Peer-assisted Media Streaming: a Holistic Review

Yuan Feng[1] and Baochun Li[1]

Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road
Toronto, ON, M5S 3G4
Canada
{yfeng,bli}@eecg.toronto.edu

Summary. This chapter presents a holistic review of recent research advances in peer-assisted streaming systems, including both live and on-demand streaming. We approach this task by first presenting design objectives of streaming systems in general, and then discuss differences between live and on-demand streaming. These common and different design objectives motivate the protocol design space in streaming systems, in categories of peer selection, segment scheduling, and distributed caching protocols. We present main results from the existing literature in each of these dimensions, with a particular focus on the pivotal role of network coding within such a protocol design space. We conclude the chapter with an outlook towards future research directions, especially in the application of network coding in peer-assisted streaming systems.

1 Introduction

To meet the demand of explosively growing multimedia applications, media streaming over the Internet has been a research topic attracting substantial interests in the literature over the past two decades. The "holy grail" of Internet media streaming is to satisfy the media streaming needs of as many end users as possible, with sustainable server bandwidth costs. The traditional client/server architecture advocates the use of large data centers to sustain streaming to end users at a large scale.

To maintain a satisfactory user experience, the bandwidth costs on servers grow rapidly as the user population scales up, and may not be bearable in corporations with limited resources. New architectural designs in the past two decades, such as IP multicast and content delivery networks (CDNs), attempted to address the problem by conserving resources in the edge or core routers, or by load balancing across a large number of edge servers that are geographically distributed. However, the problem of scalability to a large user

population in media streaming systems is only mitigated to a certain degree, not solved.

Peer-to-peer (P2P) networks propose a different architectural design perspective: it offloads part of the bandwidth burden from dedicated streaming servers hosted by the content providers, and shifts them to end hosts themselves when they serve content to each other. The total volume of bandwidth consumed is not reduced, and in most cases, is even increased due to overhead of protocol-induced messaging and redundancy. Peer-to-peer networks are not panacea: the architecture does not *replace* streaming servers — they simply *complement* them. Such an architectural paradigm is referred to as *peer-assisted media streaming* in this chapter, to highlight the complementary nature of peer-to-peer networks.

Existing peer-assisted media streaming systems can be further divided into two categories, *live* and *on-demand* media streaming, with the latter often referred to as *video-on-demand* (VoD) in the literature. At a high level, a peer-assisted media streaming system typically contains three elements: (1) a number of dedicated streaming servers that serve media content, and can be treated in an aggregate fashion as a single dedicated media source; (2) one or a small number of index servers that keeps track of state information of the system, such as existing end users (or *peers*); and (3) a web portal that provides information about media channels. An example of such a system is shown in Fig. 1.

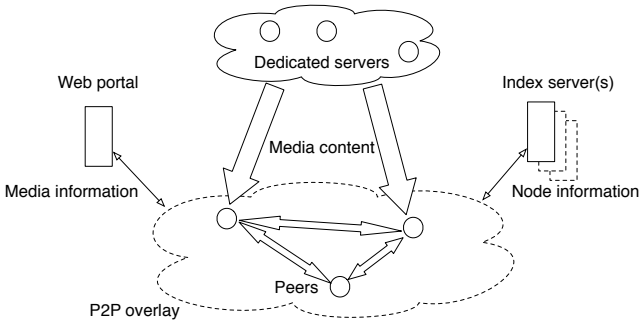

Fig. 1. The architecture of peer-assisted streaming systems.

Fundamentally, architectural and protocol designs in peer-assisted media streaming systems need to successfully address three characteristics observed in real-world streaming systems:

- *Scale:* A real-world peer-assisted streaming system needs to support at least hundreds of thousands of users and thousands of media channels simultaneously, with a reasonable playback quality to each of the end users. The CBS broadcast of NCAA tournament in March 2006 [26] has peaked

at 268,000 simultaneous viewers. UUSee Inc. experienced a flash crowd of 871,000 peers on the Chinese New Year Eve in 2007 [35].

- ▷ *Dynamics:* Real-world peer-assisted streaming systems have shown a high level of dynamics. End users, as peers who serve media content to one another, may join or leave a media channel or the system at any time. Network bandwidth between a pair of peers vary over time, sometimes quite drastically due to the shifting of bandwidth bottlenecks. Such dynamics challenge the design of any system architecture or protocol. Recent measurement studies have clearly shown that peer dynamics represent one of the most influential factors that affect the overall performance of peer-assisted streaming systems [22].
- ▷ *Heterogeneity:* According to a large set of live streaming traces obtained from the Coolstreaming system, it has been discovered that there exists a highly skewed distribution with respect to upload bandwidth contributions from peers. Such heterogeneity may have significant implications on resource allocation in peer-assisted streaming systems.

This chapter provides a taxonomy of recent research achievements in peer-assisted media streaming systems, with a special focus on on-demand streaming systems (VoD). We start our discussion with an in-depth coverage of common challenges in both live and on-demand media streaming systems. We then shift our focus to specific challenges and their corresponding solutions in on-demand streaming systems. In particular, we point out that *network coding* may be a feasible and practical solution to improve streaming performance in peer-assisted streaming systems. Finally, we present an outlook to future research directions in peer-assisted streaming systems.

2 Fundamental Challenges in Peer-Assisted Streaming Systems

In peer-assisted streaming systems that include both live and on-demand streaming, dedicated streaming servers and end users (peers) are organized into a topology at any given time, so that media content can be broadcast from the servers to all the participating peers in a streaming channel. In such a topology, a peer is directly connected with a subset of other peers in the same channel, referred to as its *neighbors*. The topology that peers and servers organize into is sometimes referred to as a streaming *overlay*. In this section, we present two fundamental challenges and their corresponding solutions in such a streaming overlay for a media channel, common to both live and on-demand streaming systems.

In order to allow peers to serve one another and to maximize its efficiency, the following challenges naturally arise. First, which subset of peers should become neighbors of a participating peer? Second, when being served by neighbors, which segment of the media stream should be served first, and

thus be given priority? Solutions to the first question are referred to as *neighbor selection* (or *overlay construction*) algorithms, and those to the second are referred to as *segment scheduling* algorithms.

2.1 Neighbor Selection Algorithms

One of the important design goals in peer-assisted media streaming systems is load balancing, *i.e.,* the redistribution of content should be balanced on the participating peers so that the costs for the system are shared, much of the existing research literature focused on designing suitable neighbor selection algorithms. We present several examples of this category of algorithms.

With an inter-overlay optimization scheme, Anysee, proposed by Liao *et al.* [25], uses an *inter-overlay manager* to find appropriate streaming paths with low delays in the peer-assisted streaming system. The main task of the inter-overlay manager is to maintain a backup streaming path set and an active streaming path set, using a reverse tracking algorithm. When the size of the backup set is less than a threshold, the peer will send out a message to some of its neighbors, which will be relayed till the receiver finds the delay from the initial peer to the current peer is greater than the minimum of all source-to-end delays from the current peer to the streaming source on different paths. Using such an algorithm, a peer is able to construct the best overlay path accordingly. When the total bit rates from active streaming paths are lower than a certain threshold, the manager will check whether a better path should be activated to replace the current one, selected from the backup streaming path set.

An alternative approach is to use a score-based neighbor selection algorithm, where a score is assigned as the criteria to decide which set of peers are selected as neighbors. OCTOPUS [24], for example, adopts a *score* that represents the degree of data overlapping between any two neighbors, *i.e.*, the amount of time slots it can provide to its neighbor and the amount of time slots the neighbor can provide to the peer. Under the assumption that underlying media streaming system provides the functionality of a Distributed Hash Table (DHT), Graffi *et al.* [17] propose another scoring function that calculates the costs for choosing a specific peer by taking into account of peer characteristics such as the uptime, network condition, and the number of tasks already performed for the system. As identified peers for a specific block may randomly leave the system, peer churn may become a critical challenge for this partner selection algorithm.

