Rise and Fall of the Peer-to-Peer Empire

Baochun Li, Yuan Feng
Department of Electrical and Computer Engineering
University of Toronto
{bli, yfeng}@eecg.toronto.edu

Bo Li
Department of Computer Science
Hong Kong University of Science and Technology
bli@cse.ust.hk

***Abstract*—The essence of the peer-to-peer design philosophy is to design protocols for end hosts, or "peers," to work in collaboration to achieve a certain design objective, such as the sharing of a large file. From a theoretical perspective, it has been recognized that the peer-to-peer design paradigm resembles gossip protocols, and with appropriate algorithmic design, it maximizes the network flow rates in multicast sessions.**

Over the past ten years, research on peer-to-peer computing and systems, a unique and intriguing category of distributed systems, has received a tremendous amount of research attention from academia and industry alike. Peer-to-peer computing eventually culminated in a number of successful commercial systems, showing the viability of their design philosophy in the Internet. The peer-to-peer design paradigm has pushed all design choices of innovative protocols to the edge of the Internet, and in most cases to end hosts themselves. It represents one of the best incarnation of the end-to-end argument, one of the frequently disputed design philosophies that guided the design of the Internet. Yet, research on peer-to-peer computing has recently receded from the spotlight, and suffered from a precipitous fall that was as dramatic as its meteoric rise to the culmination of its popularity. This article presents a cursory glimpse of existing results over the past ten years in peer-to-peer computing, with a particular focus on understanding what has stimulated its rise in popularity, what has contributed to its commercial success, and eventually, what has led to its precipitous fall in research attention. Our insights in this article may be beneficial when we develop our thoughts on the design paradigm of cloud computing.

I. Introduction

Over the past ten years, research on peer-to-peer computing has received a tremendous amount of research attention from academia and industry alike. The essence of the peer-to-peer design philosophy is to design protocols to be implemented only on end hosts, or *peers,* that reside at the edge of the Internet, without the need of new protocols on switches and routers in the Internet core. As such, a peer-to-peer protocol design is simply a unique and intriguing embodiment of a distributed system design, in which peers collaborate to achieve an objective, such as the speedy transfer of a large file from a source to multiple destination peers. In peer-to-peer computing, peers organize themselves in *overlay* topologies, in which packet transmission on each of the overlay links uses standard Internet protocols, such as TCP or UDP.

One of the most prominent design philosophies that peer-to-peer designs use is called "gossiping." With gossiping, peer-to-peer file sharing systems, such as BitTorrent, allow peers to collaborate with one another so that large files can be distributed from one peer to a large number of receivers. Peer-to-peer file sharing systems adopt a simple design philosophy: a file is divided into *blocks*, and peers connect with one another in a random mesh topology, exchanging these blocks using random gossiping. Just as its name suggests, in gossiping each peer transmits a subset of the blocks it has obtained to a subset of its neighbors that are selected using randomized algorithms. Such random gossiping on random mesh topologies is simple to implement, resilient to the level of volatility caused by peer arrivals and departures, and has been shown to achieve good performance from a theoretical perspective. In essence, with random gossiping, multiple blocks are spread across a group of participating peers.

Research on peer-to-peer computing has started about ten years ago, and since its inception, it has gone through three major stages of evolutionary progress. The earliest stage of peer-to-peer computing research focused on efficient search protocols, either by using a distributed hash table [1]–[3], or by using a gossip protocol. The second stage started around 2004, when the research

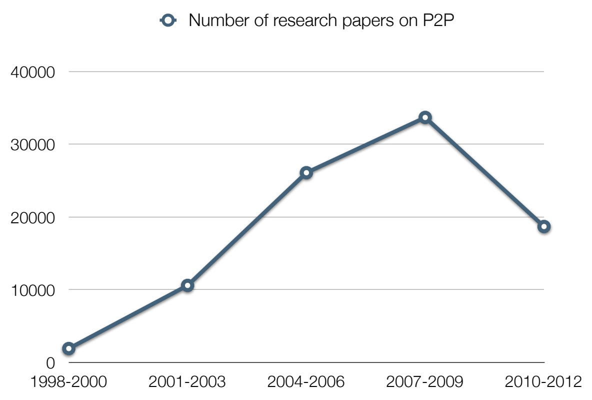

Fig. 1. Rise and fall of the popularity of peer-to-peer as a research topic. Data points are drawn from Advanced Google Scholar Search, with the term "P2P" in the paper metadata.

focus on peer-to-peer computing has gradually shifted towards optimizing protocols to transfer large files. The final stage started around 2007, when live and on-demand streaming protocols have become the topic that attracted passionate research attention. Naturally, these stages have substantial overlaps in time. For example, application-layer multicast protocols [4], [5] have started to receive attention in 2000, and it is designed with delays in mind, which is only considered useful when media streaming applications are considered.

Regardless of the research objectives, the peer-to-peer (P2P) design paradigm has pushed all design choices of innovative protocols to the edge of the Internet and to end hosts themselves. It represents one of the best incarnation of the end-to-end argument [6], one of the frequently disputed design philosophies that guided the design of the Internet. Peer-to-peer computing eventually culminated in a number of successful commercial systems, showing the viability of their design philosophy. Yet, research on peer-to-peer computing has recently receded from the spotlight, and suffered from a precipitous fall that was as dramatic as its meteoric rise to the culmination of its popularity. Fig. 1 has shown the rise and fall of its popularity using a rather crude method, counting the number of papers with the term "P2P" in its metadata, over the past ten years.

Another way of examining the popularity of a research topic is to look at new workshops, symposiums, and journals created to accommodate papers on the topic. With respect to peer-to-peer research, one such workshop may be worth paying our attention for a moment: the International Workshop on Peer-to-Peer Systems (IPTPS). The workshop was launched by a group of researchers from MIT, Microsoft Research, and University of California, Berkeley in March 2002, in order to provide a rather intimate forum for researchers interested in peer-to-peer computing to discuss its state-of-the-art challenges. It has become an annual event, and attendance is by invitation only during the first several years of its running. Yet, in 2011 the annual workshop has been canceled, presumably due to a lack of interest in the subject topic.

Why do we observe such a meteoric rise and an equally dramatic fall of research interests in peer-to-peer computing? In this article, we attempt to offer a cursory glimpse of the past ten years of peer-to-peer research, and show what has contributed to its rise in popularity in academic research, what has been key elements that promoted its commercial success, and what has led to its precipitous fall in research attention. As all published papers in the IPTPS workshop have been made available online in its website, we present our views on how research interests in peer-to-peer computing evolve over the decade, *through the lens* of papers published in this workshop. Observations we have made in this article may also be insightful when we develop our thoughts on cloud computing research.

