Transactions on Computer Systems*, 1990.

[22] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing." in *ACM SIGCOMM*, 1995.

[23] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharyya, "Reliable multicast transport protocol (rmtp)," in *IEEE INFOCOM*, 1996.

[24] C. Papadopoulos, G. Parulkar, and G. Varghese, "An error control scheme for large-scale multicast applications," in *IEEE INFOCOM*, 1998.

[25] T. Speakman, D. Farinacci, S. Lin, and A. Tweekly, "Pgm reliable transport protocol." *Internet Draft*, 1998.

[26] D. DeLucia and K. Obraczka, "A multicast congestion control mechanism for reliable multicast." in *INFOCOM*, 1997.

[27] L. Rizzo, "Pgmcc: a tcp-friendly single-rate multicast congestion control scheme," in *ACM SIGCOMM*, 2000.

[28] J. Widmer and M. Handley, "Extending equation-based congestion control to multicast applications," in *ACM SIGCOMM*, 2001.

[29] J. Macker and R. Adamson, "A tcp friendly, rate-based mechanism for nack-oriented reliable multicast congestion control," in *IEEE GLOBECOM*, 2001.

[30] J. Byers, M. Luby, and M. Mitzenmacher, "Fine-grained layered multicast," in *INFOCOM*, 2001.

[31] T. Montgomery, "A loss tolerant rate controller for reliable multicast," Tech. Rep., 1997.

[32] B. Whetten and J. Conlan, "A rate based congestion control scheme for reliable multicast," Tech. Rep., 1998.

[33] I. Rhee, N. Ballaguru, and G. N. Rouskas., "Mtcp: Scalable tcp-like congestion control for reliable multicast," in *INFOCOM*, 1999.

[34] S. Kasera, J. Kurose, and D. Towsley, "Scalable reliable multicast using multiple multicast groups," *IEEE-ACM Transactions on Networking*, 1996.

[35] S. McCanne, V. Jacobson, and M. Vettereli, "Receiver-driven layered multicast," in *ACM SIGCOMM*, Aug 1996.

[36] L. Vicisano, L. Rizzo, and J. Crowcroft, "Tcp-like congestion control for layered multicast data transfer," in *INFOCOM*, 1998.

[37] A. Legout and E. Biersack, "Plm: fast convergence for cumulative layered multicast transmission schemes," in *ACM SIGCOMM*, 2000.

[38] I. Khayat and G. Leduc, "control for layered multicast transmission," *Networking and Information Systems Journal*, 2000.

[39] J. Byers, G. Horn, M. Luby, M. Mitzenmacher, and W. Shaver, "Flid-dl: Congestion control for layered multicast," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATION*, 2002.

[40] J. Byers and G. Kwon, "Stair: Practical aimd multirate multicast congestion control," *Lecture Notes in Computer Science*, 2001.

[41] D. Sisalem and A. Wolisz, "Mlda: a tcp-friendly congestion control framework for heterogeneous multicast environments," in *IWQoS*, 2000.

[42] J. Liu, B. Li, and Y. Zhang, "hybrid adaptiation protocol for tcp-friendly layered multicast and its optimal rate allocation," in *IEEE INFOCOM*, 2002.

[43] G. Kwon and J. Byers, "Smooth multirate multicast congestion control," in *IEEE INFOCOM*, 2003.

[44] S. Ramakrishnan and B. Jain, "A negative acknowledgement with periodic polling protocol for multicast over lans," in *IEEE INFOCOM*, 1987.

[45] M. Handley and J. Crowcroft, "Network text editor (nte): A scalable shared text editor for the mbone," in *ACM SIGCOMM*, 1997.

[46] J. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the internet," in *ACM SIGCOMM*, London, UK, sep 1994, pp. 58–67.

[47] A. Hu, "Video-on-demand broadcasting protocols: A comprehensive study," in *IEEE INFOCOM*, 2001.

[48] A. Mahanti, D. Eager, and C. Williamson, "Temporal locality and its impact on web proxy cache performance," in *Performance Evaluation, Special Issue on Internet Performance Modeling*, 2000.

