Two Decades of Internet Video Streaming: A Retrospective View

BAOCHUN LI, University of Toronto
ZHI WANG, Tsinghua University
JIANGCHUAN LIU, Simon Fraser University
WENWU ZHU, Tsinghua University

Over the past two decades, research on video streaming over the Internet has received a tremendous amount of research attention from both academia and industry. Research in the first decade largely focused on transport protocols, while research in the second decade has shifted to the peer-to-peer paradigm of video distribution, and more recently, to HTTP streaming based on cloud computing and social media. This paper reviews existing results over the past two decades in video streaming, with a focus on peer-to-peer streaming protocols, the effects of social media, and the migration to HTTP streaming.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms: Multimedia streaming

Additional Key Words and Phrases: Video streaming, Multicast, P2P streaming, HTTP streaming, Cloud computing, Social media

ACM Reference Format:

Li, B., Wang, Z., Liu, J., and Zhu, W.. 2013. Two Decades of Internet Video Streaming:
A Retrospective View. ACM Trans. Multimedia Comput. Commun. Appl. 2, 3, Article 1 (May 2013), 20 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Driving by an insatiate appetite for bandwidth in the Internet, advances in media compression technologies, and accelerating user demand, video streaming over the Internet has quickly risen to become a mainstream "killer" application over the past two decades. Fundamentally different from the traditional wisdom of starting playback after complete downloads, more and more users are used to watching streaming video directly, on their portable computers or mobile devices. With video streaming, a user can begin playing a video segment before the entire file has been transmitted. As such,

Addresses of the authors: B. Li, Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario M5S 3G4, Canada; email: bli@eecg.toronto.edu; Z. Wang, W. Zhu, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China; email: wangzhi04@mails.tsinghua.edu.cn, wwzhu@tsinghua.edu.cn; J. Liu, School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, BC V5A 1S6, Canada; email: jcliu@cs.sfu.ca.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

video streaming has received a substantial amount of research attention in the past two decades, from academia and industry alike.

In this paper, we seek to go back in the time machine, and present an extensive overview of the history of Internet video streaming, covering its main milestones in the past two decades of research and development. The paper will be presented in three stages, following a chronological order.

Client-server video streaming. During the 1990s and early 2000s, research attention is mostly focused on new streaming protocols, such as the Real-time Transport Protocol (RTP), designed within the AVT working group, specifically for streaming media. The initial set of protocols were incorporated by media players installed as the clients receive video streams from the streaming servers over the Internet.

Peer-to-peer video streaming. To serve an increasing number of streaming users, from 2003-2009, peer-to-peer (P2P) media streaming received a substantial amount of research attention, and was widely applied for both live and on-demand video streaming. It was based on the design philosophy that peers serve as both clients and servers, compared to the traditional clientserver model where clients only consume video. Real-world systems, such as PPLive, succeeded to provide a large number of video streams (at around 400 Kbps) to more than 1.5 million users, while consuming less than 10 Mbps of server bandwidth.

Cloud-based HTTP streaming and the effects of social media. P2P video streaming requires users to install dedicated applications to be able to stream the videos. In the late 2000s, web-based video streaming was accepted by most of the Internet users, due to its convenience. Web-based video streaming allows users to play videos directly from their web browsers, rather than having to download and install dedicated applications. HTTP streaming works by breaking the overall stream into a sequence of small HTTP-based file downloads, each download loading one short chunk of an overall potentially unbounded transport stream. Microsoft, Apple, and Adobe have all implemented HTTP live streaming in their corresponding products. Based on web-based streaming, we are witnessing a migration to cloud computing and social media as the predominant platforms for hosting and sharing streaming media.

The remainder of this paper is divided into three main sections. Section 2 covers early research explorations in streaming video over the Internet. Section 3 covers the tremendous amount of research work on the use of the peer-to-peer design philosophy to make video streaming scalable. Section 4 presents an overview on the migration to cloud-powered HTTP streaming and the effects of social media. Finally, Section 5 summarizes the lessons learned and concludes the paper.

2. INTERNET VIDEO STREAMING USING THE CLIENT-SERVER MODEL

The development of video compression technologies in the 1980s and the growth and popularity of the Internet in the 1990s motivated video streaming over the Internet, which can be classified as continuous media because it consists of a sequence of media quanta (*i.e.*, video frames) delivered over the Internet. The vision of enabling simultaneous video delivery to a large number of receivers has been driving the research agenda in the networking community for over two decades. For much of the 1990s, the academic and industry communities have paid a substantial amount of research attention on devising new protocols to stream videos from streaming servers to a large number of clients over the Internet.

Built on top of the Internet Protocol (IP), video streaming is designed with the aim of providing Quality of Service (QoS) and achieving efficiency for streaming video over the best-effort Internet. In this section, we briefly summarize how video streaming has evolved in the first decade using the traditional client-server model, before the advent of peer-to-peer video streaming. While we are not able

to cover all important research results in this section, excellent survey papers exist in the literature on this topic (*e.g.,* [Wu et al. 2001]).

2.1 Transport Protocols for Video Streaming

Traditional Internet protocols were used to deliver data without real-time streaming requirements, including video files that can be downloaded by users and later played back after the download finishes. In comparison, in order to support interactive controls of video playback during video streaming over the Internet, a set of new protocols for video streaming have been designed and implemented. As these protocols became critically important for Internet video streaming, the Internet Engineering Task Force (IETF) standardized the RTP/RTCP/RTSP protocol suite [Schulzrinne et al. 1996; 2003; Schulzrinne et al. 1998]. These protocols are designed specifically for video streaming, and fall into the following two categories:

♢ *Transport protocols.* RTP, based on the User Datagram Protocol (UDP), defines a standardized packet format for delivering video over IP, and is designed for end-to-end real-time transfer of stream data. The RTP Control Protocol (RTCP), also based on UDP, is designed to monitor transmission statistics and QoS, and to achieve synchronization across multiple streams. Important commercial and open-source products, such as QuickTime and VLC, support RTP.

♢ *Session control protocols.* These protocols are designed to define the messages and procedures to control video delivery during an established session. The Real Time Streaming Protocol (RTSP) [Schulzrinne et al. 1998] and the Session Initiation Protocol (SIP) [Rosenberg et al. 2002] are examples of session control protocols. For example, RTSP creates and maintains media sessions, and provides VCR-style control functionality, enabling users to pause, resume, or seek in video streams just like local playback.

To support video streaming over the Internet, it was believed that transport protocols should be designed to meet real-time video delivery needs, by maximizing video quality in the presence of packet loss. Bursty loss and excessive delay had a devastating effect on video presentation quality, and they were usually caused by network congestion. To support Internet video streaming, such congestion control took the form of rate control and rate shaping [Floyd et al. 2000; Jacobs and Eleftheriadis 1998], which attempted to minimize the possibility of network congestion by matching the rate of the video stream to the available network bandwidth.

♢ *Rate control*: Rate control was a technique used to determine the sending rate of video traffic based on the estimated available bandwidth in the network. Traditional rate control schemes were classified into three categories [Wu et al. 2001]: source-based, receiver-based, and hybrid rate control.

♢ *Rate shaping*: The objective of rate shaping was to match the rate of a pre-compressed video bitstream to the target rate constraint [Jacobs and Eleftheriadis 1998; Chen and Chen 2004]. A rate shaper was required for source-based rate control, because the stored video was pre-compressed at a certain rate, which did not match the available bandwidth in the network. There were many types of rate shapers, such as the codec filter, frame-dropping filter, layer-dropping filter, frequency filter, and re-quantization filter [Wu et al. 2001].

2.2 Error Control for Video Streaming

While the purpose of congestion control is to avoid congestion, packet loss is inevitable in the Internet and may have significant impact on perceptual quality. Error control [Wang and Zhu 1998] was a set of strategies used to ensure the smooth video streaming even when there were errors in packet delivery. A large number of error control mechanisms have been proposed in the literature:

♢ *Forward Error Correction (FEC)*: The principle of FEC is to add redundant information so that the original message could be reconstructed in the presence of packet loss [Frossard 2001]. Based on the

type of redundant information to be added, FEC schemes are divided into three categories: channel coding, source coding, and joint source/channel coding [Wu et al. 2000].

