Locality-Awareness in BitTorrent-like P2P Applications

Bo Liu, Yi Cui, Yansheng Lu, and Yuan Xue

***Abstract*—This paper presents the measurement study of locality-aware P2P solutions over real-world Internet AS topology. By using the accesses of nodes of PlanetLab testbed, we create a detailed AS-level map including the end-to-end path of all nodes, as well as the relationship of all involved ASes. Based on this map, we evaluate the performance of a set of locality-aware P2P solutions, including an optimal solution guaranteeing the minimum AS hop count, as well as modified BitTorrent system with locality-awareness built into its neighbor selection, peer choking/unchoking, and piece selection processes. Our findings suggest that locality-awareness can help existing P2P solution to significantly decrease load on Internet, and achieve shorter downloading time. By comparing the performance of different kinds of locality-aware and traditional BitTorrent systems, we also point out the necessity to tradeoff between the goals of optimizing AS-related performance and achieving fairness among peers such as intra-AS traffic and peer burden fairness.**

I. Introduction

Peer-to-peer (P2P) communication has been proven to be an extremely powerful paradigm to a diverse family of Internet applications. A few examples include bulk content distribution [1], voice over IP [2], and broadcasting of TV-quality program [3], [4], all of which have been proved by the commercial deployment of planet-scale systems serving tens of millions of users. Among them, BitTorrent is arguably the biggest constituent of P2P traffic, which dominates today's Internet.

In P2P systems, every peer not only downloads content from other peers, but also makes use of its upload bandwidth to serve other peers. There is great diversity of different peers both in terms of geographic distance and Internet topology, which introduces tremendous amount of traffic crossing the boundary of Internet Service Providers (ISPs). Such traffic often causes great financial loss to ISPs with active P2P users, which motivates them to control P2P traffic by "throttling", or bandwidth limiting. As countermeasures, many P2P applications use traffic obfuscation technique to make itself indistinguishable from other applications, an attempt to stop ISPs from regulating P2P traffic. Such an interaction quickly becomes an escalating game of mouse and cat.

This work was supported by NSF award 0643488, Vanderbilt Discovery grant, and a gift from Microsoft Research. Views and conclusions of this paper are those of authors, which should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

Bo Liu and Yansheng Lu are with the school of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei, 430074 China e-mail: {*newpoo,* lys}@smail.hust.edu.cn.

Yi Cui and Yuan Xue are with the Department of Computer Science, Vanderbilt University, Nashville, TN, 37240 USA e-mail: {*yi.cui,* yuan.xue}@vanderbilt.edu.

To alleviate the tension between ISPs and P2P users, actions can be taken on both sides. On one hand, ISP can install cache nodes to increase the data availability and redirect P2P applications to such opportunities within the same ISP. On the other hand, P2P applications can employ a variety of techniques such as ISP-friendly neighbor selection and locality-aware piece scheduling algorithm to reduce the inter-ISP traffic. So far, most works dedicated to this subject have been focused on the feasibility of ISP-friendly P2P applications[5], [6], quantitative study on saving of inter-ISP traffic through trace analysis[7], and simulation study on the tradeoff between downloading performance and enhancing intra-ISP traffic[8]. Given these works, we argue that what is still missing is a comprehensive study on various ways to embed locality-awareness into P2P applications and their impacts on the ISPs. In particular, the following two facts are taken into account in our study.

First, the relationship among ISPs is not as simple as either inter- or intra-ISP. ISPs themselves interconnect into a complex network of autonomous systems (AS)[1]. Two ASes, if connected to each other, must have one of the following four kinds of relationships: Provider-Customer, Customer-Provider, Peer-to-Peer[2], and Sibling. Each customer AS should pay its provider AS for both inbound and outbound traffic. Traffic across peer-to-peer ASes or sibling ASes are usually free. The users of a single P2P application can reside in many ASes which, together with third-party ASes interconnecting them, easily form a network consisting of all above relationships.

Second, a P2P application is usually composed of sophisticated semantics. Take BitTorrent as an example, it operates at several levels. At the macroscopic level, a peer, if intended to download a file, first retrieves from the tracker a list of other peers interested in the same file (neighbor selection). At the intermediate level, during the downloading, among the list of peers, each peer dynamically determine the subset of other peers to share data with (choking and unchoking). At the microscopic level, a file is divided into many pieces. Among multiple connections with other peers, each peer chooses which piece to download from which peer (piece picking policy). Readers can find a detailed discussion on the same topic at Sec. III-C. This architectural design is inherited by many P2P applications including P2P streaming systems.