[49] S. Viswanathan and T. Imielinski, "Pyramid broadcasting for video on demand service," in *MMCN*, 1995.

[50] S. Carter and D. Long, "Improving video on-demand server efficiency through stream tapping," in *IEEE ICCN*, 1997.

[51] L. Gao and D. Towsley, "Supplying instantaneous video on-demand services using controlled multicast," in *IEEE International Conference on Multimedia Computing and Systems*, 1999.

[52] D. Eager, V. M, and J. Zahorjan, "Minimizing bandwidth requirements for on-demand data delivery," *IEEE Transactions on Knowledge and Data Engineering*, sep 2001.

[53] ——, "Bandwidth skimming: A technique for cost-effective video on demand," in *Multimedia Computing and Networking*, San Jose, CA, jan 2000.

[54] K. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true on-demand services," in *ACM Multimedia*, 1998.

[55] A. Bestavros, "Aida-based real-time fault-tolerant broadcast disks," in *IEEE RTAS*, 1996.

[56] M. Rabin, "Efficient dispersal of information for security, load balancing and fault tolerance," *Journal of the Association for Computing Machinery*, 1997.

[57] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *ACM SIGCOMM*, 2002.

[58] J. W. Byers, M. Luby, and M. Mitzenmacher, "A digital fountain approach to asynchonous reliable multicast," 2002.

[59] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment Issues for the IP Multicast Service and Architecture," in *IEEE Network Magazine*, 2000.

[60] H. Holbrook and D. Cheriton, "Ip multicast channels: Express support for large-scale single-source applications," in *ACM SIGCOMM*, 1999.

[61] Y. Chawathe, "Scattercast: An architecture for internet braodcast distribution as an infrusturucture service," in *Ph.D. Thesis*, San Jose, CA, Jan. 2000.

[62] Y. Chawathe, S. McCanne, and E. Brewer, "RMX: reliable multicast for heterogeneous networks," in *IEEE INFOCOM*, 2000.

[63] Y. Chu, S. G. Rao, and H. Zhang, "A case for end-system multicast," in *ACM SIGMETRICS*, jun 2000.

[64] P. Francis, "Yoid: Extending the multicast internet architecture," in *White Paper, http://www.icir.org/yoid*, 1999.

[65] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *NOSSDAV*, June 2001. [Online]. Available: citeseer.nj.nec.com/zhuang01bayeux.html

[66] R. Melamed and I. Keidar, "Araneola: A Scalable Reliable Multicast System for Dynamic Environments," in *IEEE NCA*, 2004.

[67] J. Almeida, D. Eager, M. Ferris, and V. M, "Provisioning content distribution networks for streaming media," in *INFOCOM*, 2002.

[68] "www.sandpiper.net."

[69] "www.mirror-image.com."

[70] Akamai, "http://www.akamai.com."

[71] "www.digitalisland.co.nz."

[72] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: A framework for delivering multicast to end users," in *INFOCOM*, 2002.

[73] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end-system multicast," in *IEEE Journal of Selected Areas in Communication*, 2002.

[74] B. Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-resilient wide-area location and routing," in *Tech. Report UCB/CSD-01-1141*, apr 2001.

[75] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *ACM SIGCOMM*, 2001.

[76] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," in *ICNP*, 2003.

[77] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *NOSSDAV*, Miami Beach, FL, 2002.

[78] W. T. Ooi, "Dagster: Contributor aware end-host multicast for media streaming in heterogeneous environment," in *Multimedia Computing and Networking*, jan 2005.

[79] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *ACM SIGCOMM*, 2002.

[80] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "Coolstreaming: A data-driven overlay network for live media streaming," in *INFOCOM*, 2005.

[81] Pplive, "http://www.pplive.com."

[82] R. Rejaie and S. Stafford, "A framework for architecting peer-to-peer receiver-driven overlays," in *NOSSDAV*, 2004.

[83] F. Pianese, J. Keller, and E. W. Biersack, "Pulse, a flexible p2p live streaming system," in *Global Internet Workshop*, 2006.