All of the aforementioned algorithms are under the assumption that users can and are willing to collaborate, which is not always the case realistically. Measurement studies have shown that, in some peer-to-peer streaming systems, a small set of nodes are requested to contribute 10 to 35 times more uploading bandwidth than downloading bandwidth [2]. As a result, peers are not willing to voluntarily contribute their own upload bandwidth, which may seriously affect the overall performance of peer-assisted streaming systems.

Therefore, an appropriate incentive mechanism is critical to the performance of a peer-assisted streaming system, and the design of neighbor selection algorithms offers an opportunity to tackle this challenge.

Silverston *et al.* [30] discovers that the incentive mechanism in BitTorrent file sharing systems, called *tit-for-tat*, is not well suited to peer-assisted streaming systems. Media flows impose temporal constraints that do not exist in bulk content distribution. Sometimes peers cannot serve data in return, not because they are non-cooperative, but because the temporal constraints make it pointless. As such, the temporal nature of the content makes the incentive mechanisms of BitTorrent obsolete. Existing solutions are mainly score-based, in which each peer is a credit entity, and peers will allocate resources according to the credits of requesting peers to maximize their own credits. However, these mechanisms suffer from a high computational complexity that prevent their real-world implementation. The design of a scalable, light-weighted incentive mechanism that can be incorporated into peer-assisted streaming systems remains to be an open problem.

Finally, recent measurement studies have unveiled that there is a highly unbalanced distribution with respect to uploading contributions from peers, which has significant implications on the resource allocation in such a system [22]. Further, by investigating the distribution of inter-peer bandwidth in various peer ISP categories, Wu *et al.* [35] shows that the ISPs that peers belong to are more correlated to inter-peer bandwidth than their geographic locations. In addition, there exist excellent linear correlations between peer last-mile bandwidth availability and inter-peer bandwidth within the same ISP, or between a subset of ISPs. The joint consideration of highly skewed resource distribution, incentive awareness and inter-peer bandwidth interference may lead to more appropriate neighbor selection algorithms in future research.

2.2 Segment Scheduling Algorithms

Different from other P2P applications such as file sharing, peer-assisted streaming systems need to consider timeliness requirements of streaming media. If media segments do not arrive in a timely fashion, they have to be skipped at playback, which degrades the playback quality, which is one of the most important performance metrics in streaming systems. Experiments show that peer-assisted streaming systems do not perform congestion control correctly, in scenarios where peer access links get congested [1]. Their response is to increase the level of redundancy, which further increases the download rate, and further exacerbate the congestion situation. These observations call for a judicious design of segment scheduling algorithms.

The first strategy for segment scheduling in peer-assisted streaming systems is a *tree-based per-slice push* strategy, in which a media stream is first divided into substreams as its slices, and then each of these slices is pushed to downstream peers in a tree. Though tree-based strategies offer minimal

source-to-peer delays (the delay between the occurrence of a live event in a live stream and its playback at a peer), it suffers from the complexity in maintaining such tree structures when peers arrive and depart frequently. In contrast, a *mesh-based per-segment pull* strategy, first proposed in Zhang *et al.* [39], has been implemented in most real-world streaming systems. In such a strategy, the media stream is divided into segments in the time domain, and peers make explicit requests for the desired media segment from its neighboring peers. Neighboring peers exchange buffer availability bitmaps periodically. This strategy is simple to implement, but increases the source-to-peer delay in live streaming systems, due to the need for periodic buffer availability exchanges. The source-to-peer delay, however, is not an issue for on-demand streaming systems.

Zhang *et al.* [38] has proposed that push and pull strategies be combined, so that a hybrid strategy can be designed to be both robust to peer departures, and to support smaller source-to-peer delays. In the proposed push-pull hybrid strategy, segments are pulled in the first time slot when a peer first joins the system, and then pushed directly by the neighbors afterwards. To achieve this, the stream is evenly partitioned into n sub-streams, and each sub-stream is composed of the packets whose sequence numbers are congruent to the same value modulo n. Every continuous n packets belong to different sub-streams separately are grouped into a *packet group* and every continuous g packet groups are further clustered into a *packet party*. Each packet group in a packet party is numbered from 0 to $g-1$ and hence the packet group number can be computed by $\lfloor s/n \rfloor \bmod g$, where s is the packet sequence number. An example is shown in Fig. 2, where the stream is partitioned into 3 sub-streams and $n = 3, g = 3$.

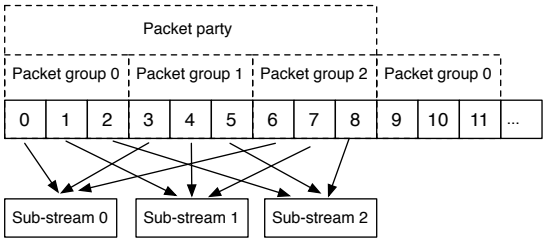

Fig. 2. Sub-streams in a push-pull hybrid strategy.

Once a packet in packet group 0 in one packet party is requested successfully from a neighbor, the peer will sent this neighbor a *sub stream subscription* to let it directly push the remaining packets in the same sub-stream. There are two types of streaming packets — the pulled packets and the pushed packets. When over 95% packets are pushed directly from the neighbors, the node will stop requesting buffer maps. And once the delivery ratio drops below 95% or a neighbor who provides over 5% streaming data quits, the peer will start to

request buffer maps and pull streaming packets from its neighbors. Thus, most of the packets received will be pushed packets from the second time interval.

Zhou *et al.* [40] form a stochastic model to analyze some widely used segment selection strategies, including the *greedy*, *rarest first*, and *mixed* strategies. With M peers in the system, each peer maintains a buffer B that can cache up to n segments received from the system, as shown in Fig. 3.

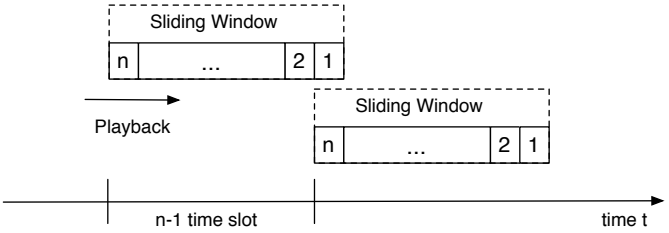

Fig. 3. The sliding window of buffer B.

The *greedy* strategy intends to fill the empty buffer locations closest to the playback point first. The segment selection function for the greedy strategy, $s(i)$, which is the probability of selecting $B(i)$, can be expressed as follows:

$$s(i) = 1 - (p(n) - p(i+1)) - p(1), i = 1, ..., n-1, \tag{1}$$

where the term $p(n) - p(i+1)$ is the probability that any particular segment is downloaded into buffer positions between $B(n)$ to $B(i+1)$; and the term $p(1)$ is the probability that any particular segment is downloaded directly from the server. The *rarest first* strategy will select a segment that has the fewest number of replicas in the system, for which the segment selection function is:

$$s(i) = 1 - p(i), \tag{2}$$

where $p(i)$ represents the probability that any particular segment is downloaded into buffer positions between $B(1)$ to $B(i-1)$. A *mixed* strategy implies that the the rarest first strategy is used for a portion of the buffer $B(1), ..., B(m), 1 \leq m \leq n$. If no segment can be downloaded using the rarest first strategy, the greedy strategy is then used for the remainder of the buffer $B(m+1), ..., B(n)$. In this case, the segment selection function for the mixed strategy can be expressed as:

$$p(1) = 1/M \tag{3}$$

$$p(i+1) = p(i) + p(i)(1 - p(i))^2, i = 1, ..., m-1. \tag{4}$$

Simulation results in [40] have shown that the rarest first strategy is much better with respect to scalability, whereas the greedy strategy is able to produce better playback performance (continuity) in small scale systems.