II. Examining the Evolution of Peer-to-Peer Research through the lens of IPTPS

The International Workshop on Peer-to-Peer Systems (IPTPS), an annual event that was initially launched in 2002 and held till 2010, is not just one of the many conferences and workshops dedicated to research on peer-to-peer systems: it is a workshop with a large number of highly cited papers, many of which had a long lasting impact in peer-to-peer systems research. As such, we believe that an analysis of published papers in this workshop can be used to guide our understanding on what catapulted this research area into the spotlight, and what ultimately led to a dramatic fall of research interests. With this objective in mind, we first attempt to thoroughly investigate all the papers published in this workshop, and to make our observations along the way.

We believe that one of the earliest and most important ("killer") applications that motivated a peer-to-peer system design is *file sharing*. Users very often need to share files with others — such as friends in a social networking setting — as conveniently and as efficiently as possible. These files are typically large in their sizes, such as a song ten years ago, or a video in recent years. Two fundamental issues arise from such a need of sharing large files: we first need to have an effective way to search for their availability, and then an efficient mechanism to distribute the file from its source to all the users who may

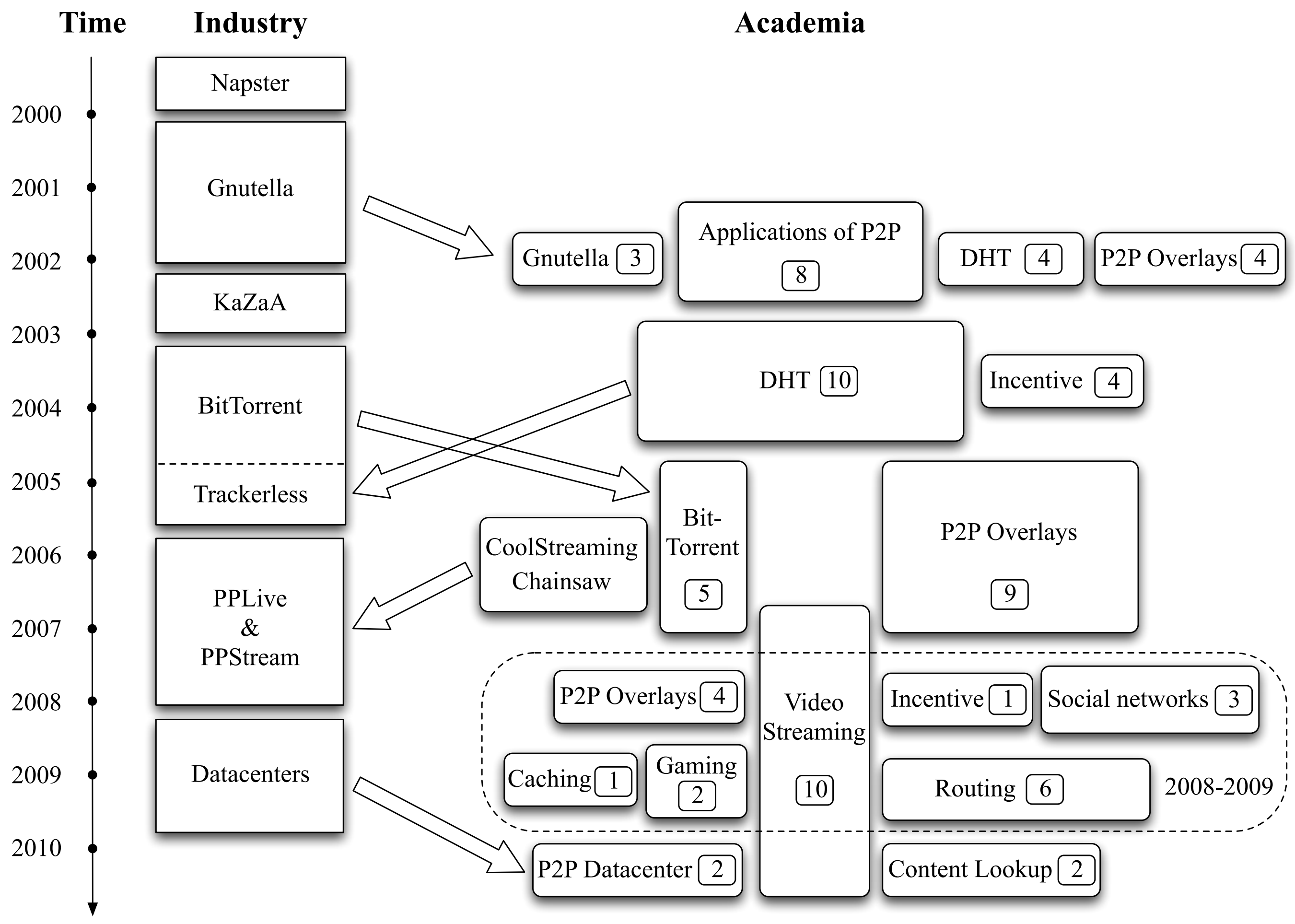

Fig. 2. The evolution of peer-to-peer research through the lens of IPTPS, where each topic is labeled with the number of papers published in the workshop (in rounded rectangles).

be interested. The former has eventually led to research on structured peer-to-peer search based on distributed hash tables (DHTs), and the latter has motivated the line of research on application-layer multicast, gossip protocols, and analysis of BitTorrent-like protocols.

With the advent of Napster and Gnutella from 1998 to 2001, the industry was the first that brought design principles of peer-to-peer systems to the attention of academic researchers. Both Napster and Gnutella were designed for file sharing [7]. Napster was designed to search for files in a centralized fashion, while Gnutella resorted to an expanded-ring search in an unstructured overlay topology. Each link in the overlay topology corresponds to the notion of a "pointer" to the identity of a different end host, including its IP address and port number. Once the requested file is found in an end host, the file can be transferred in a peer-to-peer fashion to the host that requested it, without the need of uploading it to any of the servers.

The inaugural IPTPS workshop was held in 2002, which coincided with a rapid increase in research attention on peer-to-peer systems, especially on *structured* search using distributed hash tables and application-layer multicast. Research on distributed hash tables sought to improve the search performance without using centralized servers, and research on application-layer multicast attempted to improve the efficiency when the need arose to multicast a file from one source to multiple receivers.