♢ *Error-resilient encoding*: The objective of error-resilient encoding is to enhance robustness of compressed video to packet loss. The standardized error-resilient encoding schemes include resynchronization marking, data partitioning, and data recovery [Wang et al. 2000]. However, these approaches are targeted at error-prone environments like wireless channels, and are not be applicable to the Internet environment. For video transmission over the Internet, the boundary of a packet already provided a synchronization point in the variable-length coded bit-stream at the receiver side.

♢ *Error concealment*: Error-resilient encoding is executed by the source to enhance robustness of the compressed video before packet loss actually happens [Wang and Zhu 1998]. There are two basic approaches for error concealment: spatial and temporal interpolation [Kung et al. 2006]. Multiple Description Coding (MDC) is also designed to conceal losses in Internet video streaming [Goyal 2001].

♢ *Delay-constrained retransmission*: Retransmission is usually dismissed as a method to recover lost packets in real-time video, since a retransmitted packet may miss its play-out time. However, if the one-way delay is relatively short with respect to the maximum allowable delay, a retransmission-based approach (called delay-constrained retransmission) may still be a viable option for error control [Kalman et al. 2004].

2.3 Unicast and Multicast Streaming

In the early 1990s, only a small number of users were able to enjoy video streaming on the Internet. A single streaming server was able to handle all the video requests, where a unicast connection was established between the client and the server for video content delivery. Unicast streaming worked either for live streaming or on-demand streaming [Majumda et al. 2002]. As the population of video streaming users has grown, the unicast approach limits the size of the client population, and has quickly become infeasible as the number of clients scales up. Multicast protocols have been proposed to be more scalable to a large number of clients.

The Internet's original design, while well suited for point-to-point applications, failed to effectively support large-scale video multicast. A large number of emerging applications, including Internet TV, broadcast of sports events, required the support for video broadcast, *i.e.*, simultaneous video delivery to a large number of receivers.

IP multicast. *IP multicast* has been proposed [Diot et al. 2000; Deering and Cheriton 1990] as an extension to the IP layer, with the objective of providing efficient multipoint packet delivery. Given that the network topology is best known in the network layer, multicast routing in this layer is also the most efficient. However, IP Multicast had several drawbacks that limited its scalability: (1) the relatively sparse deployment of IP Multicast; and (2) the prohibitive cost of bandwidth required for server-based solutions or Content Delivery Networks (CDNs). Clearly, there existed a vacuum for cost-effective, ubiquitous support for Internet-wide video delivery.

Multicast for heterogeneous users. Another challenge for video multicast was the intrinsic heterogeneity of users. In traditional end-to-end adaptation schemes, the sender adjusted its transmission rate according to some feedback from its receiver. Bharadvaj *et al.* [Bharadvaj et al. 1998] studied the transcoding proxy to support the heterogeneous users. Ooi [Ooi 2005] studied the multicast heterogeneous network environment.

2.4 Stream Replication and Content Caching

Stream replication can be viewed as a trade-off between single-rate multicast and multiple point-to-point connections [Kim and Ammar 2001]. Its feasibility has been well justified in a typical multicast environment where the bandwidth of receivers usually follow some clustered distribution. As a result,

a limited number of streams could be used to match these clusters to achieve reasonably good performance. For example, McCanne *et al.* [McCanne et al. 1996] proposed the first practical adaptation protocol for cumulative layered video multicast over the Internet. This protocol, known as Receiver-driven Layered Multicast (RLM), takes advantage of the dynamic group concept in the IP multicast model.

An important technique for improving scalability and reachability and for reducing the latency of video streaming is video caching, which has been deployed by the publishers, *i.e.*, CPs (Content Providers) or ISPs (Internet Service Providers). Video caching is beneficial due to the fact that different clients will load much of the same contents, and it places content closer to the clients by making local copies of contents that the clients retrieve. With video caching, bandwidth consumption and streaming server load may be reduced, latencies of streaming video to the clients may be shorter, and availability may be improved. Video caches can even form hierarchies, such that the load on the original streaming server can be further alleviated. It has been demonstrated that caching strategies that are specific to particular types of objects can help improve the overall performance. For example, Sen et al. [Sen et al. 1999] demonstrated that by only caching the prefixes of videos, a large amount of videos could be served by a few megabytes of buffer space at the proxy storage.

2.5 Application-Layer Multicast

Since video streaming over the Internet can be fundamentally represented by a multicast problem, the notion of application-layer multicast was first proposed by Chu *et al.* [Chu et al. 2000] as a reasonable application-layer replacement for IP multicast, and has since become an active research topic. When it was first proposed, it was not only intended for streaming, but for file sharing as well. Though both application-layer multicast and IP multicast require an intermediate node in the network topology to support the replication of data packets, its implementation is less demanding on end hosts in the application layer, as compared to switches and routers in the Internet core [Hosseini et al. 2007].

Chu *et al.* [Chu et al. 2000] have shown that application-layer multicast performs reasonably well as compared to IP multicast, incurring a low delay and a reasonable amount of bandwidth penalty. In initial proposals of application-layer multicast, a single multicast tree is constructed for each multicast session. For example, Overcast [Jannotti et al. 2000] attempted to organize a bandwidth-efficient tree by letting peers join near the root, and then migrating them down the tree to a position according to bandwidth availability. The advantage of such a design, shown in Fig. 1, is its simplicity.

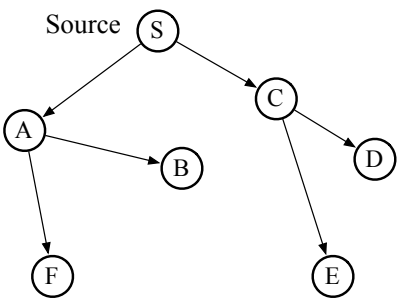

Fig. 1. Application-layer multicast with a single tree.

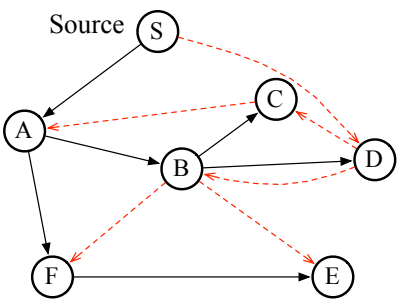

Fig. 2. Application-layer multicast with multiple trees.

As a simple design, single-tree multicast suffers from a few problems. As a start, such a design may not be *fair* to all the peers. As we can easily see from Fig. 1, when the tree is formed, some peers are

chosen to be interior nodes that must contribute their upload bandwidth to support others, while others are leaf nodes, not requiring to contribute any upload bandwidth. Even if fairness is not a concern, the multicast rate that a child node can enjoy is restricted by the upload bandwidth available at its parent. In case the parent node leaves the multicast session, its children will be left in the cold, waiting for a new parent.

To improve fairness in application-layer multicast, it was proposed in 2003 that streams were split to multiple "slices," and distributed across a "forest" of multiple interior-node-disjoint multicast trees [Castro et al. 2003], or a mesh overlay on top of a tree [Kostić et al. 2003]. Fig. 2 illustrates an example of multicasting with two multicast trees, constructed with the intention that a majority of nodes that are interior nodes in one tree will be leaf nodes in the other tree. By distributing the responsibility of contributing uploading bandwidth to most of the peers in the multicast session, the fairness problem is mitigated, yet the robustness problem remains to be solved.

3. PEER-TO-PEER VIDEO STREAMING

P2P video streaming was not largely deployed until the pull-based data-driven strategies were invented. The structure of the pull-based overlay was much simpler than the tree-based structures, and achieved promising performance [Zhang et al. 2007]. Powered by network coding, peer-to-peer paradigm was proven to be highly efficient to deliver the video streams.

3.1 P2P Video Streaming based on the Conventional Wisdom

With the advent of Napster and Gnutella from 1998 to 2001, the community was the first that brought design principles of peer-to-peer systems to the attention of academic researchers. As one of the best example of the values of academic research in peer-to-peer systems, the peer-to-peer design has been applied to the scenario of large-scale video streaming in 2005, when dedicated interests in analyzing BitTorrent and in proposing new application-layer multicast protocols have converged to the design of a new protocol for peer-to-peer video streaming. Some results in [Bharambe et al. 2006; Legout et al. 2006] confirm that the simple rarest-first policy in BitTorrent system performs near-optimally in terms of upload capacity utilization and thereby the downloading time. Some studies [Kumar et al. 2007] also discussed the impact of the churn rate on the performance in peer-to-peer streaming.