While analytical results have indicated that a sequential selection strategy is well-suited for use in peer-assisted media streaming systems, many choose to implement a hybrid segment selection algorithm, which divides the cache into a high-priority set and the remaining set according to their time to the playback deadline, as indicated in Fig. 4. Intuitively, a peer is likely to choose segments in the high-priority set with higher probability for downloading [28, 14]. Much effort has been devoted to design the appropriate distribution in these two sets. For example, the piece picking policy in Toast performs in-order selection in high-priority set and Beta random selection in remaining set, which is implemented using Python's generator for a beta distribution in remaining set. It adds a given-up area between the current streaming position and high-priority region, which corresponds to pieces that are too close to the current stream position to download on time. Simulation results show that the performance beats rarest-first and in-order; however, the simulation is only conducted in static case in which peers all are assumed to stay until the end of a video, which is less likely to happen in the real world. Then in R^2, an uniform distribution is used in high-priority set and a randomized choice subjected to Weibull distribution is adopted in the remaining set.

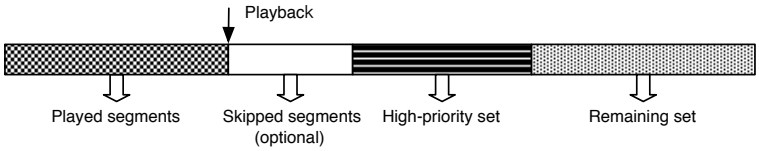

Fig. 4. Hybrid segment scheduling algorithms

Combined with network coding, Annapureddy *et al.* propose a worst-seeded-policy, which is to upload a block from a lesser represented segment whenever possible [5, 4]. Compared with greedy approach that peers greedily request blocks from their earliest incomplete segments, system throughput increases significantly. However, the assumption of source node has knowledge of the rarity of segments aggregated across the other nodes in the network is not realistic for the real world. They also examine the effect of pre-fetching, i.e. fetching a block that is needed later than the first segment of interest. By evaluating probabilistic-first-range-random-second-range-rarest policy, which performs consistently well in pre-fetching policies, it is shown that pre-fetching provides easy access to blocks when they are needed by creating additional sources for the blocks.

However, most papers discuss the situation of only one or two channels exist, which is definitely an over-simplified case. The peer-assisted streaming systems in nowadays like PPStream, PPLive and so on have hundreds of

channels available. From the server's point of view, we have to consider how to allocate bandwidth in multi-channel streaming systems. In a server capacity provisioning algorithm—Ration, which is proposed by Wu *et al.* [36], server capacity allocated to each channel in next time slot is proactively computed by an optimization problem *Provision*$(t+1)$, as follows $(\forall t = 1, 2, ...)$:

$$\begin{aligned} \text{maximize} \quad & \sum_{c \in C} p^c n_{t+1}^c q_{t+1}^c \\ \text{subject to} \quad & \sum_{c \in C} s_{t+1}^c \leq U, \\ & q_{t+1}^c = F_{t+1}^c(s_{t+1}^c, n_{t+1}^c), \forall c \in C, \\ & 0 \leq q_{t+1}^c \leq 1, s_{t+1}^c \geq 0, \forall c \in C \end{aligned} \tag{5}$$

where p^c denotes the assigned priority level, n_c represents the estimated population and q^c the streaming quality for each channel c. At each time t, Ration proactively computes the amount of server capacity s_{t+1}^c to be allocated to each channel c for time $t+1$. With full ISP awareness, Ration is carried out on a per-ISP base, and is able to guide the deployment of server capacities and channels in each ISP to maximally constrain P2P traffic inside ISP boundaries. Cheng *et al.* also evaluate the improvement of multi-video caching (MVC) [12]. They show that for any given number of peers, MVC further reduces the aggregate load on the sources and MVC has lower chunk cost for the same concurrency compared with single-video caching. Further, it is presented that MVC increases chunk replica counts across the system which improves the continuity by decreasing jitter and seek latencies. In general, designing peer-assisted streaming systems with multiple channels is more complicated and challenging, which may raise the interests of researchers in the future.

3 Special Challenges in P2P VoD streaming systems

While peer-assisted live streaming appears to be evolving towards a mature stage, VoD services based on P2P overlays have appeared as a more attractive area to most researchers. A peer-assisted VoD service is more challenging to design than a live streaming system because the peers in a live streaming scenario have a shared temporal focus, while in VoD case they have greater temporal diversity of requests. The peer dynamics resemble those of file downloading, while still requiring low startup delays for the sequential playback of large media objects. Besides, equipped with larger storage space—cache—than the limited buffer in live streaming scenario, peers in VoD case can store and further upload more content. Intuitively, questions will arise as in case of larger cache, what content should be chosen to store in each peer. The fact that different users may be watching different parts of the video at any time can greatly impact the efficiency of a swarming protocol. The lack of synchronization among users reduces the block sharing opportunities and increases

the complexity of the block transmission algorithms. This naturally bring the problem of how to search for the desired content efficiently, which results in numerous searching algorithms.

3.1 Cache Management

With larger storage capacities, peers can store redundant copies of a file near the end user and it has been proven to be extremely effective in many P2P VoD applications. Recent measurement results show that a simple static cache that stores the top long-term popular files uses 84% less space than a dynamic infinite cache solution which stores all videos, and still manages to save about 75% of the load in the server [9]. As a result, a number of algorithms that intend to choose the most "popular" content wisely to store in a finite storage space are proposed, a summary of which is shown in Table 1.

Table 1. Cache management in peer-assisted VoD systems

Topics	[31]	[3]	[11]	[27]	[21]	[18]	[37]
Popularity Prediction	×	×	×	√	√	√	√
Peer Availability Detection	×	×	×	√	×	×	×
Replication Mechanism	√	√	√	×	×	√	√
Bandwidth Allocation	×	×	×	×	√	×	×
Dynamic Repair	×	×	√	×	×	×	×

Popularity Prediction

In order to minimize VoD request rejection rates for a very large content library, one of the key question is how to translate the popularity distribution into content availability in the network. The common way of doing so is to use number of requests for a specific video content to represent its popularity and the number of replicas in the system to reflect its availability. Through numerical simulations and empirical validations, the popularity distribution of relatively popular video categories exhibits power-law behavior (a strait line in a log-log plot) with a tail truncation, which is affected by both the average requests per user and the number of videos in a category. While the popularity distribution of the unpopular content behaves more like a Zipf's law, later research shows that despite the traffic variation among different days, the popularity distributions are quite similar and the distribution is more skewed

than a Zipf distribution [9], which may due to the fact that on any given day, there are several highly popular news and business clips, with each of these clips having roughly the same popularity. Using these distributions, we can use the video's past behavior to predict their future more precisely.