[84] S. M. Hedetniemi, S. T. Hedetniemi, and A. Liestman, "A Survey of Gossiping and Broadcasting in Communication Networks," in *Networks*, 1988.

[85] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *ACM Symposium on Principles of Distributed Computing*, 1987.

[86] Q. Sun and D. Sturman, "A gossip-based reliable multicast for large-scale high-throughput applications," in *International Conference on Dependable Systems and Networks*, 2000.

[87] R. Renesee, Y. Minsky, and M. Hayden, "A gossip-style failure detection services," in *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, sep 1998.

[88] A. Kermarrec, L. Massoulie, and A. Ganesh, "Probabilistic reliable dissemination in large-scale systems," *IEEE Transactions on Parallel and Distributed Systems*, 2003.

[89] F. Wang, Y. Xiong, and J.Liu, "mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast," in *ICDCS*, 2007.

[90] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie, "Peer-to-peer membership management for gossip-based protocols," *IEEE Trans. Computers*, 2003.

[91] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massouli, "From epidemics to distributed computing," in *IEEE Computer*, 2003.

[92] F. Ball and A. Barbour, "Poisson approximation for some epidemic models," *Applied Probability*, 1990.

[93] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous Unstructured End System Multicast," in *ICNP*, 2006.

[94] V. Vishnumurthy and P. Francis, "On heterogeneous overlay construction and random node selection in unstructured p2p networks," in *INFOCOM*, 2006.

[95] M. Castro, P. Druschel, A.-M. Kermarrec, A. R. A. Nandi, and A. Singh, "Splitstream: High-bandwidth content distribution in a cooperative environment," in *ACM SOSP*, 2003.

[96] A. Sharma, A. Bestavros, and I. Matta, "dpam: A distributed prefetching protocol for scalable asynchronous multicast in p2p systems," in *INFOCOM*, 2005.

[97] I. Stoica, R. Morris, D. Krager, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, aug 2001.

[98] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP*, 2001.

[99] Z. Xu, C. Tang, S. Banerjee, and S. Lee, "Rita: Receiver initiated just-in-time tree adaptation for rich media distribution," in *ACM NOSSDAV*, 2003.

[100] J. Liang and K. Nahrstedt, "Dagstream: Locality aware and failure resilient peer-to-peer streaming," in *Multimedia Computing and Networking*, 2006.

[101] ——, "Randpeer: Membership management for qos sensitive peer-to-peer applications," in *INFOCOM*, 2006.

[102] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat, "Using random subsets to build scalable network services," in *USENIX Symposium on Internet Technologies and Systems*, 2003.

[103] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *SOSP*, 2003.

[104] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems." in *Multimedia Computing and Networking*, 2002.

[105] D. Tsoumakos and N. Roussopoulos., "A comparison of peer-to-peer search methods." in *Workshop on the Web and Databases*, 2003.

[106] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in *CIKM*, 2002.

[107] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *International Conference on Supercomputing*, 2002.

[108] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks." in *INFOCOM*, 2003.

[109] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM*, 2003.

[110] Z. Xu and Y. Hu, "Sbarc: A supernode based peer-to-peer file sharing system." *IEEE Symposium on Computers and Communications*, 2003.

[111] B. Zhao, Y. Duan, L. Huang, A. Joseph, and J. D. Kubiatowicz, "Brocade: Landmark routing on overlay networks," in *IPTPS*, 2002.

[112] D. Tsoumakos and N. Roussopoulos, "Adaptive probabilistic search for peer-to-peer networks," in *Conference on P2P Computing*, 2003.

[113] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *ICDCS*, 2002.

[114] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficienct conent location using interest-based locality in peer-to-peer systems," in *INFOCOM*, 2003.

[115] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman, "Search in power-law networks," *Physical Review E 64*, 2001.

[116] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *ICDCS*, 2002.

[117] Q. R. for the Gnutella Network, "www.limewire.com/ developer/query routing/keywordrouting.htm."

[118] M. Li, W. Lee, and A. Sivasubramaniam, "Semantic small world: An overlay network for peer-to-peer search," in *International Conference on Network Procotols*, 2004.