The peer-to-peer approach is attractive in two reasons. *First*, it does not require support from the underlying network infrastructure, and as a result, it is cost effective and easy to be deployed. *Second*, in the peer-to-peer design, a peer is not only downloading a video stream, but also uploading it to other peers watching the same program. As a result, it has the potential to scale with the group size, as a stronger demand also generates more suppliers.

There exist timing constraints that new streaming protocols must observe: if data blocks do not arrive in time, they are not useful when it comes to the time to play these blocks back. The design philosophy in BitTorrent has converged with academic solutions in application-layer multicast, and a pull-based protocol on a random mesh topology emerged, independently discovered in Chainsaw [Pai et al. 2005] and CoolStreaming [Zhang et al. 2005b]. In such a protocol, peers exchange information with their neighbors periodically about what data blocks each of them has in their buffers, and a missing data block must be explicitly requested and transferred from one of the neighbors who has it. In comparison, application-layer multicast based on multicast trees adopts a more rigid design, in that the structure of each tree needs to be actively managed as peers join and leave the session.

In a pull-based video streaming system, a dynamic *sliding window* of blocks over time needs to be distributed, unlike a fixed number of blocks in file sharing. As the sliding window moves forward in the stream over time, blocks are to be received in approximately the same sequence as they are played back, and out-of-order delivery can only occur within the confines of the sliding window. Each of the

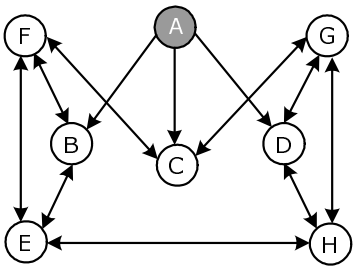

Fig. 3. An illustration of partnerships in the pull-based peer-to-peer streaming system with A being the source.

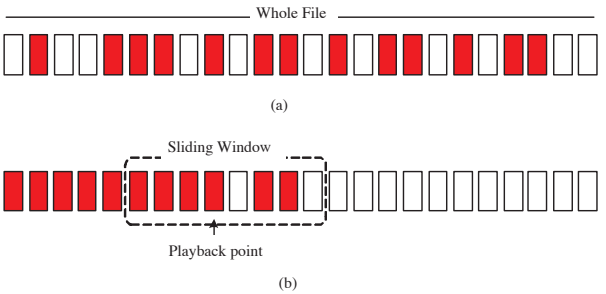

Fig. 4. Buffer snapshots of BitTorrent (a) and CoolStreaming (b), where shaded segments are available in the buffer.

blocks are distributed using a gossip protocol in a random mesh topology, requiring peers to "pull" blocks from each other. Since all participating peers are roughly synchronized with respect to their points of playback, they are able to periodically exchange the states of their respective buffers in the sliding window. Based on the knowledge of block availability in each other's sliding windows, a peer sends requests to its neighbors in order to "pull" blocks it has yet to receive.

The key aspect of the design where the pull-based peer-to-peer streaming system differs from tree-based approaches is the lack of a formal structure for video data delivery [Venkataraman et al. 2006; Magharei et al. 2007]. The main counter-argument is that mesh/pull based protocol has a lot of overhead and large delay. However, several proposed pull-push hybrid protocols have proven that they not only inherits the robustness of pull-based protocol and can achieve nearly optimal throughput but also has much lower overhead and delay [Zhang et al. 2007].

Next, we use CoolStreaming as an example to present the key modules in a general pull-based peer-to-peer streaming system. A peer consists of three key modules: (1) a membership manager, which helps the peer to maintain a partial view of other overlay peers; (2) a partnership manager, which establishes and maintains partnership with other peers; (3) a scheduler, which schedules the transmission of video data.

♢ *Group and partner management*. A pull-based peer-to-peer streaming system requires newly joining peers to contact the tracker server to obtain an initial set of partner candidates. Each peer also maintains a partial subset of other participants in the group. In particular, the system employs an existing Scalable Gossip Membership protocol, SCAM, to distribute membership messages, which enables scalable, light-weight, and uniform partial view at each peer.

A video stream is divided into segments of a uniform length, and the availability of the active segments in the buffer of a peer is represented by a Buffer Map (BM). Each peer continuously exchange its BM with its partners, and then determines which segment is to be fetched from which partner accordingly. An example of the partnership is shown in Fig. 3. Such partnerships are adaptively configured throughout a broadcast session.

♢ *Scheduling algorithm*. Timely and continuous segment delivery is crucial to video streaming systems, but not to file download. In BitTorrent, the download phases of the peers are not synchronized, and the segments can be downloaded out-of-order. The playback progress of the peers is roughly synchronized, and any segment downloaded after its playback time will be useless. A sliding window thus represents the active buffer portion, as shown in Fig. 4.

♢ *Failure recovery and partnership refinement*. A peer can depart either gracefully or accidentally due to an unexpected failure. In either case, the departure can be easily detected after an idle time, and an affected peer can quickly react through re-scheduling using the BM information of the remaining

partners. Besides this built-in recovery mechanism, the system also asks each peer to establish new partnerships with peers randomly selected from its local membership list periodically. This operation serves two purposes: first, it helps each peer to maintain a stable number of partners in the presence of peer departures; second, it helps each peer to explore partners of better quality, *e.g.*, those constantly having a higher upload bandwidth and more available segments.

Fundamentally, setting up and maintaining trees in application-layer multicast is similar to setting up a connection in a telephone network: states of parent-child relationships are established so that data blocks can be transmitted without explicit requests taking place. In contrast, the distribution of data blocks in a pull-based protocol is similar to *gossiping* in a social setting.

Following the gossiping philosophy to distribute each of the data blocks to all the peers in the multicast session, the early peer-to-peer video streaming systems have motivated the design of production-quality real-world implementations in the industry, several of which has become core technologies in start-up companies that specialized in live media streaming (*e.g.,* PPLive [Huang et al. 2008]).

Even though the gossiping philosophy sacrifices some delays in live streaming systems due to delays involved in periodic information exchanges and explicit requests, its most salient advantage is its simplicity in design and in implementation. If the current sliding window of the media stream to be distributed is divided into small media blocks, they will flow through the entire network wherever there exists idle upload bandwidth that can be tapped into, as if water flows through a wooden house with gaping holes. In particular, Zhang *et al.* [Zhang et al. 2007] have studied the performance limitation of such pull-based video streaming systems – they spotted the tradeoff between control overhead and delay, *i.e.*, the protocol has either large control overhead or large delay.

3.2 P2P Video Streaming Powered by Network Coding

Since 2005, the use of *network coding* has emerged as a potentially exciting research topic in peer-to-peer video streaming systems [Liu et al. 2010; Park et al. 2006; Chenguang et al. 2007; Seferoglu and Markopoulou 2007]. Network coding, first proposed in the information theory community in 2000 [Ahlswede et al. 2000], recognized the ability to code at intermediate network nodes in a communication session, in addition to the ability to forward and to replicate incoming packets. In contrast, traditional multicast only recognized the ability to forward and to replicate packets. In 2003, Ho *et al.* [Ho et al. 2003] have further proposed the concept of *random network coding*, where a network node transmits on each of its outgoing links a linear combination of incoming packets over a finite field, with *randomly chosen* coding coefficients.

It is natural to conceive a simple but new design of peer-to-peer video streaming systems, in which peers, as end hosts, are able to forward, replicate, and *code* incoming video data blocks. But will the use of network coding improve the performance, as compared to the traditional P2P streaming? After the initial success stories of peer-to-peer live streaming systems, coupled with the debatable advantages of using random network coding in file sharing systems, an intriguing question is whether random network coding is a suitable choice to be integrated in the design of peer-to-peer live streaming, based on pull-based random gossip protocols.

At first glance, it appears that the hazy situation is no different from the use of network coding in file sharing. However, even with production-quality implementations of pull-based live streaming protocols, they did have an "Achilles' heel:" *communication overhead.* Intuitively, since the sliding window at a peer advances itself over time, buffer availability maps need to be exchanged as frequently as needed, which may lead to a substantial amount of overhead. To mitigate such an overhead, most practical live streaming systems choose to exchange buffer states less frequently. An analytical study [Feng et al. 2009], however, has attributed the performance gap between practical systems and their theoretically optimal performance to the lack of timely exchanges of buffer states.

Would the use of random network coding be able to mitigate the problem of communication overhead? Wang *et al.* [Wang and Li 2007] have raised such a question, and presented several design principles, collectively referred to as R^2, which utilized random network coding to substantially improve the performance of live streaming systems.