Since the popularity of a video changes during a typical day and the change between two continuous short intervals is smooth, the popularity in the most recent interval is useful to predict the future requests. In addition, chunks requested recently are more likely to be requested later. Based on these observations, a chunk request predictor is constructed weighting the contribution of recent requests to decay with time. Then by discovering the fact that for the same popularity level and content availability, the VoD rejection rate for short content will be much lower than that for long content, a popularity correction method is proposed for that based on Zipf distribution, it further takes into consideration of the expected average number of simultaneous sessions (ESS) per content, which depends on both content duration and average number of requests received for this content. Thus the corrected popularity can be expressed as $P_i' = P_i \cdot (1 + ESS_i \cdot \alpha)$, where α in introduced to study how much weight the ESS parameter should be given as a large α means popular titles would be more available in the network with more copies spread around the network [27]. Guo *et al.* define the popularity of a segment as

$$p = \frac{\frac{S_{sum}}{S_0}}{T_r - T_0} \times \min\left(1, \frac{\frac{T_r - T_0}{n}}{t - T_r}\right), \tag{6}$$

where t is the current time instant, $\frac{\frac{S_{sum}}{S_0}}{T_r - T_0}$ represents the average access rate of the segment in the past, normalized by the segment size, and $\min(1, \frac{\frac{T_r - T_0}{n}}{t - T_r})$ reflects the probability of future access: $\frac{T_r - T_0}{n}$ is the average time interval of accesses in the past; if $t - T_r > \frac{T_r - T_0}{n}$, the possibility that a new request arrives is small; otherwise, it is highly possible a request is coming soon [18].

Peer Availability Detection

To ensure the maintenance of desired number of replicas of a video in the system, we should detect the availability of the peers. Due to the extensive influence of user interaction, the detection faces two requirements. First, the system must empirically measure the short-term availability distribution of its host population on an ongoing basis. Second is the non-transient failures that take stored data out of the system for infinite periods [27]. The availability monitor (AM) tracks host availability, maintains host availability metrics which are short-term host availability and long-term decay rate, and notifies the redundancy engine when the system reaches availability thresholds that trigger repair. Through observation of real systems, it is shown that online time is a predictor of peer departure, which can be used for a simple peer

departure predictor. A peer is predicted to leave if it has been online for fewer than a certain period of time. Otherwise, it is predicted to stay in the system.

Replication Mechanism

The Replication mechanisms consists of redundancy computation and replace algorithm, which we will discuss below consequently.

▷ *Redundancy computation*: With sufficient redundancy across many peers, at any moment enough peers will be in the system to make a given data item available with high probability. However, it is not trivially clear how much redundancy is necessary for a given level of availability or what redundancy mechanism is mot appropriate for a given context. The simplest form of redundancy is pure replication. Given a target level of availability A and a mean host availability of μ_H, the number of required replicas c can be calculated directly.

$$A = 1 - (1 - \mu_H)^c \tag{7}$$

$$c = \frac{\log(1 - A)}{\log(1 - \mu_H)} \tag{8}$$

However, it can be highly inefficient in low-availability environments caused by peer churn since many storage replicas are required to tolerate potential transient failures. An alternative way is to use erasure coding. Given the number of blocks in a file b, and the stretch factor c specifying the erasure code's redundancy, the delivered availability can be expressed as:

$$A = \sum_{j=b}^{cb} \binom{cb}{j} \mu_{H^j} (1 - \mu_H)^{(cb-j)} \tag{9}$$

$$c = \left(\frac{k\sqrt{\frac{\mu_H(1-\mu_H)}{b}} + \sqrt{\frac{k_2\mu_H(1-\mu_H)}{b} + 4\mu_H}}{2\mu_H} \right)^2 \tag{10}$$

However, the price for this efficiency is a quadratic coding time and a requirement that reads and writes require an operation on the entire object [7].

▷ *Replace algorithm*: We then look at the different cache strategies to manage the cached content. In general, instead of depending on peers to manually decide how much cache space is shared and which video to be kept to share or not, peer-assisted VoD systems require peers provide cache space and let the cache management algorithm to manage cache. Peers may be asked to cache media content they are not watching at the moment or even not planning to watch in the future but will be beneficial from the global point of view. The simplest one is *Least Recently Used* (LRU) strategy [3], which maintains a queue of each file sorted by when it was last accessed. When a file is accessed, it is located in the queue, updated, and moved to the front. If it is not in the

cache already, it is added immediately. When the cache is full the program at the end of the queue is discarded. Another one is *Least Frequently Used* (LFU) strategy. To compute the cache contents, the index server keeps a history of all events that occur within the last N hours (where N is a parameter to the algorithm). It calculates the number of accesses for each program in this history. Items that are accessed the most frequently are stored in the cache, with ties being resolved using an LRU strategy.

Under the belief of the greater the number of peers with video cache, the higher the probability that the video is available, and the higher the effective bandwidth, Ying *et al.* propose a cache replacement strategy by introducing *cache_needed_rank* [37] by taking account both the video popularity and the number of peers that currently are caching that video, which can be expressed as:

$$cache\_needed\_rank = \frac{\text{the number of peers demanding}}{\text{the number of online peers with cache}}. \tag{11}$$

This cache replacement strategy is based on the this evaluation metric, where stream with a lower *cache_needed_rank* is replaced by one with a higher *cache_needed_rank*.

To replace both those media segments with diminishing popularities as they rarely get accessed and those popular media segments with too many copies being cached, Guo *et al.* [18] introduce another cache replacement policy. To achieve the optimal distribution, they define the *utility function* of a segment as

$$u = \frac{(f(p) - f(p_{\min})) \times (f(p_{\max}) - f(p))}{r^{\alpha+\beta}}, \tag{12}$$

where p represents the popularity of the segment, $p_{\min}$ and $p_{\max}$ estimate the minimum and maximum of segment popularities in the system respectively. r is the number of replicas of this segment and $f(p)$ is a monotonic nondecreasing function, which is called the *adjustment factor* of the utility function. The term $\frac{(f(p)-f(p_{\min})}{r^{\alpha}}$ captures the segments with small popularities and large number of replicas while $\frac{f(p_{\max})-f(p)}{r^{\beta}}$ captures the segments with large popularities and large number of replicas. These two kinds of segment replicas should be replaced by the replicas of segments with moderate popularities but a small number of copies. As a result, segments with the smallest utility value are chosen to be replaced when a peer's cache is full.

Vratonjić *et al.* present another cache management algorithm, which is used in their newly proposed system—BulletMedia [31]. This algorithm examines the problem to ensure all content blocks are replicated at least k times system-wide, where a typical value of k is 4. To achieve this, a peer examines its local content cache and determines the set of chunkIds for chunks it is currently not replicating. It then selects chunkIds at random from this set, and performs a lookup in the DHT. The DHT simply includes a count of the number of peers replicating the chunk in the response. If the number of replicas is below a per-defined level (e.g. 4), then the peer begins to retrieve the

blocks associated with the chunk. If not, the peer chooses another chunkId at random and queries it. This process proceeds until a chunk is found that requires more replications.

Bandwidth Allocation Policy for Prefetching

When peers have surplus upload capacity, it can be used to prefetch content for the future use, which has been proved to leverage the server load dramatically [21]. Then how to allocate the instantaneous surplus upload capacity among the peers in the system becomes a critical question. One of the allocation scheme is *water-leveling policy*, which aims to equalize the reservoir levels of perfected content across all the peers. Once the reservoir level of all the peers reach the same level, an attempt is made to equally distribute the surplus capacity among all the peers. Another one being considered is *greedy policy*, where each user simply dedicates its remaining upload bandwidth to the next user right after itself. Through simulation, it is observed that the greedy policy does slightly better than the water-leveling policy and both of them are near optimal.

Dynamic Repair Policy

Over longer periods, peers leave the system permanently, which requests the system to "repair" this lost redundancy by continuously writing additional redundant data onto new peers [7]. The two key parameters in repairing file data are the degree of redundancy used to tolerate availability transients and how quickly the system reacts to peer departures. In general, the more redundancy used to store file data, the longer the system can delay before reacting to peer departures. The repair policy spectrum can be defined in terms of two categories: eager and lazy.