[119] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *International Workshop on Networked Group Communication*, nov 2001.

[120] I. Clarke, T. Hong, S. Miller, O. Sandberg, and B. Wiley, "Protecting free expression online with freenet," *IEEE Internet Computing*, 2002.

[121] A. Fiat and J. Saia, "Censorship resistant peer-to-peer content addressable networks," in *IPTPS*, 2002.

[122] M. Datar, "Butterflies and peer-to-peer networks," Tech. Rep., 2002.

[123] M. Castro, P. Druschel, Y. Hu, and A. Rowstro, "Exploiting network proximity in peer-to-peer overlay networks," Tech. Rep., 2002.

[124] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *PODC*, 2002.

[125] M. Junginger and Y. Lee, "The multi-ring technology-high performance group communication in peer-to-peer networks," in *International Conference on Peer-to-Peer Computing*, 2002.

[126] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *INFOCOM*, 2002.

[127] B. Loo, R. Huebsch, I. Stoica, and J. Hellerstein, "The case for a hybrid p2p search infrastructure," in *Workshop on Peer-to-Peer Systems*, 2004.

[128] W. Yiu, X. Jin, and S. Chan, "Challenges and approaches in large-scale peer-to-peer media streaming," *IEEE Multimedia*, 2007.

[129] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient overlays using multicast," in *ACM/IEEE Transactions on Networking*, 2005.

[130] C. Yeo, B. Lee, and M. Er, "A survey of application level multicast techniques," *Computer Communications,*, 2004.

[131] M. Hosseini, D. Ahmed, , S. Shirmohammadi, and N. Georganas, "A survey of application-layer multicast protocols," *Communications Surveys & Tutorials*, 2007.

[132] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "Almi: An application level multicast infrastructure," in *USNIX Symposium on Internet Technologies and Systems (USITS '01)*, 2001.

[133] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol," *RFC 1075, Internet Engineering Task Force*, 1988.

[134] S. Banerjee and B. Bhattacharjee, "A comparative study of application layer multicast protocols," Tech. Rep. Univ. Maryland, College Park, Technical Report, 2002.

[135] M. Kwon and S. Fahmy, "Topology-aware overlay networks for group," in *NOSSDAV*, 2002.

[136] V. Roca and A. El-Sayed, "A host-based multicast (hbm) solution for group communications," in *International Conference on Networking*, 2001.

[137] R. Douglas, "Np-completeness and degree restricted spanning trees," *Discrete Mathematics*, 1992.

[138] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over a peer-to-peer network," Tech. Rep., 2001.

[139] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr., "Overcast: Reliable multicasting with an overlay network," in *OSDI*, 2000.

[140] K. Sripanidkulchai, A. Ganjam, and B. Maggs, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," in *ACM SIGCOMM*, 2004.

[141] M. Guo and M. Ammar, "Scalable live video streaming to cooperative clients using time shifting and video patching," in *INFOCOM*, 2004.

[142] T. S. E. Ng, Y. Chu, S. G. Rao, K. Sripanidkulchai, and H. Zhang, "Measurement-based optimization techniques for bandwidth-demanding peer-to-peer systems," in *INFOCOM*, 2003.

[143] R. Carter and M. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *IEEE INFOCOM*, 1997.

[144] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar, "A novel server selection technique for improving the response time of a replicated service," in *IEEE INFOCOM*, 1998.

[145] K. Hanna, N. Natarajan, and B. Levine, "Evaluation of a novel two-step server selection metric," in *ICNP*, 2001.

[146] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," in *SIGCOMM*, 2000, pp. 283–294. [Online]. Available: citeseer.nj.nec.com/lai00measuring.html

[147] ——, "Nettimer: a tool for measuring bottleneck link bandwidth," in *USENIX Symposium on Internet Technologies and Systems*, 2001.

[148] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relationship with tcp throughput," in *ACM SIGCOMM*, 2002, reviewed.

[149] S. Savage, "Sting: A tcp-based network measurement tool," in *USENIX Symposium on Internet Technologies and Systems*, 1999.