Similar to the use of network coding in file sharing, R^2 divides the content of the media stream in a sliding window into generations, each of which is further divided into m blocks. With the introduction of generations in R^2, we can afford to design parameter settings so that a block is much smaller than its counterpart in traditional live streaming systems based on random gossiping. This is due to the fact that buffer states only need to be exchanged at the granularity of a generation (one bit to represent each generation), rather than a block. With the same amount of communication overhead to exchange buffer states, the size of a single block can be much smaller in R^2. When a peer serves a generation s to its downstream target peer p, it linearly encodes all the blocks it has received so far from s using random coefficients in GF(2^8), and then transmits the coded block to p. Since each peer buffers coded blocks it has received so far, it is able to linearly combine them using random coefficients, much like how random network coding is used in file sharing. Only blocks from the same generation are allowed to be coded, in order to reduce the computational complexity of network coding.

Since live streaming systems have timing requirements during playback, R^2 advocates the use of *random push* instead of "pull" to transmit data: each peer randomly selects a small number of downstream peers based on certain criteria. When serving a chosen downstream peer, it then randomly selects a generation to code within, among those that the downstream peer has not yet completely received. If generations closer to the point of playback have not been completely received, they are given a higher priority as they are more urgent.

There are a number of clear advantages brought forth by the use of random network coding in R^2. *First*, it greatly simplifies protocol design. Since coded blocks within a generation are equally useful, a peer only needs to blindly push coded blocks in the same generation till the downstream peer has obtained a sufficient number of them to decode. This eliminates the need of sending explicit requests to "pull" missing blocks, and saves the communication overhead associated with these requests. *Second*, R^2 induces much less overhead involved in buffer state exchanges, due to a smaller number of generations in the sliding window. *Finally*, R^2 makes it possible for an incoming peer to start its playback with the shortest initial buffering delay. Shown in Fig. 5, as a new peer joins the session, multiple existing peers are able to collaborate and push fresh coded blocks in the first one or two generations after the playback point, so that the rate of accumulating blocks in these generations is only limited by the new peer's available downlink bandwidth.

The advantage of such "perfect collaboration" among multiple upstream peers when serving coded blocks is not limited to shorter initial buffering delays. It also allows the streaming protocol to be more resilient to peer dynamics, since the downloading process of a particular generation is not adversely affected by the departure of any of its serving peers. Without the use of random network coding, a complex coordination protocol across multiple serving peers is needed to avoid sending duplicates to the new peer. This is another example where network coding simplifies protocol design, and as a result it not only reduces the amount of protocol overhead, but also becomes more resilient to packet losses, excessive delays, and peer departures.

It has become evident that, thanks to the power of random network coding, multiple serving peers are able to coordinate their actions serving a peer, leading to minimized buffering delays and bandwidth costs on servers. The playback quality has been satisfactory for normal-quality videos. For high-quality videos, the use of network coding has mitigated negative effects when the server bandwidth supply becomes tight in meeting the demand for bandwidth. It certainly appears that, at least in the

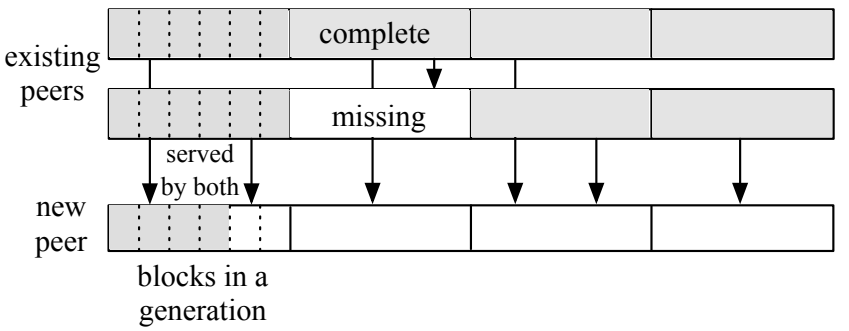

Fig. 5. Peer-to-peer live streaming with the use of random network coding: multiple existing peers are able to collaborate and serve coded blocks within the same generation to a new peer joining the session, minimizing its initial buffer delay.

context of peer-to-peer streaming systems, network coding has the true potential to deal a winning hand.

4. HTTP VIDEO STREAMING OVER THE CLOUD AND SOCIAL MEDIA

Though P2P has proven to be highly efficient in video delivery, there are issues that prevent P2P streaming from large deployment by content providers: (1) *Ease-of-use*. In P2P streaming, users are usually required to install the P2P clients or plugins to be able to cache the video contents watched and exchange the contents with others – this is not user-friendly given that today's users are so familiar with using web browsers to consume Internet contents directly; (2) *Security*. P2P relies on peers' contribution to the system – peers need to keep TCP or UDP ports open for other users to access for video chunks. When addressing the firewall block and NAT traversal problems, P2P streaming also exposes security issues; (3) *Copyright*. In a P2P streaming system that arose from illegal file sharing, users exchange contents with each other – it is very difficult for the content providers to control the copyright in the video streaming. In this section, we present the HTTP streaming, which was proposed even before P2P video streaming was largely deployed [Carmel et al. 2002], but has been popularly used to address these issues on today's Internet.

4.1 Moving to Dynamic Adaptive Streaming over HTTP

Since invented in 1990, web browsers based on HTTP are becoming more and more popular for users to consume a large variety of multimedia contents. Recently, more complicated multimedia applications have seen rapid growth over HTTP, including video streaming.

4.1.1 *CDN and HTTP Streaming.* One thing for the rapid adoption of HTTP is that it can be extensively supported by the CDNs [Peng 2004], which deploy servers in multiple geographically diverse locations, distributed over multiple ISPs [Vakali and Pallis 2003]. CDN allow users to download the contents from servers close to them. User requests are redirected to the best available server based on proximity, server load, *etc.*. For video streaming, users observe higher quality experience by receiving a more reliable bandwidth from the CDN servers.

Alternatively to streaming, progressive download may be used for media delivery from standard HTTP Web servers. Clients that support HTTP can seek to positions in the media file by performing byte range requests to the Web server [Fielding et al. 1999]. As a result, CDN can be effectively used for high-quality TV content [Cahill and Sreenan 2004], Adhikari *et al.* have discovered that the leading on-

Table I. HTTP Streaming/DASH implementation by industry

Type	Implementation feature	Server	Client	Description File
Adobe Adaptive Streaming [Adobe 2011]	Presuming upon both RTMP and HTTP protocols but by implementing different server applications	Flash Media Server	Flash Media Player	Standardized formats of VP6 and H246 file types which are chopped into fragments and used from a specific manifest, the FMF which is just a similar format to the one of XML
Apple HTTP Live streaming [Apple 2011]	Applying DASH process in its simplest form by clipping the stream into smaller HTTP-based file downloads	General HTTP servers	QuickTime player and the iOS players	Apple's manifest reflected on a playlist which contains the list of available qualities – it is divided into smaller sub-playlists which include URLs for each M3U segment
Microsoft Live Smooth Streaming [Microsoft 2012]	Using Internet Information Services (IIS) Media Services	Microsoft IIS with the extension of Smooth Streaming	Microsoft Silverlight Player	Protected Interoperable File Format (PIFF) to convey the table of segments URLs to the client which contain audio and video material of fragmented MP4

demand Internet video streaming provider, Netflix which accounts for 29.7% of the peak downstream traffic in US, employs a blend of data centers and CDNs for content distribution. Torres *et al.* [Torres et al. 2011] have studied the metrics for the selection strategies in the YouTube CDN. Yin *et al.* [Yin et al. 2009] studied using a hybrid CDN and P2P architecture for video streaming, whose performance is guaranteed by the CDN while the cost is reduced by the peer contribution.

HTTP video streaming overcomes the problems in case of firewalls and NAT traversals (*e.g.*, UDP commonly used in P2P video streaming is likely to be blocked by the firewalls). It works by breaking the overall video stream into a sequence of small HTTP-based file downloads. HTTP progressive downloading [Rubio Romero 2011] is a key to HTTP video streaming – it allows users to download while the specific file is being viewed or generally played. Since the requests use standard HTTP transactions, HTTP video streaming is capable of delivering the contents to users over widely available CDNs we discussed above. Today, HTTP video streaming has been implemented by Microsoft, Google, Adobe in their corresponding products. Meanwhile, the representative online video providers, including Netflix, YouTube, Hulu, are all resort to HTTP to stream their videos to the users.