Eager repair means the system immediately repairs the loss of redundant data when a peer fails. Using this policy, data only becomes unavailable when peers fail more quickly than they can be detected and repaired. The primary advantage of eager repair is its simplicity. Every time a peer departs, the system only needs to place redundant data on another peer in reaction. Moreover, detecting peer failure can be implemented in a completely distributed fashion since it isn't necessary to coordinate information about which peers have failed. However, the eager policy makes no distinction between permanent departures that require repair and transient disconnections that do not. Consequently, in public peer-to-peer environments, many repair actions may be redundant and wasteful.

Lazy repair tries to defer immediate repair and use additional redundancy to mask and tolerate peer departures for an extended period. The key advantage of lazy repair is that, by delaying action, it can eliminate the overhead of redundant repairs and only introduce new redundant blocks when availability is threatened. However, lazy repair also has disadvantages. In particular, it

must explicitly track the availability of individual peers and what data they carry. This is necessary to determine when an object's availability is threatened and a repair should be initiated. Consequently, the system must maintain explicit metadata about which peers hold what data.

Cheng *et al.* propose and evaluate a framework for lazy replication in GridCast, a P2P VoD system [11]. Their algorithm uses a peer departure predictor to choose when to replicate (just before a peer leaves) and to whom (peers that are predicted to be most unlikely to leave) and uses a chunk request predictor to choose what to replicate (the chunks that will be most popular in the coming sessions). They use a lazy factor α to control the upload used for replication. Then through simulation evaluation, though eager replication reduces more departure missed than lazy replication, the total number of replicated chunks in lazy case is 45% of that in eager algorithm and the cost is much lower. Further, the efficiency of lazy algorithm more than three times that of eager replication, which suggests that lazy replication can replicate more efficiently than eager replication.

Interestingly, almost all the existing articles on peer-assisted VoD systems are under the assumption of small cache or even no cache. Each peer in the network is only equipped with a larger buffer that is able to store a part of the media the user is watching currently, which requires a wise cache management mechanism to make better use of the limited storage space. With the development of hard devices, every peer can have a large storage space that is able to store not only the current media, but also the other media it has watched a certain time ago, at a reasonable price. With the consideration of large cache, the desired content can be found either on the peers who are watching the same media stream at this time or some "offline" peers who are watching something else at the moment. Then the challenging task of designing more efficient content lookup algorithms that are capable of detecting offline peers with the interested content is desired. What is more, from peers' point of view, how to allocate the limited upload bandwidth to contribute to both the currently watching media and the stored content might be an interesting research topic in the future. Till now, to the best of our knowledge, only [20] considers the hybrid-forwarding architecture which integrates both buffer-forwarding approach and storage-forwarding approach, as shown in Fig. 5. And they try to maximize the throughput by solving an optimization problem.

3.2 Content Searching Algorithm

For peer-assisted VoD systems, one salient additional characteristic is the allowance of user interaction with video streams. Examples include pause, fast-forward, back-forward, random seek operations and so on, which further lead to more complicated system design. To address the issue of the unpredictability of these user interactions, different kinds of searching algorithms

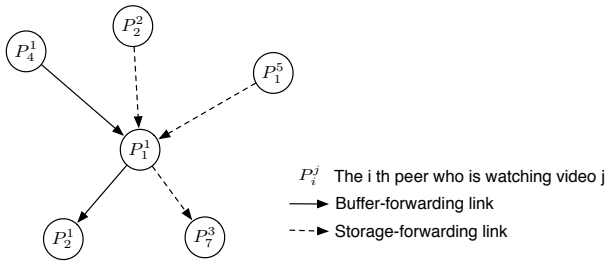

Fig. 5. A hybrid-forwarding P2P VoD system

aiming at finding the desired content efficiently with shorter initial buffering delay have been proposed.

Used to get the optimal match between clients and servers, Shaking is a searching algorithm proposed to find the server that can provide the fastest service [8]. Since the server schedules the pending requests in a FIFO manner, a shorter waiting time can be achieved by trying to move the requests that arrived earlier at the server to other servers. That is, given a set of server candidates, a client can contact them for their pending requests and try to find each of them a new server. The key idea can be found in Fig. 6.

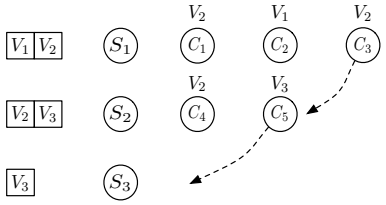

Fig. 6. Shaking algorithm

A client C trying to shake request $[C_3, V_2]$ out of server S_1 may find server, S_2 has V_2. Although this server cannot serve $[C_3, V_2]$ earlier than S_1, the client can try to see if it can shake any pending requests out of S_2. For example, it may launch a search for V_3 and find server S_3, which is free of workload at this moment. It shows that in order to successfully move out a request, a client may need to shake a chain of servers. To address the challenge of chaining effect, Shaking uses a 3-step solution, building closure set, shaking closure set and executing shaking plan. Further, Shaking makes it possible to for a client to be served by a server that is beyond the client's search scope.

RINDY, which is proposed by Cheng *et al.*, is a ring-assisted overlay topology in P2P VoD systems for efficient content lookup [10]. Under this scheme, a peer only needs $O(\log(T/w))$ hops (where T is the total time length of the video stream and w is the buffer window size of peers) to identify and connect to a new group of peers close to the destination playing positions when

a random seek occurs. In RINDY, each peer organizes all its neighbors into a series of concentric and non-overlapping logical rings according to their relative distances calculated from the current playing positions. The rings are separated into two types, *gossip-ring* and *skip-ring*. A gossip-ring is a peer's innermost ring, which is mainly responsible for collecting some near neighbors with close playing positions and overlapped buffer windows. And all the outer rings, which are mainly used to improve the speed of lookup operations and reduce the load of the tracker server are called skip-ring. When a peer seeks to a target position d, it first checks whether d is within its gossip-ring. If so, it performs a local search to find enough near-neighbors (neighbors in the gossip-ring) and far-neighbors (neighbors in the skip-ring). Otherwise, it finds the closest neighbor to d as its next hop, pushes its address into the *routing_path* and *result_set* fields of the query message, and finally forwards this query to that nest hop neighbor. Upon receipt of this query, the next-hop neighbor executes the same procedure, which will be iterated until this request arrives at some peer whose gossip-ring covers d. The final peer adds all of its near-neighbors into the *result_set* field of the query message and returns the message to the source peer along the routing path. Then the source peer adds all the members of *result_set* into its mCache and then change its current playing position to d.

Inspired by the fact that if a peer's buffer range is fully covered by other peers, removing it from the index structure will not cause much trouble as the peers who use this removed peer as a partner can still find other possible partners, Chi *et al.* introduce Buffer Assisted Search (BAS) to select as few index peers as possible without reducing search effectiveness, i.e., the total buffer coverage [13]. The problem can be formulated as the Minimum Buffer Cover (MBC) problem that can be solved using a distributed algorithm. The basic flow is to divide the existing index peers into two groups according to whether they overlap with the newcomer, and then apply the dynamic programming algorithm to the newcomer plus the group overlapping with the newcomer. When a first-join-peer tries to find the index peers with buffer overlapping by tracing backward and forward along the closest index peer's predecessor and successor, the expected number of hops to be traced is a constant. Then the dynamic programming algorithm is applied to the found peers plus the new peer to figure out which peers should be pruned from the index overlay. The expected number of nodes a new client should contact is also a constant.