[1]In the remainder of this paper, we use the terms AS and ISP interchangeably.

[2]We note that this notion should not be confused with the same term used to described applications such as BitTorrent. To avoid confusion, we use *peer-to-peer* to refer to relationship among ASes, and *P2P* to refer to BitTorrent and applications alike.

Although existing work has studied ISP-friendly neighbor selection[8] (macroscopic level), awareness to ISP-friendliness can be actually built in at all levels.

These two facts call for a systematic field study on various locality-aware P2P solutions and their impacts on ISPs. In light of the first fact, we need a detailed and up-to-date AS-level map revealing the connection among ASes and their financial relationships. While mature techniques have been practiced to infer AS relationship, we still need to know the exact end-to-end path two peers connect with each other, which often traversing multiple ASes. Such knowledge can be only obtained when one is able to access a large number of peer machines. In light of the second fact, it is extremely challenging to rely on analytical study to model subtle locality-aware mechanisms deployed at all operating levels of a P2P application such as BitTorrent. Having a reasonably-sized testbed to run the P2P application on all its nodes would be a better appproach to fully capture the behavior of peers and the impact on the ISPs they belong to.

As such, we conduct our study on the PlanetLab testbed[9], we obtain the access to PlanetLab nodes around the globe, and use the information on their all-pair end-to-end path to construct a detailed AS-level map. Based on this map, we conduct both simulation and real-system studies. First, we devise the optimal locality-aware strategy, which minimizes AS hop count of the entire P2P distribution structure. Since the optimal structure is a static tree, obtaining its performance via simulation on the AS-level map would achieve the same effect as deploying it on PlanetLab. Second, we modify the BitTorrent system at all operating levels and test its performance on PlanetLab. In particular, we build locality-awareness into neighbor selection (macroscopic level), choking/unchoking mechanism (intermediate level) and piece picking policy (microscopic level).

Our findings suggest that locality-awareness can help existing P2P solution to significantly decrease load on Internet and achieve shorter downloading time. We point out the necessity to tradeoff between the goals of achieving fairness among peers and optimizing AS-related performance such as intra-AS traffic and peer burden fairness. We also find that continuous seeding can not improve the downloading time of standard BitTorrent, but can significantly improve it for locality policies. It can also reduce the number of connected peers for choker and piece picker locality policies.

The rest of this paper is organized as follows. First, we discuss the related work in Sec. II to prepare background information to our study. Sec. III presents the evaluation methodology. We present findings of our experiment in Sec. IV and conclude the paper in Sec. V.

II. Related Work

We summarize previous works in four key areas related to our research: (1) BitTorrent system and studies which analyze and measure it, (2) P2P streaming solutions and efforts to adapt BitTorrent to streaming applications, (3) studies on locality-aware P2P solutions, and (4) studies on inferring Internet AS relationships.

Many analytical studies[10], [11], [12] have proved that BitTorrent is nearly optimal in terms of user experienced downloading time. In particular, [12] shows that the optimistic unchoking policy and rarest-first policy are in fact unnecessary for BitTorrent to achieve asymptotic optimality in terms of user population, where random peer and piece selection would suffice. The near-optimal performance of BitTorrent is also confirmed in many simulation and measurement studies, e.g., [13], [14]. However, all these studies, in fidelity with the original BitTorrent design, do not consider the issues of locality and ISP friendliness.

An extended set of works have proposed P2P solutions to the large-scale distribution problem in the context of multimedia streaming. Many works are developed under the term overlay multicast, which we consider equal to P2P streaming. Narada[15] is the pioneering work promoting the utilization of peer resources to replace the infrastructure support, i.e., IP multicast. Other notable works include Bullet[16] and SplitStream[19], etc. The P2P solution is shown to be able to handle a variety of application scenarios, including live streaming such as conferencing, and on-demand streaming such as video-on-demand. In particular, peer-side caching is widely used to address the asynchronous request problem in on-demand streaming. oStream[20] utilizes application-layer multicast and peer buffering to support video-on-demand. A cache-and-relay architecture is proposed in [21]. Advanced coding schemes are also applied to increase the system resilience or throughput, such as multiple description coding[22], erasure coding[23], rateless coding[24], and network coding[25], etc. All these solutions build on certain distribution structure, e.g., single tree, multiple trees, or mesh. They are dynamic to withstand peer joining or leaving, but have a clearly-defined parent-child relationship between any pair of connected peers. This is in contrast with BitTorrent design, where peers exchange data within a much less structured swarm.