4.1.2 *HTTP Streaming and DASH.* In recent years, the Internet access has become a commodity on mobile devices, including the popular iOS and Android devices. The overall traffic caused by mobile devices is expected to grow to 6 Exabytes per Month by 2014 expanding [Cisco 2010].

The heterogeneous network and devices require that the video streaming be dynamical and adaptive. DASH (Dynamic Adaptive Streaming over HTTP) is a streaming technology which set off in 2010 developing in parallel with MPEG [(MPEG) 2010] and managed to standardize in 2011 with 3GPP [Stockhammer 2011]. In DASH, video segments are hosted on the conventional HTTP streaming servers, *along* with the media presentation description (MPD), which describes the relation of the segments and how they form a video presentation. Using the MPD, the clients request the segments for smooth playback of the sequence of segments, by adjusting bitrates or other attributes. To summarize, DASH defines a set of implementation protocols across the server, client and description files. In Table I, we present the popular DASH implementations from the industry.

4.2 Video Streaming over the Cloud

As HTTP streaming is increasingly employed by large content providers, more and more video streaming applications/systems have been invented based on the HTTP-browser paradigm, in which the UGC (user generated content) video sharing systems (*e.g.*, YouTube) have fundamentally changed the video

streaming landscape – users dynamically generate the video contents and an elastic video streaming service resource is in demand for the new HTTP streaming.

Since 2006, leading IT companies, such as Google, Amazon, and Microsoft, have started to build datacenters that are able to handle demand at scale. The highly centralized design of datacenters had catapulted the popularity of *cloud computing* since 2008, and had embraced a design philosophy that is exactly the opposite to that used by peer-to-peer systems. Attracted by the abundant networking resources in the cloud and the on-demand "pay-as-you-go" pricing model, many applications are heading to the cloud, including the video streaming service. For example, Netflix, one of the leading video streaming service providers, has been reported to resort to the cloud service [Cockcroft 2011]. As a result, the design of network topologies in datacenters has quickly become an emerging active area of research, drawing significant research attention since 2008. Such a paradigm shift to cloud computing, in the industry and academia alike, may have stymied research interests in peer-to-peer video streaming systems, even as peer-to-peer streaming systems, such as PPStream, are still being used on a daily basis by millions of users.

A cloud computing platform offers reliable, elastic and cost-effective resource provisioning, which has been dramatically changing the way of enabling scalable and dynamic network services. There have been pioneer studies on demand-driven resource provision; there have been also initial attempts leveraging cloud services to support media streaming applications, from both industry (*e.g.*, Netflix) and academia [Wu et al. 2011][Huang et al. 2011]. Content clouds let VoD providers pay by bytes for bandwidth resources, potentially leading to long-term cost savings given that renting a machine is relatively cheaper than owning one. VoD providers could save more than 30% bandwidth expense by migrating 40% traffic to the cloud. Short-term-wise, cloud-assisted deployments can well handle burst traffic with much lower cost as compared to over-provisioning in self-owned servers.

Fig. 7 shows a generic framework that facilitates the migration of existing live media streaming services to a cloud-assisted solution [Wang et al. 2012a]. The framework, consisting of a *cloud layer* and *user layer*, adaptively leases and adjusts cloud servers in a fine granularity to accommodate temporal and spatial dynamics of user demands. Upon receiving a user's subscription request, the cloud layer redirects this user to a selected cloud server, with the re-direction being transparent to the user. Given time-varying user demands, more server resources will be leased upon demand increase during peak times, or otherwise terminated. In essence, cloud serves a storage- and bandwidth-buffer for the user layer, which largely mitigate the impact of demand dynamics.

There are however a number of critical theoretical and practical issues to be addressed in this generic migration framework. First, the cloud servers have diverse capacities and lease prices; the lease duration is not infinitesimally short, either, *e.g.*, 1 hour for Amazon's EC2. As such, when being leased, the server and the pricing cannot be simply terminated at anytime. For a newly leased server, the configuration and boot up takes time, too, *e.g.*, 10-30 minutes for EC2. Though the cloud services are improving, given the hardware, software, and network constraints, such latencies can hardly be eliminated in the near future. Therefore, it must well predict when to lease a new servers to meet the ever-changing demands and when to terminate a server to minimize the lease costs [Niu et al. 2012].

These problems are further complicated given the global heterogeneous distributions of the cloud servers and that of the user demands. Today's live streaming applications have become highly globalized, with subscribers from all over the world. Such a globalization makes user behaviors and demands even more diverse and dynamic. For illustration, Fig. 6 shows a one-day normalized user request distribution of a representative channel (CCTV3) in the PPTV system, where the time shown on the x-axis is aligned to EST. It is easy to see that PPTV, though based on China, has attracted users from all over the world, and the peak time shifts from region to region, depending on the timezone. The impact of

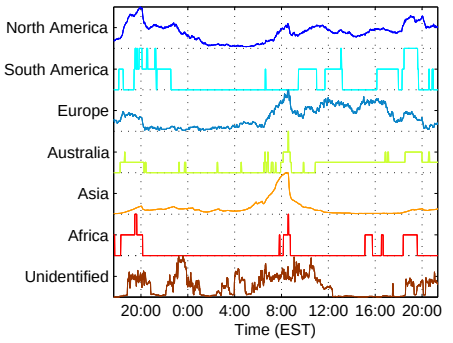

Fig. 6. One day user demand distribution of CCTV3 channel in PPTV.

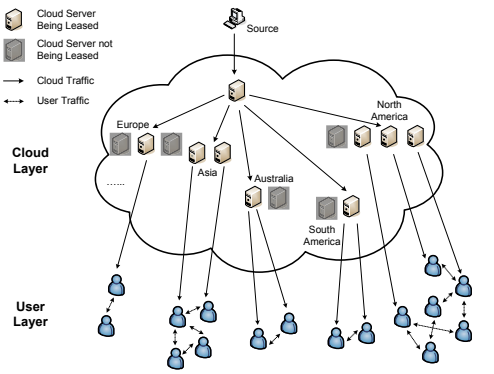

Fig. 7. A generic framework for cloud-assisted streaming.

such globalized demand turbulence has yet to be addressed for a single-datacenter-based cloud, and a distributed cloud would be a better solution.

Besides storage and network bandwidth, modern cloud providers have also enabled computation as a service. A cloud-based video proxy system is presented in [Zhao et al. 2012], which transcodes the original video in real time using a scalable codec based on H.264/SVC for streaming videos of various qualities. Offloading computation-intensive tasks from mobile devices is particularly attractive for mobile devices. To date, video compression on mobile devices remains a challenging task due to their limited computation power and energy. However, simply transferring raw video files to cloud server from compression and processing introduces huge energy consumption, which contradicts the benefit. The question then becomes how to leverage cloud server computation resources through careful partitioning between remote and local computation. Given that motion estimation is the most computation intensive task, accounting for 90% computation time, it is a natural target. One possibility is to make use of mesh-based motion estimation by uploading anchor frames and mesh nodes to cloud server for calculating motion vectors; the motion vectors are then pushed back to mobile devices for motion estimation of sub blocks and the rest video compression steps. By carefully choosing mesh structure, video compression energy consumption can be largely reduced on mobile devices.

4.3 The Effects of Social Media

Compared with the Internet ten years ago, networked services in the Web 2.0 era focus more on user experience, user participation and interaction with rich media; people are now actively engaged to be part of the new ecosystem, rather than passively receive information as in the past.

For streaming services, one prominent example reflecting this change is YouTube. Established in 2005, it is now serving well over 4 billion views a day, with most of the contents being generated by users. The sheer number of UGC objects is orders of magnitude higher than that of traditional movies or TV programs, and evolves rapidly. As of March 2013 [you 2013], every second, 1.2 hours worth of video is uploaded on YouTube by users around the world, averagely attracting almost 140 views for every person on earth. Earlier industry insiders estimated that YouTube spent over $1 million a day to pay for its server bandwidth. This has defeated any effort towards increasing server capacity and improving user experiences. Saxena *et al.* reveal that the average service delay of YouTube is nearly 6.5 seconds, which is much longer than the other measured sites [Saxena et al. 2008].

While peer-to-peer mechanisms would be a candidate to relieve such server bottleneck, the huge number of videos with highly skewed popularity implies that many of the peer-to-peer overlays will be

too small to function well. Moreover, user generated videos are generally short (70% are shorter than 1 minute, even though YouTube has largely relaxed the length limit [Cheng et al. 2012]), implying an overlay will suffer from an extremely high churn rate. They together make existing per-video based overlay design suboptimal, if not entirely inapplicable.