With the success of peer-to-peer systems in the industry, most notably Gnutella, interests in unstructured overlays were evident in the first IPTPS workshop, with an objective of analyzing and improving Gnutalla in the context of file sharing. Examples include possible solutions to increase its scalability [8], to discover its overlay network topology through analysis [9], and to improve the peer selection algorithm based on machine learning [10].

Without a surprise, since structured search was first proposed in 2001, quite a number of papers in IPTPS 2002 were investigating more efficient ways to search for replicas in structured search, especially when peers are highly dynamic [11]–[16]. In addition, research on structured search has also focused on its potential problems [17] and open questions [18].

Beyond the focus on structured and unstructured search, it appears that academic researchers who published in IPTPS 2002 were mainly interested in possible applications of the peer-to-peer system design. These applications include data replication, real-world measurement proxies, scientific data sharing collaborations, DHT-supported peer-to-peer DNS, and stenographic storage [19]–[24].

Structured search using distributed hash tables (DHTs) has also been receiving strong research attention at subsequent IPTPS workshops from 2003 to 2004 — it certainly appears that academic researchers loved to analyze and improve DHTs! Topics of particular interest include efficient content lookup protocols [25]–[34], measurement studies of DHT-based file sharing systems [35], [36], as well as load balancing algorithms [37]–[39]. In addition, the challenge of providing sufficient incentives for peers to contribute has become a topic attracting strong interests, as a number of mechanisms were proposed to prevent collusions among peers and to discourage "free-riders" [40]–[43].

The industry (including the open-source community), on the other hand, has not been staying around idling. After a few projects focused on improving Gnutella (*e.g.*, KaZaA), it was felt that the Gnutella family of protocols did not scale well to files with large sizes, as a file can only be downloaded from one peer. Bram Cohen believed that, in order to improve downloading performance, it was important to download a large file from a large number of peers, say, 30–50. Cohen then implemented BitTorrent with a few thousand lines of Python code, and fine tuned the BitTorrent protocol by releasing it for the world to use. By the end of 2004, BitTorrent was running on an estimated 10 million end hosts.

Initial implementations of BitTorrent required centralized *trackers* to keep track of the identities of peers, and each bundle of files was associated with its own `.torrent` file that contains sufficient metadata to locate other peers interested in the same bundle, with the assistance of trackers. As such, BitTorrent is designed to maximize downloading performance in file sharing, and the efficiency of search is relegated to other mechanisms, such as a website with indexed `.torrent` files. By using centralized servers for search and for bootstrapping, BitTorrent has returned to the legacy of Napster (which used servers to find files), and has focused entirely on the performance of downloading files. Later implementations of BitTorrent added support for *trackerless* operations, which used distributed hash tables instead of trackers. Yet, trackerless operations were not often used: even today, BitTorrent users continue to use trackers for locating other peers interested in the same bundle of files.

Thanks to the enormous popularity of BitTorrent in attracting real-world file sharing users, it was no surprise that it has also attracted strong research interests from academia. Qiu *et al.*, in their SIGCOMM 2004 paper [44], first analyzed the performance of BitTorrent using a stochastic fluid model. BitTorrent has also become an active research topic in papers published in IPTPS from 2005 to 2007. As examples, BitTorrent was extensively measured and evaluated, especially with respect to its performance in the presence of a "flash crowd" — a large number of peers arriving at the same time [45]. It was also extensively modelled and analyzed, focusing on its robustness against selfish peers that are not willing to contribute or are downloading more than their allocated quota [46]–[48]. BitTorrent's incentive mechanism, called Tit-for-Tat, was also believed by academic researchers to be quite crude, and was sought to be improved [49].

As the industry (and later academia) was keenly interested in improving the real-world performance of file sharing, and as trackerless operations were rarely used in BitTorrent, the perennial interest in structured search continued in academic research on peer-to-peer systems, from 2005 to 2006. Through the lens of IPTPS, there exist papers that attempted to improve the performance of structured search by using square root topologies [50], [51], or by using hybrid designs that inherited the benefits of both structured and unstructured search [52]–[54]. Disadvantages of unstructured overlays have also been thoroughly analyzed and understood [55], well after the industry shifted its focus to the performance of distributing large files, rather than that of search.

Perhaps the best example of the values of academic research in peer-to-peer systems came 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 media streaming, rather than file sharing. Streaming protocols are to be designed under a timing constraint: if media data blocks do not arrive in time, they are not useful when it comes to media playback. By gradually revising application-layer multicast protocols — first proposed in academic papers in 2000-2001 [4], [5] — so that they became less rigid and more flexible, design clues from BitTorrent were more and more evident.

In their paper published in IPTPS 2005, Pai *et al.* [56]

proposed *Chainsaw*; and in their paper published in INFOCOM 2005, Zhang *et al.* [57] proposed *CoolStreaming*. CoolStreaming and Chainsaw had independently presented a peer-to-peer system design that supported live media streaming: by following the BitTorrent design philosophy that randomly "pulls" from multiple peers in a mesh topology, the notion of application-layer multicast trees was completely eliminated, and was replaced by a *pull-based* design in a random mesh topology. In addition to protocol designs, both Chainsaw and CoolStreaming had presented their own prototype implementations, with evaluations on PlanetLab. In 2006-2007, such a pull-based random mesh design spearheaded by CoolStreaming and Chainsaw was adopted by a wide range of media streaming products in the industry. Most notably, they include PPLive [58] and PPStream, both of which were start-up companies in China at the time.

The shift of research interest from file sharing to media streaming was evident in the peer-to-peer research community. From IPTPS 2007 to IPTPS 2009, a large number of papers have been published, on both theoretical analysis and real-world measurement studies of peer-to-peer media streaming systems [59]–[68]. These papers naturally progressed on their scope and complexity, from the distribution of a single media stream to that of multiple concurrent streams with random seek.

Beyond the core research interest on peer-to-peer streaming systems, topics of papers published in IPTPS tend to become more diverse since 2008. There were papers that continued to investigate peer-to-peer overlay structures [69]–[72], incentive mechanisms [73], caching mechanisms [74], routing algorithms [75]–[80], analyses and measurements of BitTorrent systems [81]–[86], peer-to-peer social networks [87]–[89], peer-to-peer games [90], [91], and lookup algorithms in hybrid peer-to-peer search [92], [93].