Wang *et al.* introduce the Dynamic Skip List (DSL) to inherently accommodates dynamic and asynchronous peers [32]. A skip list is an ordered list of *keys* with additional parallel links. The key here is chosen to be the playback offset of each node. Assume that there are N keys in the list, indexed from 1 to N. The $(i \times 2^l)th$ key, $i = 1, 2, ..., l = 0, 1, ...$, will have links to the $((i-1) \times 2^l)th$ keys respectively. Besides, DSL is built with no MaxLayer limit and a newly inserted key will stop promoting itself only when it fails. As unnecessary neighbor information has to be maintained, which reduces the

search efficiency, DSL compresses all the layers above L into a single *top layer*, where $L = \log(\frac{N}{\log N})$. Its structure is shown in Fig. 7.

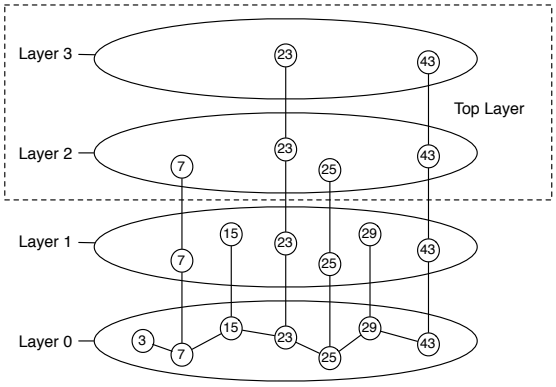

Fig. 7. A DSL of 7 nodes. The number in each node represents the key of this node.

In this layered representation, a single key in the list is mapped into multiple logical nodes along the same column. Since the parallel links in higher layers skip geometrically more than those in lower layers, a key search can be started from the highest layer so as to quickly skip unnecessary parts and then progressively move to lower layers until it hits a logical node having the key. The complexity of this top-down search is $O(\log N)$ [29]. The key is updated over time according to the playback progress. The server periodically checks the size of a randomly selected layer, estimates the overlay size N, and then accordingly releases or merges layers to ensure that there are $O(\log N)$ logical nodes in the top layer. It is shown that each overlay node in the system, including the server, maintains at most $O(\log N)$ extra information.

In a new distributed cache management model (DCMM) proposed by Liang *et al.*, peers are assigned fixed segments in turn according to the order peer join the system and all the peers whose cache content compose an integrated program into some chord-like rings [23]. The size of each ring equals the number of segments of a media file, i.e., if the media file is divided into M segments, then the ring consists of M peers. All segments have a number from 1 to M according to the playback time. Each peer in ring caches the corresponding media segment according to the time when peers join the system in turn. Segments of a program can be quickly located within these rings. When searching a specific segment, peer tries to find peers caching this segment in its own ring. If it fails, the requests will be transfered to the lower ring. This process is repeated until the peer is found or the message arrives at the ring 0.

GridCast uses two content proximity metrics *playhead* and *playcache* to estimate the sharing potential between pairs of peers [12], which can be de-

scribed as follows:

$$playhead_{(B\rightarrow A)} = |offset_B - offset_A| \tag{13}$$

$$playcache_{(B\rightarrow A)} = \frac{\sum_{i=\min}^{\max} chunkmap_{(B)}[i]}{\max - \min + 1}. \tag{14}$$

Playhead proximity roughly estimates the potential for sharing using only the current locations of the peers' playheads. Two peers are considered to have better sharing potential if their playheads are closer. On the other hand, *playcache* measures how many chunks can be provided by the other peer by comparing the contents of two caches directly. The more chunks B caches within the prefetch window of A, the better sharing it has. Neighbors who share the metadata but do not share chunks are selected using playhead proximity while partners used for chunk sharing are promoted by playcache proximity when the number of them fails below a certain degree, 50 for neighbors and 25 for partners in this case.

While most of content searching algorithms in the peer-assisted VoD systems support random seek, few of them consider the problem of enabling advanced VCR functions such as fast forward or fast rewind. Offering more freely user interactivity that is lacking in the traditional video broadcasting services, advanced VCR operations are highly desirable, yet, challenging to implement in online streaming environment as they change the playback speed of the media and require the extra bandwidth at the server and the underlying network.

To the best of our knowledge, only two papers discuss this problem. In [19], Guo *et al.* try to solve the fast forward and fast rewind problem by adjusting segment deadlines to reflect the change of playback speed, so that the new deadlines can be used for the proposed real-time scheduling based peer selection algorithm to recompute the uploading urgency metrics. Inspired by the idea of realizing these two functions by playing one segment out of v segments, where v is the speed of fast forward or rewind, Wang *et al.* present the number of logical nodes to be skipped ahead is $\frac{n(v-1)}{d}$ for each jump operation for fast forward, where d is the average difference between two consecutive nodes, and it needs $O(1)$ time [32]. Further, they obtain the upper bound of the speed a system can provide. Under simplified assumptions and through roughly evaluation, the ability of solving the fast forward or rewind problem of these mechanisms are not that convincing. Thus, designing peer-assisted VoD systems that can better support advanced VCR functions is still in its infancy.

4 Network Coding in Peer-Assisted Streaming Systems

In this section, we illustrate the role of network coding in peer-assisted streaming systems. Due to the ability of making optimal use of bandwidth resources, network coding is proposed to be used in segment scheduling algorithms in peer-assisted streaming systems. By using random network coding, peers are allowed to retrieve fractionalized segments in the parallel machine scheduling context and any received segment is useful with high probability. An excellent property of network-coding based segment scheduling algorithm is that the missing segment on a downstream peer can be served by *multiple* seeds, as each uses its randomized selection algorithm to select a segment to send coded blocks, which is illustrated in Fig. 8.

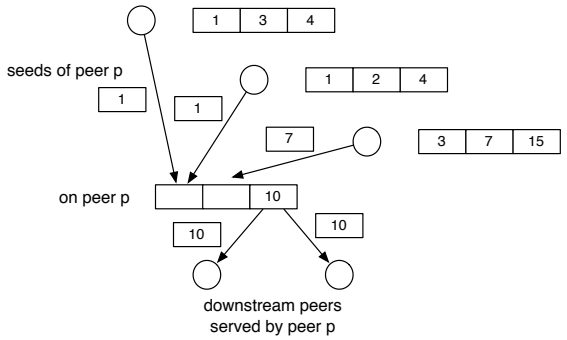

Fig. 8. An illustration of network-coding based segment scheduling algorithm

However, the limit lies in that peers have to wait to download the complete file before they can start decoding. This problem is solved by restricting network coding into one segment, which reduces the start-up delay into one segment size. What is more, coding prevents the occurrence of rare blocks within a segment, and ensures that the bandwidth is not wasted in distributing the same block multiple times. In essence, coding minimizes the risk of making the wrong uploading decision.

4.1 Core Idea

Here we introduce the core idea of network-coding based segment scheduling algorithm. In it, each segment of the media streaming is further divided into n blocks $[b_1, b_2, ..., b_n]$, with each b_i a fixed number of bytes k (refereed to as the block size). When a segment is selected by a seed p to code for its downstream peer, the seed independently and randomly chooses a set of coding coefficients $[c_1^p, c_2^p, ..., c_m^p](m \leq n)$ in a finite field F_q for each coded block. It then randomly chooses m blocks —$[b_1^p, b_2^p, ..., b_m^p]$ —out of all the blocks in the segment is has received so far, and produces one coded block x of k bytes, as shown in Fig. 9

$$x = \sum_{i=1}^{m} c_i^p \cdot b_i^p \tag{15}$$

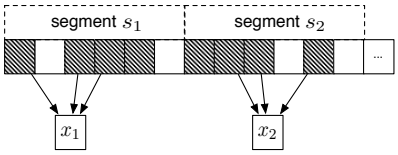

Fig. 9. An example of coding operation on peer p, where m=3, n=6

As soon as a peer receives the first coded block of this segment, it starts to progressively decode using Gauss-Jordan elimination. As a total of n coded blocks $\mathbf{x} = [x_1, x_2, ..., x_n]$ have been received, the original blocks can be immediately recovered as Gauss-Jordan elimination computes:

$$\mathbf{b} = \mathbf{A}^{-1}\mathbf{x}^T, \tag{16}$$

where $\mathbf{A}$ is the matrix formed by coding coefficients of $\mathbf{x}$.