Interesting enough, BitTorrent has been highly influential to the design and development of many modern commercial P2P streaming systems[3], [4], [26], which adopt a receiver-driven piece selection approach. Given BitTorrent's statue as the de facto P2P downloading protocol and a high-quality open-source software, there have been several proposals to directly modify it to support the "viewing-while-downloading" feature, such as BASS[27], BiToS[28], and Toast[29]. The basic idea of these works is to restrain the piece picking action within a moving window along with the playback, which is also adopted by our research. These previous works primarily focus on server load reduction by the aid of P2P downloading, and proved that significant saving is achievable via simulation and system deployment. In our work, we also explore other performance-enhancement dimensions such as locality-aware P2P downloading. BitTorrent inc. also promotes DNA[30], a content distribution solution which claims to support "viewing-while-downloading" as well. However, its technical details remain unknown at the moment of writing.

There have been several proposals on locality-aware P2P solutions. Bindal et al.[8] propose biased neighbor selection mechanism to reduce inter-ISP traffic, which requires no dedicated central servers. Karagiannis et al.[5] show locality-aware P2P solutions can significantly alleviate the induced cost at the ISP. The work by Ren et al.[6] confirm the benefits gained by peer-relay in VoIP, and then propose an AS-aware peer-relay protocol for P2P VoIP system. Huang et al.[7] confirm the benefits of peer-assisted VoD, and show that locality-aware P2P solution can reduce inter-ISP traffic. [17], [18] propose some practical methods to provide Internet locality topology information to peers. However, there is little work on the evaluation of impact of locality-aware P2P solutions based on real world Internet AS topology.

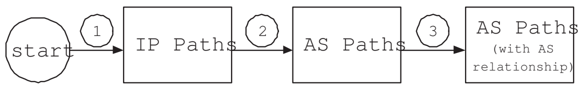
Finally, there are many works on inferring Internet AS topology, which we simply leverage. The work by Gao [31] is the first comprehensive study targeted on this topic. By analyzing BGP table entries, it finds valley-free property of Internet AS path, and further identifies relationships between neighbor AS pairs. Spring et al.[32] present a technique for mapping the router-level topology of an ISP or a focused portions of the Internet, which use only end-to-end traceroute measurements. Dimitropoulos et al.[33] introduce some heuristics to address the problems of inferring peer-to-peer and sibling relationships. They also validate the inferred AS relationships. Though there are many works on inferring Internet AS relationships, the data set of inferred AS relationships is still incomplete as we will see in Sec. III.

III. Evaluation Methodology

In this section, we present our methodology to evaluate locality-aware P2P solutions. We first introduce how we obtain the AS-level map, the topological foundation upon which our study is performed. We then describe key issues determined when planning the evaluation. Finally, we introduce how we build locality-awareness into the BitTorrent system.

A. *Obtaining AS-level Map*

Our experiment is conducted over real-world Internet topology. We construct an AS-level map on the PlanetLab testbed[9]. Fig. 1 illustrates this process step by step.



```
(1) planet-lab
(2) http://www.cymru.com/BGP/asnlookup.html
(3) http://as-rank.caida.org/ and Valley Free
```

Fig. 1. Step-by-step Process of Obtaining AS-Level Map

First, we run traceroute between each pair of PlanetLab nodes to obtain the IP-level end-to-end path between them. We then assemble these paths into an IP-level map. Some nodes are eliminated from the map since the traceroute program fails to return the IP-level path over them.

Second, we convert each IP path obtained in step one into an AS path. For each IP address shown up in the IP-level map, we find its AS number through public AS-lookup service such as the one run by the CYMRU team[34]. Since an end-to-end path linearly traverses multiple ASes, we aggregate consecutive IP addresses with the same AS number into a single AS node. In this way, we condense an IP-level path into an AS-level path, and further transfer the IP-level map into an AS-level map. A small number of nodes are further eliminated due to the failure of AS lookup.