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. The design philosophy in cloud computing naturally endorsed the need for central management and control, the need for performance and scalability, as well as the notion of *utility computing,* where computing services are treated as utilities that can be tapped into on an on-demand basis with "pay-as-you-go" pricing. 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 systems, even as peer-to-peer systems are still being used on a daily basis by millions of users. Such a trend was also reflected by the last IPTPS workshop in 2010, in which two papers were related to the interplay between peer-to-peer systems and cloud computing [94], [95], and its Call for Papers explicitly redefined *peer-to-peer systems* to be "large-scale distributed systems that are mostly decentralized, are self-organizing, and might or might not include resources from multiple administrative domains." This marked a dramatic departure from the original meaning of the term.

As a concise summary of our observations in this section, Fig. 2 has shown a graphical illustration of the evolution of research topics related to peer-to-peer systems, through the lens of the IPTPS workshop since 2002. It has also included an illustration of how work in the industry has affected academic research, and vice versa in the case of peer-to-peer media streaming.

III. From Multicast Trees to Gossiping — From Streams to Torrents

We have just walked through representative papers in IPTPS, a visible workshop on peer-to-peer systems research. With our investigation of how peer-to-peer research evolved through the lens of IPTPS, we have seen that it was typical for real-world systems coming from the industry to affect the path of academic research in a significant way. Yet, it may be a rare occurrence that results from academic research has led to production-quality implementations in the industry.

We would now like to more carefully examine a particular topic of research interest, *peer-to-peer media streaming*, under a "magnifying glass." We have chosen to study this topic since it was one of the few in which academic research had led to a number of success stories with start-up companies in the industry. By studying the evolution of research on this topic, we shall observe how industry work has influenced the direction of academic research, and in return, how academic results have affected industry products in a significant way, closing a "feedback loop."

The problem of peer-to-peer media streaming is, in essence, similar to the problem of file sharing. Fundamentally, it is a problem of *multicast*, in that a media stream is to be distributed from a source peer to a number of receiving peers. With live media streaming, it is desirable that the stream is received by the receiving peers with minimized delays, in that the time difference between the occurrence of a live event in the playback at the source and at a receiver is as small as possible. As compared to file sharing, streaming has a rather strict timing constraint: media blocks not arriving in time will be discarded during playback.

Since both file sharing and media streaming can be fundamentally represented by a multicast problem, the notion of application-layer multicast was first proposed by Chu *et al.* [4] as a reasonable application-layer replacement for IP multicast, and has since become an active research topic in peer-to-peer systems. 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 [96].

Chu *et al.* [4] have shown that application-layer multicast has the potential to perform reasonably well as compared to IP multicast, incurring a low delay and a reasonable amount of bandwidth penalty. Still, in initial proposals of application-layer multicast, a single multicast tree would be constructed for each multicast session (corresponding to one file to be shared or one live media stream). For example, Overcast [97] 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. 3, is its simplicity.

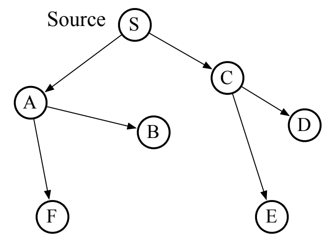

Fig. 3. Application-layer multicast with a single tree.

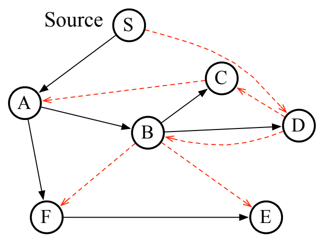

Fig. 4. Application-layer multicast with multiple trees.

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. 3, 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. It is intuitive to see that such a simple design is not robust against peer departures.

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 [98], or a mesh overlay on top of a tree [99]. Fig. 4 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.

Around the same time, Bram Cohen's BitTorrent has become a game changer in the industry when it comes to maximizing the performance of file sharing. The basic concepts are rather simple, perhaps more so than application-layer multicast: it makes sense to break a large file into smaller pieces, and disseminate each piece along an arbitrarily selected path to all the receivers. For each piece, it naturally follows its own multicast tree; but since there are a large number of pieces in a file, it is impossible to explicitly manage so many trees at the same time.

As we have elaborated in the previous section, the design philosophy in BitTorrent has converged with academic solutions to application-layer multicast, and a pull-based protocol on a random mesh topology emerged, independently discovered in Chainsaw [56] and CoolStreaming [57]. 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.

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. To show the gossiping effect a bit more formally, consider a network with n peers who wish to receive a file, divided into k blocks. If time can be divided into "rounds," each peer uploads to a few neighbors in a random mesh topology in each round, selected randomly. How many rounds are needed for all the peers to receive the file?

As an interesting problem that can be rigorously defined and analyzed, research results on the performance of gossip protocols exist without a doubt. Pittel [100] showed that, if $k = 1$, *i.e.,* if the file is not subdivided, and if each peer who has already received the file communicates it to a peer chosen at random, it takes $\log_2 n + \log n + O(1)$ rounds for the file to reach all n peers. Sanghavi *et al.* [101] were among the first to study the time needed to spread multiple blocks with a gossip protocol. In their analysis, in each round, each peer communicates with another peer chosen uniformly at random from the entire network, and each peer can upload at most one of the blocks it possesses. With such a model, even if centralized block scheduling is allowed, at least $k + \log_2 n$ rounds are needed to disseminate all k blocks from a single source to all n peers [102], [103]. The intuition is that it takes at least k rounds for the source to issue the last block, and a further $\log_2 n$ rounds for that block to reach all n peers. Sanghavi *et al.* [101] showed that, with a *decentralized* block selection protocol, one can finish distributing k blocks from a single source to n peers in $9(k+\log n)$ time with high probability for a large number of peers.

Following the gossiping philosophy to distribute each of the data blocks to all the peers in the multicast session, CoolStreaming and Chainsaw 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 [58]). Even though the gossiping — or equivalently, the BitTorrent — 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. This analogy makes it ideal to call such a design "torrents": every time we think of it, we marvel at the ingenuity of Bram Cohen on not only designing and implementing BitTorrent, but also naming it.

IV. Network Coding: Savior or Fiction?

Since 2005, the use of *network coding* has emerged as a potentially exciting research topic in peer-to-peer systems. Network coding, first proposed in the information theory community in 2000 [104], 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.* [105] 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 systems, in which peers, as end hosts, are able to forward, replicate, and *code* incoming data blocks, in both file sharing and media streaming scenarios. But will the use of network coding improve the performance, as compared to, say, Bram Cohen's BitTorrent?