Fig. 8. A sample graph of YouTube videos and their links.

On the other hand, the UGC nature introduces new social relations and interactions among videos and users. In particular, there are interesting relations among the YouTube videos: the links to related videos generated by uploader's choices form a small-world network, as illustrated in Fig. 8 [Cheng et al. 2012]. This suggests that the videos have strong correlations with each other. A user often quickly loads another related video when finishing the previous one. An earlier work that explores such relationships is NetTube [Cheng and Liu 2009], which introduces an upper-layer overlay on top of the swarms of individual videos. In the upper-layer overlay, given a peer, neighborhood relations are established among all the swarms that contains this peer. This conceptual relation facilitates a peer to quickly locate the potential suppliers for the next video and enable a smooth transition. The relations also enables effective pre-fetching of videos. While the repository of short videos is large, the next video in YouTube is most likely confined by the related list of the current video. This list in general includes 20 videos at most. Therefore, even if the number of pre-fetched videos is small, the hit ratio of the next video could still be quite high. After multiple rounds, the rate can easily reach over 90% given the small world of the videos.

Migrating YouTube-like UGC services to cloud is non-trivial either. A unique and critical step here is to partition the contents and assign them into a number of cloud servers. Not only the loads of cloud servers have to be balanced like for traditional standalone videos, but also the relationships among the videos and users have to be preserved so as to promote access locality. There are some existing works trying to solve this problem [Pujol et al. 2010; Cheng and Liu 2011], yet an all-around optimal solution remains to be developed.

The latest development of general social network applications, in particular, Facebook and Twitter, have further changed the information distribution landscape and even people's daily life. These online Social Networking Services (SNS) directly connect people through cascaded relations, and information

thus spread much faster and more extensively than through conventional web portals or newsgroup services.

Such word-of-mouth spreading [Rodrigues et al. 2011] has also drastically expanding the ways of discovering the videos, beyond traditional web browsing and searching. There have been pioneer studies on information propagation over generic networks and, more recently, over social networks [Budak et al. 2011]. Yet their focuses have been largely on the conventional text or image objects and on their stationary coverage among users. The sheer and ever increasing data volume, the broad coverage, and the long access durations of video objects present more significant challenges than other types of objects to the SNS management, and to that of video sharing sites (VSSes). A measurement [Broxton et al. 2010] based on YouTube data showed that between April 2009 and March 2010, 25% of views on YouTube come from social sharing. Despite recent work towards this direction [Wang et al. 2012b], the characteristics of such requests from OSNs have yet to be comprehensively measured at large scales, so do video requests modeling and generation.

5. CONCLUDING REMARKS AND LESSONS LEARNED

In retrospect, the rise of research interests in video streaming systems was largely fueled by user demand and the increasingly abundant availability of bandwidth in the Internet, reflected both in the uplink bandwidth at end users (motivating the peer-to-peer philosophy), and in the downlink bandwidth at servers in the cloud datacenters and in Content Distribution Networks (CDNs).

The rise (and eventual fall) of peer-to-peer designs have reflected an interesting phenomenon that is worth noting. The major contributing factor to their ability to attract strong academic interests, in our opinion, was the fact that the design of peer-to-peer systems was able to start from a *clean slate*, in that it was not confined by any legacy protocols in the Internet. Due to the freedom of designing overlay topologies, the peer-to-peer paradigm was also amenable to theoretical treatments, from modeling to analyses. For example, the following are some technical challenges that have received heated discussions from researchers in peer-to-peer video streaming.

$\diamondsuit$ *Tree based vs. data driven, could there be any hybrid?* Both tree-based structured and pull-based structureless overlays have shown their success in practical deployment, and yet neither completely overcomes the challenges from the dynamic peer-to-peer environment. The selling point for pull-based systems is their simplicity, but they suffer from a latency-overhead trade-off [Venkataraman et al. 2006] [Zhang et al. 2005a]. If peers choose to send notifications for every segment arrival, then the overhead will increase. Periodical notifications containing buffer maps reduces the overhead, but at the expense of increasing the latencies. On the other hand, a tree-based system does not suffer from this trade-off, but has to face the inherent instability, maintenance overhead, and bandwidth underutilization. A natural question is therefore whether we can combine them to realize a hybrid overlay that is both efficient and robust.

The combination can be achieved in different dimensions. An example is Chunkyspread [Venkataraman et al. 2006], which splits a stream into distinct slices and transmits over separate but not necessarily disjoint trees. The participating peers also form a neighboring graph, and the degree in the graph is proportional to its desired transmission load. This hybrid design greatly simplifies the tree construction and maintenance, and largely retains its efficiency and achieves fine-grained control over load.

Another direction is a more explicit tree-bone based approach [Wang et al. 2007]. The trace studies have shown strong evidence that most of the data blocks delivered through a mesh-based data-driven overlay essentially follow a specific tree structure or a small set of trees. The similarity of the trees, defined as the fraction of the common links, can be as high as 70%. The overlay performance thus closely depends on the set of common internal peers and their organization. This suggests that, while main-

taining a prior topology for all the peers is costly, optimizing the organization for a core subset is worth consideration. In particular, if such a subset consists mainly of the stable peers, with others being organized through a mesh, we can expect high efficiency with low overhead and delay simultaneously.

♢ *Incentives and fairness*. Many research papers in peer-to-peer video streaming have made an implicit assumption that users can and are willing to collaborate. In reality, however, this is not always the case. Measurement studies have shown that, in some peer-to-peer video streaming systems, a small set of peers are requested to contribute 10 to 35 times more uploading bandwidth than downloading bandwidth. Such overhead will hinder any potential users from being cooperative. These autonomous users can be selfish and misbehave in order to maximize their benefits. As a result, there could be many free riders in a peer-to-peer system that either refuse to contribute or avoid contributing bandwidth, *e.g.*, in tree construction, always acting as a leaf by declaring a poor outbound bandwidth. This situation as a result, can seriously affect the overall service quality experienced by cooperative peers. Therefore, a proper incentive mechanism is critical to the performance of a peer-to-peer video streaming system.

Designing incentive mechanisms for video streaming system is more challenging than traditional file download applications, due to the real-time requirements. In particular, one solution in the file download context involves use of reputation based on past performance. However, this is feasible because the total time to download a file can often be long providing sufficient time to collect enough credits or build reputation. Further, file download users can tolerate slow download rates for a period of time by keeping the program running at the background. In contrast, users in video streaming applications stay for shorter times, and will simply leave the system when the playback quality is not satisfying. A micro-payment mechanism may be a good solution that enables video broadcast users to cooperate. However, this often asks for a centralized broker for coordination that can hinder the scalability of a peer-to-peer systems. Systems like Coopnet [Padmanabhan et al. 2002] assume each node contributes as much bandwidth as it receives - however this does not consider the heterogeneity in node bandwidth.

BitTorrent-like applications adopt a *tit-for-tat* strategy to solve the incentive problem. Tit-for-tat is a highly effective strategy in game theory originally proposed for the iterated prisoner's dilemma. The strategy works well in peer-to-peer file download because the segments of a file are downloaded independently. The basic idea is when a peer is not uploading in return to a peer's uploading, the application will choke the connection with the uncooperative peer and allocate this upload slot to a hopefully more cooperating peer. However, this approach does not trivially extend to video broadcast because of the timeliness requirements involved. The design of a scalable, light-weight incentive mechanism which fits the application of peer-to-peer video streaming system remains receiving discussion these days.

♢ *Extreme peer dynamics and flash crowd*. During a flash crowd, there is a large increase in the number of users joining the video streaming sessions in a short period of time. This poses challenges for a peer-to-peer video streaming system as it has to rapidly assimilate the new peers into the distribution structure without significantly impacting the video quality of existing and newly-arrived peers. The opposite situation is when a large number of users leave a video session during a short period of time. The peer-to-peer video streaming system has to repair the delivery structure to minimize the service interruption as well. During a high churn situation, users arrive and depart frequently, in which case the peer-to-peer video streaming system has to continue to adapt with the peer dynamics.

As discussed in [Sripanidkulchai et al. 2004], flash crowd and high churn situation are very common. The extreme scenario can be very difficult to handle. Consider the example of a popular concert broadcast that attracts 1 million users. If these users arrive within the the first 100 seconds of the concert, the peak arrival rate will be $10,000$ peers per second. If the video quality is not good for the initial period, a user is more likely to quit. This not only represents a failure of the system to provide service to this particular user, but also generates a peer departure event, thus more churn in the sys-

tem. Designing peer-to-peer video streaming system that is robust to extreme peer dynamics is also an interesting research topic.