Third, we mark the AS-level map with AS relationship data provided by CAIDA[35]. Each adjacent AS pair must have one of the following four kinds of relationships: Provider-Customer, Customer-Provider, Peer-to-Peer, and Sibling. Each customer AS should pay its provider AS for both inbound and outbound traffic. Traffic across peer-to-peer ASes or sibling ASes are usually free. 70% of the AS pairs in our AS-level map are identified via the CAIDA dataset.

To identify the relationship of the remaining AS pairs, we apply the "valley-free" property proposed in [31]. In brief, if we represent our AS-level map in a hierarchical structure where every AS is positioned lower than its provider, higher than its customer, and at the same level with its siblings and peers, then any AS-level path should not form a valley, i.e., the path should start with zero or more customer-provider pairs, then zero or more peer-to-peer pairs, finally zero or more provider-customer pairs, and sibling pairs can exist in any place of an AS path. Assuming all AS-level paths follow this property, we improve the percentage of identified AS pairs to 90%. Finally, we eliminate the unidentified AS pairs from the map.

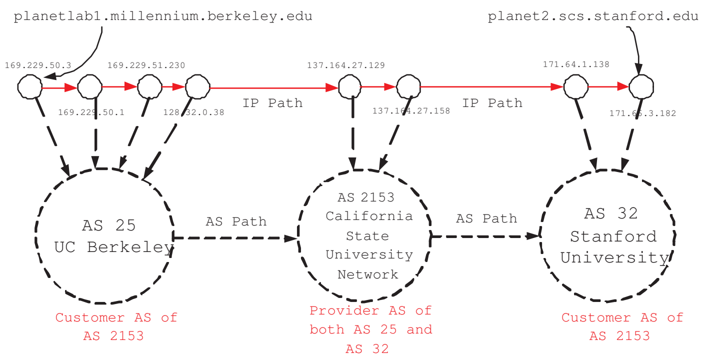


Fig. 2. A Processing Example

In Fig. 2, we illustrate the above steps by a sample PlanetLab pair "planetlab1.millennium.berkeley.edu" and "planet2.scs.stanford.edu". In this example, the IP path consists of eight IPs, which locate in three different consecutive ASes. These ASes are UC Berkeley(AS 25), California State University Network(AS 2153) and Stanford University(AS 32). The AS 2153 is the provider of both AS 25 and AS 32. Therefore, for traffic from AS 25 to AS 32 or vice versa, both of them will be charged by AS 2153.

B. Evaluation Setup

We evaluate various locality-aware P2P solutions on the AS-level map we have obtained, either through simulation or system deployment on PlanetLab. In what follows, we discuss a few key issues we have determined when planning the evaluation. We start by discussing the application scenarios our evaluation covers, followed by the choice of primary and secondary performance metrics, and finally the optimal strategy we have derived based on our choice of the primary performance metric.

1) Downloading vs. On-demand Streaming: The basic semantic in each experiment of our evaluation is to have a group of peers downloading a video file from a seed (a peer possessing the whole copy of the file). Under this basic setting, we mainly evaluate two common scenarios, *downloading* and *on-demand streaming*.

In the downloading scenario, we assume that all peers show the interest to the file to be downloaded at the same time. Besides the purpose to mimic flash crowd, we set all peers to join the P2P network simultaneously to avoid the temporal dependency problem where a new peer has to download from an earlier-joined peer. Instead, under the current setting, all peers initiate downloading under the same condition, where only a single copy is available at the seed, and race to finish. From the end users' perspective, the most important metric is downloading time, i.e., the time it takes from the start of downloading until the file is fully downloaded.

In the on-demand streaming scenario, we simulate a video-on-demand application, where all peers are interested to view one video file and these peers start viewing the video at different times. In addition, this scenario supports the "viewing-while-downloading" feature, where the video file must be downloaded in an approximately sequential fashion. In this scenario, the temporal dependency plays a much more important role, where a peer is more likely to download from an earlier-joined peer, unless a later-joined peer downloads at a much faster speed exceeding its predecessors. From the end users' perspective, the viewing experience becomes the most important factor, i.e, the video viewing should be continuous. To achieve so, the downloading should be no slower than the video playback speed, which will cause viewing interruption otherwise. In Sec. IV, we will introduce "interruption time", a metric introduced to describe this phenomenon unique to streaming.