[150] S. Shi, J. Turner, and M. Waldvogel, "Dimensioning server access bandwidth and multicast routing in overlay networks," in *ACM NOSSDAV*, 2001.

[151] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrustructure for real-time applications," in *INFOCOM*, 2003.

[152] Y. Zhong, S. Shirmohammadi, and A. E. Saddik, "Measurement of the effectiveness of application-layer multicasting," in *IEEE IMTC*, 2005.

[153] L. Mathy, R. Canonico, and D. Hutchison, "An overlay tree building control protocol," in *Workshop on Networked Group Communication*, 2001.

[154] G. Dn, V. Fodor, and I. Chatzidrossos, "On the performance of multiple-tree-based peer-to-peer live streaming," in *Infocom 2007 as short paper*, 2007.

[155] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," in *IEEE Journal of Selected Areas in Communication*, 2004.

[156] M. Adler, R. Kumar, K. W. Ross, D. Rubenstein, T. Suel, and D. D. Yao, "Optimal Peer Selection for P2P Downloading and Streaming," in *INFOCOM*, 2005.

[157] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2cast: Peer-to-peer patching scheme for vod service," in *World Wide Web Conference*, 2003.

[158] M. Yang and Z. Fe, "A proactive approach to reconstructing overlay multicast trees," in *INFOCOM*, 2004.

[159] A. Fei, J. Cui, M. Gerla, and D. Cavendish, "A dual-tree scheme for fault-tolerant multicast," in *IEEE International Conference on Communications*, 2001.

[160] W. Jia, W. Zhao, D. Xuan, and G. Xu, "An efficient fault-tolerant multicast routing protocol with core-based tree techniques," in *IEEE International Conference on Communications*, 1999.

[161] W. Wang, D. A. Helder, S. Jamin, and L. Zhang, "Overlay optimizations for end-host multicast," in *Workshop on Networked Group Communication*, 2002.

[162] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal of Selected Areas in Communication*, 2002. [Online]. Available: citeseer.nj.nec.com/castro02scribe.html

[163] S. Xie, B. Li, G. Keung, and X. Zhang, "Large scale peer-to-peer live vide streaming: Theory and practice," Tech. Rep., 2006.

[164] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of P2P Live Streaming Services," in *INFOCOM*, 2007.

[165] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R. Wang, "Overlay mesh construction using interleaved spanning trees," in *IEEE INFOCOM*, 2004.

[166] S. Annapureddy, C. Gkantsidis, P. Rodriquez, and L. Massoulie, "Providing video-on-demand using peer-to-peer networks," in *Internet Protocol TeleVision (IPTV) workshop in conjunction with WWW*, 2006.

[167] C. Dana, D. Li, D. Harrison, and C. Chuah, "Bass: Bittorrent assisted streaming system for video-on-demand," in *IEEE Signal Processing Soc. Workshop on Multimedia Signal Processing*, 2005.

[168] X. Jiang, Y. Dong, D. Xu, and B. Bhargava, "GnuStream: A P2P Media Streaming System Prototype," in *ICME*, 2003.

[169] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: Peer-to-peer media streaming using collectcast," in *ACM Multimedia*, 2003.

[170] P. Shah and J. Paris, "Peer-to-Peer Multimedia Streaming Using BitTorrent," in *IPCCC*, 2007.

[171] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev, "Collectcast: A peer-to-peer service for media streaming," *ACM/Springer Multimedia Systems Journal*, 2003.

[172] A. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *INFOCOM*, 2006.

[173] N. Magharei and R. Rejaie, "Dissecting the performance of Live Mesh-based P2P Streaming," Tech. Rep. CIS-TR-07-05, 2007. [Online]. Available: http://mirage.cs.uoregon.edu/pub/tr07-05.pdf

[174] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," in *IPTPS*, 2005.

[175] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang, "On the optimal scheduling for media streaming in data-driven overlay networks," in *GLOBECOM*, 2006.

[176] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "Bitos; enhancing bittorrent for supporting streaming applications," in *Global Internet Workshop*, 2006.

[177] G. Urvoy-Keller and P. Michiardi, "Impact of inner parameters and overlay structure on the performance of bittorrent," in *Global Internet Symposium*, 2006.