A. File Sharing: Network Coding vs. BitTorrent

Gkantsidis *et al.* [106] was the first to consider random network coding as a substitute for peer-to-peer file sharing based on exchanges of individual blocks (*e.g.,* BitTorrent). It is argued in [106] that, if peers can linearly combine all the blocks it has already received so far using randomly generated coding coefficients, and then transmit such coded blocks to other peers, the amount of time required to distribute a large file to all the peers in the network may be reduced.

To briefly illustrate the idea proposed in [106], a simple example of distributing two data blocks is given in Fig. 5, where B has received blocks B_1 and B_2, and D has received block B_2. Although both B and D can serve E at this point, they may end up transmitting the same block B_2 to E without network coding, and without explicit coordination via information exchanges, as proposed in BitTorrent. In this case, the upload bandwidth of B may be wasted. With the use of random network coding, however, B can transmit to E a coded block $c_1^E B_1 + c_2^E B_2$, with coefficients c_1^E and c_2^E randomly chosen. E can then solve for B_1 using the received blocks B_2 and $c_1^E B_1 + c_2^E B_2$.

Similarly, with neither network coding nor periodic information exchanges, B may transmit block B_1 to F, who has it already. With random network coding, B can instead transmit to F another randomly encoded block $c_1^F B_1 + c_2^F B_2$, which is always useful to peer F if c_1^F and c_2^F are appropriately chosen.

Intuitively, at least on paper, the use of random network coding has increased the diversity of blocks being transmitted, and has eliminated the need for periodic information exchanges. Indeed, in [106], random network coding was to be used for file sharing, replacing BitTorrent. Gkantsidis *et al.* has made the claim that "the performance benefits provided by network coding

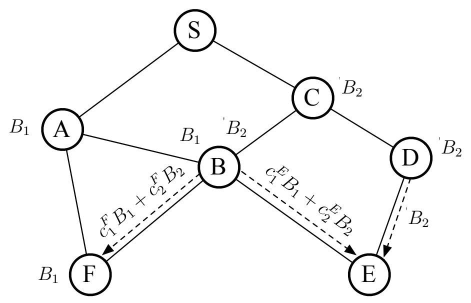

Fig. 5. An example of distributing two blocks B_1 and B_2 with network coding. S is the source peer.

in terms of throughput can be more than 2-3 times better compared to transmitting unencoded blocks." If their statement is to be believed, one may conclude that network coding does offer significant advantages as compared to BitTorrent.

Nevertheless, the claims were not viewed to be sufficiently conclusive, as they were based on simulation studies, rather than real-world implementations. In his 2005 rebuttal, Bram Cohen expressed the following opinion in his blog[1]: "Badly missing from this paper is back-of-the-envelope calculations about ... on-the-wire overhead, CPU usage, memory usage, and disk access time." Although his opinions were not based on any specific experiments, it has nonetheless raised reasonable concerns about the feasibility of implementing network coding, especially with respect to the additional computational overhead of coding operations.

It turns out that Cohen was, at least partially, right. the computational complexity of network coding escalates with an increasing number of blocks. To manage such complexity, it has been proposed [107] that blocks in a file be divided into multiple *generations*, and network coding is only performed within the same generation. To be more specific, the original file with F bytes is divided into G generations, each of which is further divided into m blocks, referred to as the *generation size*. There are a total of $M = G \cdot m$ original blocks, each with a size of $k = F/M$ bytes.

With the idea of dividing a file into generations, we are now ready to present in a slightly more formal way how random network coding can be used in file sharing. When the file is to be distributed, random network coding is applied across the blocks within a generation, say generation i, containing m original blocks $\mathbf{B^{(i)}} = [B_1^i, B_2^i, \ldots, B_m^i]$. On the source, a coded block b from this generation is a linear combination of these original blocks in the Galois field GF(2^q). Network coding is, of course, not limited to the source: if a peer (including the source) possesses l ($l \leq m$) coded blocks $[b_1^i, b_2^i, \ldots, b_l^i]$ of generation i, when the need arises to serve a new coded block to a neighbor p, it independently and randomly chooses a set of coding coefficients $[c_1^p, c_2^p, \ldots, c_l^p]$ in the Galois field GF(2^q), and encodes all the blocks of generation i it possesses to produce one coded block x of k bytes: $x = \sum_{j=1}^{l} c_j^p \cdot b_j^i$. For the benefit of successful decoding at a receiver, a coded block x is self-contained, in that coding coefficients used to encode *original blocks* to x are embedded in its header. As soon as a peer has received a total of m coded blocks from generation i that are linearly independent — $\mathbf{x} = [x_1^i, x_2^i, \ldots, x_m^i]$, it will be able to recover all original blocks in this generation with Gaussian Elimination, taking advantage of coding coefficients embedded in each of the m coded blocks received.

Perhaps in response to the increased skepticism with respect to the practicality of using network coding in peer-to-peer file sharing, in IPTPS 2006 [108] and IMC 2006 [109], Ghantsidis *et al.* sought to demonstrate the feasibility of network coding with a real-world implementation in C#. In their work, file sharing with network coding has been implemented and used to establish a session involving 100 peers across the Internet. With its experiments in a long-lived multicast session lasting around 38 hours to distribute a 4.3 GB file, the paper has concluded that "network coding incurs little overhead, both in terms of CPU and I/O, and it results in smooth and fast downloads." In particular, with respect to the computational overhead of network coding, the conclusion was drawn from the observation that each of the clients only consumes about 20% – 40% of its CPU throughout the session.

Albeit promising, this work has failed to compare the performance of its network coding implementation with a vanilla BitTorrent implementation. Doubts may be raised even without Cohen's response to the paper. Intuitively, since random network coding can only be performed on blocks within the same generation, block reconciliation — in the form of periodic information exchanges between a pair of peers — may still be necessary across the boundary of generations. In other words, rather than collecting fine-grained blocks, now a receiver only needs to collect all the distinct coarse-grained generations that constitute the file to be distributed. As such, network coding mitigates — but not completely solved — the problem of locating rare portions of a file that may be less popular. In practice, such a need for reconciliation, even at the coarser granularity of generations, may negatively affect the advantage of network coding. In contrast, BitTorrent is designed to download the rarest blocks first (with the aid of frequent

[1]Reference: http://bramcohen.livejournal.com/20140.html.

exchanges of buffer states among peers), in the hope of mitigating some of the adverse effects of locating rare blocks as the file download approaches completion. The additional computational complexity incurred by random network coding may not be well justified.

B. Streaming: Network Coding vs. CoolStreaming

As we have briefly alluded, in pull-based peer-to-peer live streaming systems (*e.g.* CoolStreaming [57]), 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 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.