▷ *Coding window optimization*: The key difference of network-coding based segment scheduling algorithms lies in the way of choosing m, which is also called the coding window. In Wang *et al.*'s algorithm [32], m is simply chosen to be the length of the segment, which means the coding range covers all the expected segment. Though a larger coding window makes better utilization of resource, it causes a higher probability of segments being missing upon the playback deadline due to the longer waiting time before decoding. R^2 [34], which is proposed by Wang *et al.*, refers the ratio m/n to as density, and a low ratio leads to a sparse decoding matrices. Their previous work [33] has experimentally shown that the density can be as low as 6% without leading to linear dependency among coded blocks. Then in deadline-aware network coding scheduling introduced by Chi *et al.* [13], it tries to find the as large an m as possible for better coding efficiency while controlling m so that no segment will miss its playback deadline by transforming the formulated coding window assignment problem in to a max-flow problem.

4.2 Analytical Study of the Basic Model

Other than protocol designs, Feng *et al.* seek to mathematically analyze peer-assisted streaming systems with network coding, with a focus on playback quality, initial buffering delays, server bandwidth costs, as well as extreme peer dynamics [15]. They consider the coding performance in flash crowd scenarios when most peers join the system approximately at the same time and highly dynamic scenarios in which peer join and leave the system in a highly volatile fashion.

In particular, the server strength δ is defined as $\delta = \frac{U_s}{NU_p}$, where U_p is the average upload capacity of participating peers, and U_s is the server upload capacity. It is proved that the sufficient conditions on smooth playback at a streaming rate R during any flash crowd with scale N is as follows:

$$\begin{aligned} U_s + NU_p &= (1+\epsilon)NR, \\ \epsilon &= \alpha + \frac{\ln(1+\delta) - \ln\delta}{m}. \end{aligned} \tag{17}$$

where m is the number of coded blocks in each segment and α denotes the fraction of linearly dependent coded blocks induced by network coding. This demonstrates that using network coding can achieve near-optimal performance in term of sustainable streaming rate during a flash crowd. Also, they show that in terms of initial buffering delay, algorithms using coding are within a factor of $2(1+\epsilon)$ of the optimal streaming scheme, which reflects the ability of guarantee very short initial buffering delays during a flash crowd of network coding. Besides, an upper bound of the additional server capacity to handle peer dynamics in current time slot is given to be $W_1U_1 - (1+\epsilon)W_1R$, where W_1 denotes the number of departures of high-bandwidth peers in current time slot.

In general, network-coding based segment scheduling algorithms are shown sufficient to achieve provably good overall performance in realistic settings.

4.3 Implementation Results

Annapureddy *et al.* evaluate the efficiency of network-coding based segment scheduling protocols through simulation [5, 6]. Considering a flash crowd where 20 peers join the network, by comparing with an optimal scheme without network coding—global-rarest first policy, where peer requests the globally rarest block in the target segment of its interest, it is shown that with network coding, the segment scheduling protocol provides a greater throughput than the global-rarest policy (about 14% times better) and results in significantly less variance and more predictable download times.

5 Conclusion

The study of peer-assisted media streaming systems has boomed for recent years as videos has become the dominant type of traffic over the internet, dwarfing other types of traffic. On one hand, peer-to-peer solutions have shown great promise in supporting media broadcast, which is witnessed by their increasingly widespread deployments. On the other hand, there still exists some technical challenges that might be the hurdles need to be overcome, or otherwise they may obstacle the development of peer-assisted streaming systems.

In the analytical works that try to model and characterize the behavior of such peer-assisted systems, the majority of them are under the assumption that all the peers are homogeneously, which means most of them model the peers with same upload and download capacities. The problem is that heterogeneity exists among peers in the real world. For example, peers behind Ethernet can have an upload and download bandwidth of up to $1Mbps$ to $10Mbps$, while in contrast, peers behind ADSL only have an upload bandwidth of $256Kbps$ to $512Kbps$ and their download capacity is about $1Mbps$ to $2Mbps$. Treating them all equally is obviously not a good idea.

The conventional method of proving the accuracy of analysis is by simulation. While most of the existing empirical studies only have a few hundred peers involved typically, with the only exception of [15, 16], which have a scale of more than $200,000$ peers at times throughout their simulations. By stating characterizing the large scale property of peer-assisted streaming systems, only a few hundred of peers can definitely affect the accuracy of their analysis.

By implementing real peer-assisted media streaming systems over the Internet, NATs and firewalls impose fundamental restrictions on pair-wise connectivity of nodes on an overlay, and may prohibit direct communication with one another. Whether communication is possible between two nodes depends on several factors such as the transport protocol(UDP or TCP), the particular kind of NAT/firewall, and whether the nodes are located behind the same private network. For example, under UDP protocol, the connectivity form NAT to firewall is only possible for some cases. When under TCP protocol, the connectivity from NAT to firewall is not possible [26].

In this chapter, we reviewed the state-of-art of peer-assisted media streaming systems, paying special attention to peer-assisted VoD systems. Bedsides, we also present the implementation of network coding in peer-assisted media streaming systems and the benefits it brought. While networking researchers have achieved substantial progress in this area over the past years, there are still several open problems, such as incentive-based mechanism, multichannel consideration, lager storage space in VoD systems, advanced VCR functions supporting and so on, which may present some areas of future research.

References

[1] Alessandria E, Gallo M, Leonardi E, Mellia M, Meo M (2009) P2P-TV Systems under Adverse Network Conditions: A Measurement Study. In: Proc. IEEE INFOCOM, pp 100–108, DOI 10.1109/INFCOM.2009.5061911

[2] Ali S, Mathur A, Zhang H (2006) Measurement of Commercial Peer-To-Peer Live Video Streaming. In: Proc. of ICST Workshop on Recent Advances in Peer-to-Peer Streaming

[3] Allen MS, Zhao BY, Wolski R (2007) Deploying Video-on-Demand Services on Cable Networks. In: Proc. 27th International Confer-

ence on Distributed Computing Systems (ICDCS), p 63, DOI http://dx.doi.org/10.1109/ICDCS.2007.98

[4] Annapureddy S, Gkantsidis C, Rodriguez P, Massoulie L (2006) Providing Video-on-Demand using Peer-to-Peer Networks. In: Proc. Internet Protocol TeleVision Workshop (IPTV), vol 6

[5] Annapureddy S, Guha S, Gkantsidis C, Gunawardena D, Rodriguez P (2007) Exploring VoD in P2P Swarming Systems. In: Proc. IEEE INFOCOM, pp 2571–2575, DOI 10.1109/INFCOM.2007.323

[6] Annapureddy S, Guha S, Gkantsidis C, Gunawardena D, Rodriguez PR (2007) Is High-quality VoD Feasible using P2P Swarming? In: Proc. 16th International Conference on World Wide Web (WWW), pp 903–912, DOI 10.1145/1242572.1242694

[7] Bhagwan R, Tati K, Cheng YC, Savage S, Voelker GM (2004) Total Recall: System Support for Automated Availability Management. In: Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI), pp 337–350

[8] Cai Y, Natarajan A, Wong J (2007) On Scheduling of Peer-to-Peer Video Services. IEEE Journal on Selected Areas in Communications 25(1):140–145

[9] Cha M, Kwak H, Rodriguez P, Ahn YY, Moon S (2007) I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In: Proc. 7th SIGCOMM Conference on Internet Measurement (IMC), pp 1–14, DOI 10.1145/1298306.1298309