However, whether the peer-to-peer design philosophy will thrive or survive hinges upon the crucial judgment of whether its advantages in substantial bandwidth savings outweigh its drawbacks, being so vulnerable to flash crowds and high turnover rates. Unfortunately, with recent downward trends in storage and bandwidth resource pricing, the momentum of the pendulum swing to the opposite side has accelerated. According to [Stoica 2010], bandwidth pricing of Content Distribution Networks (CDN) was observed to be dropping quickly every year, from 40 cents per GB in 2006 to less than 5 cents per GB in 2010. As a result, the streaming cost per hour had decreased 15%–35%, with a streaming cost of less than 3 cents per hour in 2010. The consequence of these observations was dramatically reduced distribution costs for content providers. For example, for paid content that is usually priced at $0.99 per episode, the distribution cost is less than 3% of the provider's total cost; for subscription-based premium content, it only costs $1.60 per month to stream content to an user watching 2 hours per day! What contributes to the majority of a content provider's revenue is the income from ad-supported premium content. The cost per thousand of ad impressions (CPM) for premium content has reached $20–$40, with a single ad covering the cost of an entire hour of streaming.

The ever decreasing cost of content delivery and the emergence of such ad-based business models have boosted the importance of *video quality*. Recent measurement results have shown that there is a crucial interplay between video quality and user engagement [Dobrian et al. 2011]. As a consequence, content providers, with an objective of generating more revenue, care more about the quality of their streaming service to maximize user engagement, than the cost of bandwidth.

By leasing storage and bandwidth resources to leading content providers such as NetFlix, cloud computing has emerged as the winner by purchasing resources at wholesale and selling them to cloud users at retail. As a result, HTTP/DASH streaming powered by cloud computing and datacenters, and the social media and social effects due to the highly-available online social network services , recently enjoyed a similar meteoric rise in popularity, attracting an enthusiastic level of research attention just like peer-to-peer systems did a decade ago.

REFERENCES

Last accessed, March 24, 2013. http://www.youtube.com/yt/press/statistics.html.

ADOBE. 2011. HTTP Dynamic Streaming.

AHLSWEDE, R., CAI, N., LI, S. R., AND YEUNG, R. W. 2000. Network Information Flow. *IEEE Trans. Inform. Theory 46,* 4, 1204–1216.

APPLE. 2011. HTTP Live Streaming Overview.

BHARADVAJ, H., JOSHI, A., AND AUEPHANWIRIYAKUL, S. 1998. An Active Transcoding Proxy to Support Mobile Web Access. In *Proc. of IEEE Symposium on Reliable Distributed Systems*.

BHARAMBE, A. R., HERLEY, C., AND PADMANABHAN, V. N. 2006. Analyzing and Improving a Bittorrent Networks Performance Mechanisms. In *Proc. of IEEE INFOCOM*.

BROXTON, T., INTERIAN, Y., VAVER, J., AND WATTENHOFER, M. 2010. Catching a viral video. In *Proc. The 10th IEEE International Conference on Data Mining (ICDM'10)*.

BUDAK, C., AGRAWAL, D., AND ABBADI, A. E. 2011. Limiting the spread of misinformation in social networks. In *Proc. The 20th International World Wide Web Conference (WWW'11)*.

CAHILL, A. J. AND SREENAN, C. J. 2004. An Efficient CDN Placement Algorithm for the Delivery of High-Quality TV Content. In *Proc. of ACM Multimedia*.

CARMEL, S., DABOOSH, T., REIFMAN, E., SHANI, N., ELIRAZ, Z., GINSBERG, D., AND AYAL, E. 2002. Network Media Streaming. US Patent 6,389,473.

CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., NANDI, A., ROWSTRON, A., AND SINGH, A. 2003. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*. 298–313.

CHEN, T. P.-C. AND CHEN, T. 2004. Fine-Grained Rate Shaping for Video Streaming over Wireless Networks. *EURASIP Journal on Applied Signal Processing 2004*, 176–191.

CHENG, X. AND LIU, J. 2009. Nettube: Exploring social networks for peer-to-peer short video sharing. In *Proc. IEEE INFOCOM'09*.

CHENG, X. AND LIU, J. 2011. Load-balanced migration of social media to content clouds. In *Proc. ACM NOSSDAV'11*.

CHENG, X., LIU, J., AND DALE, C. 2012. Understanding the characteristics of internet short video sharing: A youtube-based measurement study.

CHENGUANG, X., YINLONG, X., CHENG, Z., RUIZHE, W., AND QINGSHAN, W. 2007. On Network Coding Based Multirate Video Streaming in Directed Networks. In *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa*. IEEE, 332–339.

CHU, Y.-H., RAO, S. G., AND ZHANG, H. 2000. A Case for End System Multicast. In *Proc. ACM SIGMETRICS*. 1–12.

CISCO, C. V. N. I. 2010. Global Mobile Data Traffic Forecast Update, 2009–2014. *White Paper*.

COCKCROFT, A. 2011. Netflix in the Cloud.

DEERING, S. AND CHERITON, D. 1990. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transaction on Computer Systems 8,* 2, 85–110.

DIOT, C., LEVINE, B., LYLES, B., KASSEM, H., AND BALENSIEFEN, D. 2000. Deployment Issues for the IP Multicast Service and Architecture. *Network, IEEE 14,* 1, 78–88.

DOBRIAN, F., AWAN, A., JOSEPH, D., GANJAM, A., ZHAN, J., SEKAR, V., STOICA, I., AND ZHANG, H. 2011. Understanding the Impact of Video Quality on User Engagement. In *Proc. ACM SIGCOMM*. 362–373.

FENG, C., LI, B., AND LI, B. 2009. Understanding the Performance Gap between Pull-based Mesh Streaming Protocols and Fundamental Limits. In *Proc. IEEE INFOCOM*.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. 1999. Hypertext Transfer Protocol–HTTP/1.1.

FLOYD, S., HANDLEY, M., PADHYE, J., AND WIDMER, J. 2000. *Equation-Based Congestion Control for Unicast Applications*. Vol. 30. ACM.

FROSSARD, P. 2001. FEC Performance in Multimedia Streaming. *IEEE Communications Letters 5,* 3, 122–124.

GOYAL, V. K. 2001. Multiple Description Coding: Compression Meets the Network. *IEEE Signal Processing Magazine 18,* 5, 74–93.

HO, T., KOETTER, R., MEDARD, M., KARGER, D., AND EFFROS, M. 2003. The Benefits of Coding over Routing in a Randomized Setting. In *Proc. Int'l Symposium on Information Theory (ISIT)*.

HOSSEINI, M., AHMED, D., SHIRMOHAMMADI, S., AND GEORGANAS, N. 2007. A Survey of Application-Layer Multicast Protocols. *IEEE Communications Surveys Tutorials 9,* 3, 58 –74.

HUANG, Y., FU, T. Z., CHIU, D.-M., LUI, J. C., AND HUANG, C. 2008. Challenges, Design and Analysis of a Large-scale P2P-VoD System. In *Proc. ACM SIGCOMM*. 375–388.

HUANG, Z., MEI, C., LI, L.-E., AND WOO, T. 2011. Cloudstream: Delivering high-quality streaming videos through a cloud-based svc proxy. In *Proc. IEEE INFOCOM'11*.

JACOBS, S. AND ELEFTHERIADIS, A. 1998. Streaming Video Using Dynamic Rate Shaping and TCP Congestion Control. *Journal of Visual Communication and Image Representation 9,* 3, 211–222.

JANNOTTI, J., GIFFORD, D. K., JOHNSON, K. L., KAASHOEK, M. F., AND O'TOOLE, JR., J. W. 2000. Overcast: Reliable Multicasting with an Overlay Network. In *Proc. 4th Symposium on Operating System Design and Implementation (OSDI)*. Vol. 4. 1–14.

KALMAN, M., STEINBACH, E., AND GIROD, B. 2004. Adaptive Media Playout for Low-Delay Video Streaming over Error-Prone Channels. *IEEE Transactions on Circuits and Systems for Video Technology 14,* 6, 841–851.

KIM, T. AND AMMAR, M. H. 2001. A Comparison of Layering and Stream Replication Video Multicast Schemes. In *Proc. of ACM NOSSDAV*.