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 [110], 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.* [111] 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, such as CoolStreaming. 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. 6, 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

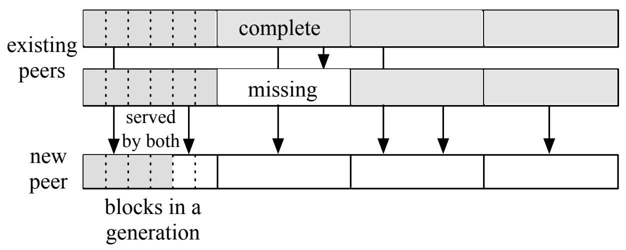

Fig. 6. Peer-to-peer live streaming with the use of random network coding: multiple existing peers are able to collaborate and serve coded blocks within the same generation to a new peer joining the session, minimizing its initial buffer delay.

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.

With theoretical analysis, Feng *et al.* [112] have shown that the design principles in R^2 have led to much better performance than live streaming protocols without network coding, especially in extreme scenarios such as "flash crowd" sessions where a large number of peers arrive in a short period of time.

But if past experience is of any value, a better performance in theory may not translate well to a competitive edge in practice. Fortunately, in 2010, Liu *et al.* [113] have first reported the use of random network coding in a production-quality on-demand peer-to-peer media streaming system that has been in operational use by UUSee Inc., a start-up company based in China. An excellent level of real-world streaming performance has been observed, based on measurement studies using 200 GB worth of operational traces. 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 context of peer-to-peer streaming systems, network coding has the true potential to deal a winning hand.

V. To Infinity and Beyond

In retrospect, the meteoric rise of research interests in peer-to-peer systems was largely fuelled by real-world peer-to-peer systems from the industry, most notably Gnutella, as well as by the need of millions of users to share large files in a peer-to-peer fashion. The essence of peer-to-peer systems is to take full advantage of both storage and bandwidth resources on end hosts, realizing potentially substantial resource savings on dedicated servers. For example, peer-to-peer live streaming systems were designed to utilize the upload bandwidth of end hosts to alleviate bandwidth demand on servers in Content Distribution Networks (CDNs). Peer-to-peer file sharing, on the other hand, utilized peer upload bandwidth to improve the downloading performance.

The other major contributing factor to the 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 modelling to analyses.

Once the honeymoon was over, however, major drawbacks of peer-to-peer systems started to be realized, with extensive research efforts to mitigate them. The number one challenge on the list was its lack of robustness against *peer dynamics*, such as a flash crowd scenario in which a large number of peers join around the same time, or an alarmingly high attrition rate in which a large number of peers leave the system. By dividing the file into smaller blocks and serving them from multiple peers, BitTorrent was able to mitigate the negative impact of peer dynamics, but was still not able to provide any performance guarantees that are routine if dedicated servers are to be used.

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 [114], 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 [115]. 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, cloud computing and datacenters, the antithesis of peer-to-peer systems, 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.

As cloud computing begins its ascent to infinity and beyond like Buzz Lightyear in Toy Story, can Woody be rescued from its precipitous fall in the attention he gets? Network coding, a simple yet far-reaching idea, has been touted since 2005 to be the savior in peer-to-peer systems. We concur that peer-to-peer systems may be the most promising context in which network coding is to be applied, only because end hosts are able to afford the increased computational complexity introduced by network coding. Though benefits of random network coding in file sharing remain hazy and debatable, network coding in streaming systems may be of a more practical value. Still, the shift in paradigm to cloud computing is unstoppable, and the cancellation of IPTPS in 2011 may be sufficient evidence that the train has already left the station.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. ACM SIGCOMM*, 2001, pp. 149–160.

[2] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proc. International Conference on Distributed Systems Platforms (Middleware)*, 2001, pp. 329–350.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proc. ACM SIGCOMM*, 2001, pp. 161–172.

[4] Y.-h. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," in *Proc. ACM SIGMETRICS*, 2000, pp. 1–12.

[5] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling Conferencing Applications on the Internet Using an Overlay Muilticast Architecture," in *Proc. ACM SIGCOMM*, 2001, pp. 55–67.

[6] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Trans. Comput. Syst.*, vol. 2, pp. 277–288, 1984.

[7] A. Oram, *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*, 1st ed. O'Reilly Media, 2001.

[8] Q. Lv, S. Ratnasamy, and S. Shenker, "Can Heterogeneity Make Gnutella Scalable?" in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 94–103.

[9] M. Ripeanu and I. Foster, "Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 85–93.

[10] D. S. Bernstein, Z. Feng, B. N. Levine, and S. Zilberstein, "Adaptive Peer Selection," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 237–246.

[11] D. R. Nancy Lynch, Dahlia Malkhi, "Atomic Data Access in Content Addressable Networks," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[12] D. Liben-nowell, H. Balakrishnan, and D. Karger, "Observations on the Dynamic Evolution of Peer-to-Peer Networks," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 22–33.

[13] M. J. Freedman and R. Vingralek, "Efficient Peer-to-Peer Lookup Based on a Distributed Trie," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 66–75. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687798

[14] J. Saia, A. Fiat, S. D. Gribble, A. R. Karlin, and S. Saroiu, "Dynamically Fault-Tolerant Content Addressable Networks," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 270–279.

[15] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 53–65. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687801

[16] R. van Renesse and K. Birman, "Scalable management and data mining using Astrolabe," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[17] P. Keleher, B. Bhattacharjee, and B. Silaghi, "Are Virtualized Overlay Networks Too Much of a Good Thing," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 225–231.

[18] S. Ratnasamy, I. Stoica, and S. Shenker, "Routing Algorithms for DHTs: Some Open Questions," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 45–52.

[19] B. Wilcox-O'Hearn, "Experiences Deploying a Large-Scale Emergent Network," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 104–110.

[20] A. Iamnitchi, M. Ripeanu, and I. T. Foster, "Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 232–241. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687944

[21] S. Hand and T. Roscoe, "Mnemosyne: Peer-to-Peer Steganographic Storage," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 130–140.

[22] S. Srinivasan and E. W. Zegura, "Network Measurement as a Cooperative Enterprise," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 166–177.