[10] Cheng B, Jin H, Liao X (2007) Supporting VCR Functions in P2P VoD Services Using Ring-Assisted Overlays. In: Proc. 16th IEEE International Conference on Communications (ICC), pp 1698–1703, DOI 10.1109/ICC.2007.284

[11] Cheng B, Stein L, Jin H, Zhang Z (2008) A Framework for Lazy Replication in P2P VoD. In: Proc. 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), pp 93–98, DOI 10.1145/1496046.1496068

[12] Cheng B, Stein L, Jin H, Zhang Z (2008) Towards Cinematic Internet Video-on-Demand. In: Proc. of the 3rd European Conference on Computer Systems (EuroSys), pp 109–122, DOI 10.1145/1352592.1352605

[13] Chi H, Zhang Q, Jia J, Shen X (2007) Efficient Search and Scheduling in P2P-based Media-on-Demand Streaming Service. IEEE Journal on Selected Areas in Communications 25(1):119–130, DOI 10.1109/JSAC.2007.070112

[14] Choe YR, Schuff DL, Dyaberi JM, Pai VS (2007) Improving VoD Server Efficiency with BitTorrent. In: Proc. 15th ACM International Conference on Multimedia, pp 117–126, DOI 10.1145/1291233.1291258

[15] Feng C, Li B (2008) On Large-Scale Peer-to-Peer Streaming Systems with Network Coding. In: Proc. 16th ACM International Conference on Multimedia, pp 269–278, DOI 10.1145/1459359.1459396

[16] Feng C, Li B, Li B (2009) Understanding the Performance Gap between Pull-based Mesh Streaming Protocols and Fundamental Limits. In: Proc. IEEE INFOCOM

[17] Graffi K, Kaune S, Pussep K, Kovacevic A, Steinmetz R (2008) Load Balancing for Multimedia Streaming in Heterogeneous Peer-to-Peer Systems. In: Proc. 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), pp 99–104, DOI 10.1145/1496046.1496069

[18] Guo L, Chen S, Zhang X (2006) Design and Evaluation of a Scalable and Reliable P2P Assisted Proxy for On-Demand Streaming Media Delivery. IEEE Transanctions on Knowledge and Data Engineering 18(5):669–682, DOI http://dx.doi.org/10.1109/TKDE.2006.79

[19] Guo Y, Yu S, Liu H, Mathur S, Ramaswamy K (2008) Supporting VCR Operation in a Mesh-Based P2P VoD System. In: Proc. 5th Consumer Communications and Networking Conference (CCNC), pp 452–457, DOI 10.1109/ccnc08.2007.107

[20] He Y, Lee I, Guan L (2008) Distributed Throughput Maximization in Hybrid-Forwarding P2P VoD Applications. In: Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp 2165–2168, DOI 10.1109/ICASSP.2008.4518072

[21] Huang C, Li J, Ross KW (2007) Can Internet Video-on-Demand be Profitable? In: Proc. ACM SIGCOMM, vol 37, pp 133–144, DOI 10.1145/1282427.1282396

[22] Li B, Xie S, Keung G, Liu J, Stoica I, Zhang H, Zhang X (2007) An Empirical Study of the Coolstreaming+ System. IEEE Journal on Selected Areas in Communications 25(9):1627–1639, DOI 10.1109/JSAC.2007.071203

[23] Liang W, Huang J, Huang J (2008) A Distributed Cache Management Model for P2P VoD System. In: Proc. International Conference on Computer Science and Software Engineering (ICCSSE), vol 3, pp 5–8, DOI 10.1109/CSSE.2008.1059

[24] Liao X, Jin H (2007) OCTOPUS: A Hybrid Scheduling Strategy for P2P VoD Services. In: Proc. 6th International Conference on Grid and Cooperative Computing (GCC), pp 26–33, DOI 10.1109/GCC.2007.89

[25] Liao X, Jin H, Liu Y, Ni LM, Deng D (2006) AnySee: Peer-to-Peer Live Streaming. In: Proc. IEEE INFOCOM, pp 1–10, DOI 10.1109/INFOCOM.2006.288

[26] Liu J, Rao SG, Li B, Zhang H (2008) Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. In: Proceedings of the IEEE, vol 96, pp 11–24, DOI 10.1109/JPROC.2007.909921

[27] Nafaa A, Murphy S, Murphy L (2008) Analysis of a Large-Scale VoD Architecture for Broadband Operators: A P2P-Based Solution. IEEE Communications Magazine 46(12):47–55, DOI 10.1109/MCOM.2008.4689207

[28] Parvez N, Williamson C, Mahanti A, Carlsson N (2008) Analysis of Bittorrent-like Protocols for On-Demand Stored Media Streaming.

SIGMETRICS Performance Evaluation Review 36(1):301–312, DOI 10.1145/1384529.1375492

[29] Pugh W (1990) Skip Lists: A Probabilistic Alternative to Balanced Trees. Communications of the ACM 33:668–676

[30] Silverston T, Fourmaux O, Crowcroft J (2008) Towards an Incentive Mechanism for Peer-to-Peer Multimedia Live Streaming Systems. In: Proc. 8th International Conference on Peer-to-Peer Computing (P2P), pp 125–128, DOI http://dx.doi.org/10.1109/P2P.2008.25

[31] Vratonjić N, Gupta P, Knežević N, Kostić D, Rowstron A (2007) Enabling DVD-like Features in P2P Video-on-Demand Systems. In: Proc. Workshop on Peer-to-Peer Streaming and IP-TV (P2P-TV), pp 329–334, DOI 10.1145/1326320.1326326

[32] Wang D, Liu J (2008) A Dynamic Skip List-Based Overlay for On-Demand Media Streaming with VCR Interactions. IEEE Transactions on Parallel and Distributed Systems 19(4):503–514, DOI http://dx.doi.org/10.1109/TPDS.2007.70748

[33] Wang M, Li B (2006) How Practical is Network Coding? In: Proc. 14th International Workshop on Quality of Service (IWQoS), pp 274–278, DOI 10.1109/IWQOS.2006.250480

[34] Wang M, Li B (2007) R^2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming. IEEE Journal on Selected Areas in Communications 25(9):1655–1666, DOI 10.1109/JSAC.2007.071205

[35] Wu C, Li B, Zhao S (2007) Characterizing Peer-to-Peer Streaming Flows. IEEE Journal on Selected Areas in Communications 25(9):1612–1626, DOI 10.1109/JSAC.2007.071202

[36] Wu C, Li B, Zhao S (2008) Multi-Channel Live P2P Streaming: Refocusing on Servers. In: Proc. of IEEE INFOCOM, pp 1355–1363

[37] Ying L, Basu A (2005) pcVOD: Internet Peer-to-Peer Video-On-Demand with Storage Caching on Peers. In: Proc. 11th International Coference on Distributed Multimedia Systems (DMS), pp 218–223

[38] Zhang M, Luo JG, Zhao L, Yang SQ (2005) A Peer-to-Peer Network for Live Media Streaming Using a Push-Pull Approach. In: Proc. 13th ACM International Conference on Multimedia, pp 287–290, DOI 10.1145/1101149.1101206

[39] Zhang X, Liu J, Li B, Yum YS (2005) CoolStreaming/DONet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming. In: Proc. IEEE INFOCOM, vol 3, pp 2102–2111, DOI 10.1109/INFCOM.2005.1498486

[40] Zhou Y, Chiu DM, Lui J (2007) A Simple Model for Analyzing P2P Streaming Protocols. In: Proc. 15th International Conference on Network Protocols (ICNP), pp 226–235, DOI 10.1109/ICNP.2007.4375853