KOSTIĆ, D., RODRIGUEZ, A., ALBRECHT, J., AND VAHDAT, A. 2003. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*. 282–297.

KUMAR, R., LIU, Y., AND ROSS, K. 2007. Stochastic Fluid Theory for P2P Streaming Systems. In *Proc. of IEEE INFOCOM*.

KUNG, W.-Y., KIM, C.-S., AND KUO, C.-C. 2006. Spatial and Temporal Error Concealment Techniques for Video Transmission over Noisy Channels. *IEEE Transactions on Circuits and Systems for Video Technology 16,* 7, 789–803.

LEGOUT, A., URVOY-KELLER, G., AND MICHIARDI, P. 2006. Rarest First and Choke Algorithms Are Enough. In *Proc. of ACM IMC*.

LIU, Z., WU, C., LI, B., AND ZHAO, S. 2010. UUSee: Large-Scale Operational On-Demand Streaming with Random Network Coding. In *Proc. IEEE INFOCOM*.

MAGHAREI, N., REJAIE, R., AND GUO, Y. 2007. Mesh or Multiple-Tree: a Comparative Study of Live P2P Streaming Approaches. In *Proc. IEEE INFOCOM*.

MAJUMDA, A., SACHS, D. G., KOZINTSEV, I. V., RAMCHANDRAN, K., AND YEUNG, M. M. 2002. Multicast and Unicast Real-Time Video Streaming over Wireless LANs. *IEEE Transactions on Circuits and Systems for Video Technology 12,* 6, 524–534.

MCCANNE, S., JACOBSON, V., AND VETTERLI, M. 1996. Receiver-Driven Layered Multicast. In *ACM SIGCOMM Computer Communication Review*. Vol. 26. ACM, 117–130.

MICROSOFT. 2012. IIS Smooth Streaming Technical Overview.

(MPEG), I. J. S. W. . 2010. Dynamic adaptive streaming over HTTP.

NIU, D., XU, H., LI, B., AND ZHAO, S. 2012. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In *Proc. IEEE INFOCOM'12*.

OOI, W. T. 2005. Dagster: Contributor-Aware End-Host Multicast for Media Streaming in Heterogeneous Environment. In *Electronic Imaging 2005*. International Society for Optics and Photonics, 77–90.

PADMANABHAN, V. N., WANG, H. J., CHOU, P. A., AND SRIPANIDKULCHAI, K. 2002. Distributing Streaming Media Content Using Cooperative Networking. In *Proc. Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*.

PAI, V., KUMAR, K., TAMILMANI, K., SAMBAMURTHY, V., AND MOHR, A. 2005. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS)*. 127–140.

PARK, J.-S., GERLA, M., LUN, D. S., YI, Y., AND MEDARD, M. 2006. Codecast: a Network-Coding-Based Ad Hoc Multicast Protocol. *Wireless Communications, IEEE 13,* 5, 76–81.

PENG, G. 2004. CDN: Content Distribution Network. *arXiv preprint cs/0411069*.

PUJOL, J. M., ERRAMILLI, V., SIGANOS, G., YANG, X., LAOUTARIS, N., CHHABRA, P., AND RODRIGUEZ, P. 2010. The little engine(s) that could: Scaling online social networks. In *Proc. ACM SIGCOMM'10*.

RODRIGUES, T., BENEVENUTO, F., CHA, M., GUMMADI, K.-P., AND ALMEIDA, V. 2011. On word-of-mouth based discovery of the web. In *Proc. ACM IMC'11*.

ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., SCHOOLER, E., ET AL. 2002. SIP: Session Initiation Protocol. Tech. rep., RFC 3261, Internet Engineering Task Force.

RUBIO ROMERO, L. 2011. A Dynamic Adaptive HTTP Streaming Video Service for Google Android. Ph.D. thesis, KTH.

SAXENA, M., SHARAN, U., AND FAHMY, S. 2008. Analyzing video services in web 2.0: A global perspective. In *Proc. NOSSDAV'08*.

SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V. 1996. RFC 1889. *RTP: a transport protocol for real-time applications*.

SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V. 2003. RFC 3550 RTP: A Transport Protocol for Real-Time Applications.

SCHULZRINNE, H., RAO, A., AND LANPHIER, R. 1998. Real time streaming protocol (RTSP) RFC 2326. *IETF (April 1998)*.

SEFEROGLU, H. AND MARKOPOULOU, A. 2007. Opportunistic Network Coding for Video Streaming over Wireless. In *Packet Video 2007*. IEEE, 191–200.

SEN, S., REXFORD, J., AND TOWSLEY, D. 1999. Proxy Prefix Caching for Multimedia Streams. In *Proc. of IEEE INFOCOM*.

SRIPANIDKULCHAI, K., GANJAM, A., MAGGS, B., AND ZHANG, H. 2004. The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points. In *Proc. ACM SIGCOMM*.

STOCKHAMMER, T. 2011. Dynamic Adaptive Streaming over Http: Standards and Design Principles. In *Proc. of ACM Conference on Multimedia Systems*.

STOICA, I. 2010. It's Not the Cost, It's the Quality! In *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*.

TORRES, R., FINAMORE, A., KIM, J. R., MELLIA, M., MUNAFO, M. M., AND RAO, S. 2011. Dissecting Video Server Selection Strategies in the YouTube CDN. In *Proc. of IEEE ICDCS*.

VAKALI, A. AND PALLIS, G. 2003. Content Delivery Networks: Status and Trends. *IEEE Internet Computing 7,* 6, 68–74.

VENKATARAMAN, V., YOSHIDA, K., AND FRANCIS, P. 2006. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS)*. 2–11.

WANG, F., LIU, J., AND CHEN, M. 2012a. Calms: Cloud-assisted live media streaming for globalized demands with time/region diversities. In *Proc. IEEE INFOCOM'12*.

WANG, F., XIONG, Y., AND LIU, J. 2007. mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast. In *Proc. 27th IEEE International Conference on Distributed Computing Systems (ICDCS)*.

WANG, M. AND LI, B. 2007. R^2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming. *IEEE J. on Sel. Areas in Communications*.

WANG, Y., WENGER, S., WEN, J., AND KATSAGGELOS, A. K. 2000. Error Resilient Video Coding Techniques. *IEEE Signal Processing Magazine 17,* 4, 61–82.

WANG, Y. AND ZHU, Q.-F. 1998. Error Control and Concealment for Video Communication: a Review. *Proceedings of the IEEE 86,* 5, 974–997.

WANG, Z., SUN, L., CHEN, X., ZHU, W., LIU, J., CHEN, M., AND YANG, S. 2012b. Propagation-based Social-aware Replication for Social Video Contents. In *Proc. of ACM Multimedia*.

WU, D., HOU, Y., AND ZHANG, Y. 2000. Transporting real-time video over the internet: Challenges and approaches. *Proceedings of the IEEE 88,* 12, 1855–1877.

WU, D., HOU, Y., ZHU, W., ZHANG, Y.-Q., AND PEHA, J. M. 2001. Streaming video over the internet: approaches and directions.

WU, Y., WU, C., LI, B., QIU, X., AND LAU, F.-C. 2011. Cloudmedia: When cloud on demand meets video on demand. In *Proc. the 21st IEEE International Conference on Distributed Computing Systems (ICDCS'11)*.

YIN, H., LIU, X., ZHAN, T., SEKAR, V., QIU, F., LIN, C., ZHANG, H., AND LI, B. 2009. Design and Deployment of a Hybrid Cdn-P2P System for Live Video Streaming: Experiences With LiveSky. In *Proc. of ACM Multimedia*.

ZHANG, M., LUO, J.-G., ZHAO, L., , AND YANG, S.-Q. 2005a. A Peer-to-Peer Network for Live Media Streaming - Using a Push-Pull Approach. In *Proc. ACM Multimedia*.

ZHANG, M., ZHANG, Q., SUN, L., AND YANG, S. 2007. Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? *Selected Areas in Communications, IEEE Journal on 25,* 9, 1678–1694.

ZHANG, X., LIU, J., LI, B., AND YUM, Y.-S. 2005b. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *Proc. IEEE INFOCOM*. Vol. 3. 2102– 2111.

ZHAO, Y., ZHANG, L., MA, X., LIU, J., AND JIANG, H. 2012. Came: Cloud-assisted motion estimation for mobile video compression and transmission. In *Proc. ACM NOSSDAV'12*.