[23] B. Cooper and H. Garcia-Molina, "Peer-to-Peer Resource Trading in a Reliable Distributed System," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 319–327. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687803

[24] R. Cox, A. Muthitacharoen, and R. Morris, "Serving DNS Using a Peer-to-Peer Lookup Service," in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 155–165. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687812

[25] M. R. David R Karger, "Diminished Chord : A Protocol for Heterogeneous Subgroup Formation in Peer-to-Peer Networks," in *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

[26] E. Sit, F. Dabek, and J. Robertson, "UsenetDHT: A Low Overhead Usenet Server," in *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004, pp. 206–216.

[27] C. B. Rodrigo Rodrigues, "When Multi-Hop Peer-to-Peer Lookup Matters," in *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004, pp. 112–122.

[28] A. B. Robbert van Renesse, "Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol," in *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004, pp. 173–183.

[29] U. W. M Naor, "A Simple Fault Tolerant Distributed Hash Table," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 1–6.

[30] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, "Efficient Broadcast in Structured P2P Networks," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 304–314.

[31] B. Bhattacharjee, S. Chawathe, V. Gopalakrishnan, P. Keleher, and B. Silaghi, "Efficient Peer-To-Peer Searches Using Result-Caching," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 225–236.

[32] I. Gupta, K. Birman, P. Linga, A. Demers, and R. Renesse, "Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 160–169.

[33] D. R. K. M Frans Kaashoek, "Koorde: A Simple Degree-Optimal Distributed Hash Table," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 98–107.

[34] M. Freedman and D. Mazières, "Sloppy Hashing and Self-Organizing Clusters," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 45–55.

[35] J. K. Sean Rhea, Timothy Roscoe, "Structured Peer-to-Peer Overlays Need Application-Driven Benchmarks," in *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 56–67.

[36] G. V. Ranjita Bhagwan, Stefan Savage, "Understanding Availablility," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[37] D. R. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," in *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004, pp. 36–43.

[38] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[39] M. M. John Byers, Jeffrey Considine, "Simple Load Balancing for Distributed Hash Tables," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 80–88.

[40] N. Christin and J. Chuang, "On the Cost of Participating in a Peer-to-Peer Network," in *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004, pp. 22–32.

[41] S. Suri, C. Tóth, and Y. Zhou, "Uncoordinated Load Balancing and Congestion Games in P2P Systems," in *Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004, pp. 123–130.

[42] T.-W. Ngan, D. S. Wallach, and P. Druschel, "Enforcing Fair Sharing of Peer-to-Peer Resources," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 149–159.

[43] J. Shneidman and D. C. Parkes, "Rationality and Self-Interest in Peer to Peer Networks," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003, pp. 139–148.

[44] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," in *Proc. ACM SIGCOMM*, 2004, pp. 367–378.

[45] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The BitTorrent P2P File-Sharing System: Measurements and Analysis," in *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005, pp. 205–216.

[46] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting BitTorrent For Fun (But Not Profit)," in *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[47] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, "Free-Riding in BitTorrent Networks with the Large View Exploit," in *Proc. 6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007, pp. 1–7.

[48] J. Wang, C. Yeo, V. Prabhakaran, and K. Ramch, "On the Role of Helpers in Peer-to-Peer File Download Systems: Design, Analysis and Simulation," in *Proc. 6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[49] Q. Lian, Y. Peng, M. Yang, Z. Zhang, Y. Dai, and X. Li, "Robust Incentives via Multi-level Tit-for-tat," in *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006, pp. 167–178.

[50] B. Cooper, "Quickly Routing Searches without Having to Move Content," in *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005, pp. 163–172.

[51] M. Zhong and K. Shen, "Popularity-Biased Random Walks for Peer-to-Peer Search under the Square-Root Principle," in *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[52] R. Tian, Y. Xiong, Q. Zhang, B. Li, B. Y. Zhao, and X. Li, "Hybrid Overlay Structure Based on Random Walks," in *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005, pp. 152–162.

[53] I. Abraham, A. Badola, D. Bickson, D. Malkhi, S. Maloo, and S. Ron, "Practical Locality-Awareness for Large Scale Information Sharing," in *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005, pp. 173–181.

[54] M. Zaharia and S. Keshav, "Gossip-Based Search Selection in Hybrid Peer-to-Peer Networks," in *Proc. 5th International*

Workshop on Peer-to-Peer Systems (IPTPS), 2006, pp. 139–153.

[55] T. Moscibroda, S. Schmid, and R. Wattenhofer, "On the Topologies Formed by Selfish Peers," in *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006, pp. 133–142.

[56] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," in *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005, pp. 127–140.

[57] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, "Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming," in *Proc. IEEE INFOCOM*, vol. 3, 2005, pp. 2102–2111.

[58] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, Design and Analysis of a Large-scale P2P-VoD System," in *Proc. ACM SIGCOMM*, 2008, pp. 375–388.

[59] C. Huang, J. Li, and K. W. Ross, "Peer-Assisted VoD: Making Internet Video Distribution Cheap," in *Proc. 6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007, pp. 1–6.

[60] B. Cheng, X. Liu, Z. Zhang, and H. Jin, "A Measurement Study of a Peer-to-Peer Video-on-Demand System," in *Proc. 6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[61] R. J. Dunn, S. D. Gribble, H. M. Levy, and J. Zahorjan, "The Importance of History in a Media Delivery System," in *Proc. 6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[62] K. Park, S. Pack, and T. Kwon, "Climber: an Incentive-Based Resilient Peer-to-Peer System for Live Streaming Services," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008, pp. 1–10.