[178] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Scalable live streaming service based on inter-overlay optimization," in *INFOCOM*, 2006.

[179] M. Zhang, J. Luo, L. Zhao, and S. Yang, "A peer-to-peer network for live media streaming using a push-pull approach," in *ACM Multimedia*, 2007.

[180] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Transaction on Computer Systems*, 1999.

[181] J. Seibert, D. Zage, S. Fahmy, and C. Nita-Rotaru, "Experimental comparison of peer-to-peer streaming overlays: An application perspective," Tech. Rep., 2007.

[182] M. Bawa, H. Deshpande, and H. Garcia-Molina, "Transience of peers and streaming media," in *HotNets-I*, 2002.

[183] T. Silverston and O. Fourmaux, "Source vs data-driven approach for live p2p streaming," in *IEEE International Conference on Networking*, 2006.

[184] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "Measurement study of peer-to-peer file system sharing," in *Multimedia Computing and Networking*, jan 2002.

[185] E. Adar and B. Huberman, "Free riding on gnutella," *First Monday*, 2000.

[186] P. Golle, K. Leyton-Brown, and I. Mironov, "Incentives for sharing in peer-to-peer networks," in *ACM Electronic Commerce*, 2001.

[187] C. Buragohain, D. Agrawal, and S. Suri, "A game theoretic framework for incentives in p2p systems," in *IEEE P2P*, 2003.

[188] D. Turner and K. Ross, "The lightweight currency protocol," *Internet Draft*, 2003.

[189] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Scalable and robust incentive techniques for p2p networks," in *ACM Electronic Commerce*, 2004.

[190] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster, "To share or not to share: An analysis of incentives to contribute in collaborative file sharing environments," in *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[191] T. Ngan, D. Wallach, and P. Druschel, "Enforcing fair sharing of peer-to-peer resources," in *IEEE International Workshop on Peer-To-Peer Systems*, 2003.

[192] M. Feldman and J. Chuang, "Overcoming free-riding behavior in peer-to-peer systems," in *ACM SIGecom Exchanges*, 2005.

[193] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[194] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice, "Dissecting bittorrent: Five months in a torrents lifetime," in *Passive Analysis and Measurement Workshop*, 2004.

[195] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *Workshop on Peer-to-Peer Systems*, 2005.

[196] R. Thommes and M. Coates, "Bittorrent fairness: analysis and improvements," in *Internet, Telecom. and Signal Processing*, 2005.

[197] Y. Sung, M. Bishop, and S. Rao, "Enabling Contribution Awareness in an Overlay Broadcasting System," in *ACM SIGCOMM*, 2006.

[198] N. Magharei, R. Rejaie, and Y. Guo, "Incorporating contribution-awareness into mesh-based peer-to-peer streaming services," Tech. Rep., 2007.

[199] Y. Chu, J. Chuang, and H. Zhang, "A case for taxation in peer-to-peer streaming broadcast," in *PINS, ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, 2004.

[200] Y. Zhu, B. Li, and J. Guo, "Multicast with network coding in application-layer overlay networks," *IEEE Journal of Selected Areas in Communication*, 2004.

[201] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network information flow," *IEEE TRANSACTIONS ON INFORMATION THEORY*, 2000.

[202] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Allerton Conf. Communication, Control and Computing*, 2003.

[203] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *IEEE International Symposium on Information Theory*, 2003.

[204] Z. Li, B. Li, and L. Lau, "On achieving maximum information flow rates in undirected networks," *Joint Special Issue on Networking and Information Theory*, 2006.

[205] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *INFOCOM*, 2007.

[206] Wikipedia, "Network coding, http://en.wikipedia.org/wiki/network_coding." [Online]. Available: http://en.wikipedia.org/wiki/Network_coding

[207] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive view of a live network coding p2p system," in *ACM/USENIX Internet Measurement Conference*, 2006.

[208] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *INFOCOM*, 2005.

[209] M. Wang and B. Li, "How practical is network coding?" in *EEE International Workshop on Quality of Service (IWQoS)*, 2006.