[63] P. Garbacki, D. H. J. Epema, J. Pouwelse, and M. V. Steen, "Offloading Servers with Collaborative Video on Demand," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[64] M. Cha, P. Rodriguez, S. Moon, and J. Crowcroft, "On Next-Generation Telco-Managed P2P TV Architectures," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[65] Y. Boufkhad, F. Mathieu, F. Montgolfier, D. Perino, and L. Viennot, "Achievable Catalog Size in Peer-to-Peer Video-on-Demand Systems," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[66] X. Yang, M. Gjoka, P. Chhabra, A. Markopoulou, and P. Rodriguez, "Kangaroo: Video Seeking in P2P Systems," in *Proc. 8th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[67] F. Liu, L. Zhong, and B. Li, "How P2P Streaming Systems Scale Over Time Under a Flash Crowd?" in *Proc. 8th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[68] H. Wang, J. Liu, and K. Xu, "On the Locality of BitTorrent-based Video File Swarming," in *Proc. 8th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[69] O. Görlitz, S. Sizov, and S. Staab, "PINTS: Peer-to-Peer Infrastructure for Tagging Systems," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[70] B. D. McBride and C. Scoglio, "Constructing Traffic-Aware Overlay Topologies: a Machine Learning Approach," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[71] M. Matos, A. Nunes, R. Oliveira, and J. Pereira, "StAN: Exploiting Shared Interests without Disclosing Them in Gossip-based Publish/Subscribe," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[72] Y. Shavitt, E. Weinsberg, and U. Weinsberg, "Estimating Peer Similarity using Distance of Shared Files," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[73] M. J. Freedman, C. Aperjis, and R. Johari, "Prices are Rright: Managing Resources and Incentives in Peer-Assisted Content Distribution," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[74] G. Dán, "Cooperative Caching and Relaying Strategies for Peer-to-peer Content Delivery," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[75] P. Di and K. Kutzner, "Providing KBR Service for Multiple Applications," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[76] M. Haridasan and R. V. Renesse, "Renesse. Gossip-Bbased Distribution Estimation in Peer-to-Peer Networks," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[77] S. W. Ho, T. Haddow, J. Ledlie, M. Draief, and P. Pietzuch, "Deconstructing Internet Paths: An Approach for AS-Level Detour Route Discovery," in *Proc. 8th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[78] M. Chow and R. V. Renesse, "A Middleware for Gossip Protocols," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[79] J. Leitão, R. V. Renesse, and L. Rodrigues, "Balancing Gossip Exchanges in Networks with Firewalls," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[80] C. Lumezanu, D. Levin, B. Han, N. Spring, and B. Bhattacharjee, "Don't Love Thy Nearest Neighbor," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[81] P. Marciniak, N. Liogkas, A. Legout, and E. Kohler, "Small Is Not Always Beautiful," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[82] A. L. H. Chow, L. Golubchik, and V. Misra, "Improving BitTorrent: A Simple Approach," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[83] D. Choffnes, J. Duch, D. Malmgren, R. Guimerà, F. Bustamante, and L. A. N. Amaral, "Strange Bedfellows: Community Identification in BitTorrent," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[84] G. Dán and N. Carlsson, "Power-law Revisited: A Large Scale Measurement Study of P2P Content Popularity," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[85] M. Iliofotou, G. Siganos, X. Yang, and P. Rodriguez, "Comparing BitTorrent Clients in the Wild: The Case of Download Speed," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[86] M. Meulpolder, L. D'Acunto, M. Capotă, M. Wojciechowski, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "Public and Private BitTorrent Communities: A Measurement Study," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[87] J. Li, Y. Sovran, and A. Libonati, "Pass it on: Social Networks Stymie Censors," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[88] B. Wong and S. Guha, "Quasar: a probabilistic publish-subscribe system for social networks," in *Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[89] D. R. Sandler and D. S. Wallach, "Birds of a FETHR: Open, Decentralized Micropublishing," in *Proc. 8th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[90] J. Bai, D. Seah, J. Yong, and B. Leong, "Offloading AI for Peer-to-Peer Games with Dead Reckoning," in *Proc. 8th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[91] M. Varvello, C. Diot, and E. Biersack, "A Walkable Kademlia Network for Virtual Worlds," in *Proc. 8th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[92] R. A. Ferreira, "SplitQuest: Controlled and Exhaustive Search in Peer-to-Peer Networks," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[93] R. S. Peterson and B. W. Emingünsirer, "Blindfold: A System to "See No Evil" in Content Discovery," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[94] Y. Feng, B. Li, and B. Li, "Peer-to-Peer Bargaining in Container-Based Datacenters," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[95] Z. Yang, B. Y. Zhao, Y. Xing, S. Ding, F. Xiao, and Y. Dai, "AmazingStore: Available, Low-cost Online Storage Service Using Cloudlets," in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[96] M. Hosseini, D. Ahmed, S. Shirmohammadi, and N. Georganas, "A Survey of Application-Layer Multicast Protocols," *IEEE Communications Surveys Tutorials*, vol. 9, no. 3, pp. 58 –74, 2007.

[97] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr., "Overcast: Reliable Multicasting with an Overlay Network," in *Proc. 4th Symposium on Operating System Design and Implementation (OSDI)*, vol. 4, 2000, pp. 1–14.

[98] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003, pp. 298–313.

[99] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003, pp. 282–297.

[100] B. Pittel, "On Spreading a Rumor," *SIAM Journal of Applied Mathematics*, vol. 47, no. 1, pp. 213–223, 1987.

[101] S. Sanghavi, B. Hajek, and L. Massoulie, "Gossiping with Multiple Messages," in *Proc. IEEE INFOCOM*, Anchorage, Alaska, 2007.

[102] A. Bar-Noy and S. Kipnis, "Broadcasting Multiple Messages in Simultaneous Send/Receive Systems," *Discrete Applied Mathematics*, vol. 55, pp. 95–105, 1994.

[103] J. Mundinger, R. Weber, and G. Weiss, "Optimal Scheduling of Peer- to-Peer File Dissemination," *Journal of Scheduling*, pp. 105–120, 2007.

[104] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Trans. Inform. Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[105] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *Proc. Int'l Symposium on Information Theory (ISIT)*, 2003.

[106] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. IEEE INFOCOM*, March 2005.

[107] P. Chou, Y. Wu, and K. Jain, "Practical Network Coding," in *Proc. Allerton Conference on Communication, Control and Computing*, October 2003.

[108] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P Content Distribution System with Network Coding," in *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[109] ——, "Comprehensive View of a Live Network Coding P2P System," in *Proc. 6th ACM Internet Measurement Conference (IMC)*, 2006, pp. 177–188.

[110] C. Feng, B. Li, and B. Li, "Understanding the Performance Gap between Pull-based Mesh Streaming Protocols and Fundamental Limits," in *Proc. IEEE INFOCOM*, April 2009.

[111] M. Wang and B. Li, "R^2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE J. on Sel. Areas in Communications*, December 2007.

[112] C. Feng and B. Li, "On Large-Scale Peer-to-Peer Streaming Systems with Network Coding," in *Proc. ACM Multimedia*, October 2008, pp. 269–278.

[113] Z. Liu, C. Wu, B. Li, and S. Zhao, "UUSee: Large-Scale Operational On-Demand Streaming with Random Network Coding," in *Proc. IEEE INFOCOM*, 2010.

[114] I. Stoica, "It's Not the Cost, It's the Quality!" in *Proc. 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[115] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang, "Understanding the Impact of Video Quality on User Engagement," in *Proc. ACM SIGCOMM*, 2011, pp. 362–373.