2) Performance Metrics: Besides user-oriented metrics such as downloading time or interruptions, we must choose metrics matching the theme of this work: locality-awareness. With this regard, our primary choice is *AS hop count*, the number of ASes a data piece traverses from its sender to receiver. This is a more generalized version of the intra-ISP traffic used in previous works. Obviously, if a piece only travel within a single ISP, its AS hop count is 0. Otherwise, its value will be a positive integer representing the number of ASes it has traveled. From this basic definition, we can also derive *weighted AS hop count*, which is the average AS hop count that all pieces downloaded by a peer have traversed.

We also consider *redundancy* proposed in [8], which measures the number of times a piece has to enter an ISP until all peers in the ISP finish their downloading or streaming. The lowest value is 1, which means that the piece only needs to enter the ISP once, and all peers within the ISP are able to distribute it without asking for additional help outside. On the contrary, the highest value is N, the number of peers within the ISP. We also propose *normalized redundancy*, which is the redundancy normalized by the number of peers. We use this metric to measure the relative effectiveness of a P2P solution at restraining traffic within a single ISP. Finally, we note that these redundancy metrics only apply to ISPs with peers inside, not ISPs which only carry through traffic.

To evaluate the economic impact different P2P solutions have on ISPs, we also introduce *gain/cost*, which is the financial gain or loss of an ISP. An ISP gains by carrying incoming/outgoing traffic for its customer ISPs, and likewise, the customer ISP loses by asking its provider ISP to relay its incoming/outgoing traffic. The traffic between a pair of peering or sibling ISPs is not counted. Since financial charges agreed by ISPs are unknown, we assume all provider ISPs charge by the same rate and use number of bytes to represent the gain or cost.

3) Minimum-AS-hop Strategy: With AS hop count as the primary performance metric, we are able to derive the optimal P2P strategy which minimizes the total number of AS hops. We are also able to find optimal strategy for both downloading and on-demand streaming scenarios.

For downloading scenario, the optimal strategy first constructs a complete graph, where each node represents a peer, and the edge weight represents the AS hop count between the pair of peers at both ends. Then, it finds the minimum spanning tree on this graph, which is the P2P distribution structure able to minimize the AS hop count. We also note that this structure minimizes the redundancy value for each ISP, since all peers within the same ISP, except one, choose the edge whose weight is 0. Therefore, the algorithm could be accelerated by aggregating all peers within a single ISP into a cluster node, and find the minimum spanning tree among these cluster nodes.

For on-demand streaming scenario, the minimum spanning tree algorithm still applies with minor modification, where each peer must choose, only from earlier-joined peers, the one with the minimum AS hop count. In other words, the complete graph described in previous paragraph becomes directed, where at each edge the earlier-joined peer directs to the later-joined peer. We note that because this graph is acyclic, the above simple procedure is optimal since no loop-removing action is needed as in the minimum spanning tree algorithm for general directed graphs. Also under this strategy, the redundancy value is minimized for each ISP.

Obviously, the optimal distribution structure for both scenarios is a single tree, where a peer downloads the entire content from its only parent who has the minimum AS hop count. This structure suffers from all drawbacks of a tree solution, such as lowest level of resilience. Also, no degree constraint is enforced, which means a peer might be required to upload to unlimited number of children. Nevertheless, the minimum-AS-hop strategy constitutes the theoretical baseline, against which other realistic P2P solutions can be measured in terms of AS hop count. Given the static nature of these solutions, we can easily obtain their performance through simulation on AS-level map, instead of implementing and deploying it on PlanetLab.

C. Locality-Aware BitTorrent

As mentioned in Sec. I, a BitTorrent peer operates at three levels, *neighbor selection* at the macroscopic level, *peer choking/unchoking* at the intermediate, and *piece selection* at the microscopic level. In what follows, we first review design of the original BitTorrent on these aspects, then propose our modifications to bring locality-awareness into each of them. Finally, we discuss how we adapt BitTorrent to make it serve video streaming applications.

1) Overview: During the neighbor selection process, the peer learns from the tracker the knowledge of other peers in the same swarm, which is the group of peers interested in the same file. The tracker returns a peer list to the the inquiring peer, which contains the IP addresses and port numbers of at most a constant number (default value is 50) of peers. If the population of the swarm exceeds this number, the selection is entirely random. Upon receiving the peer list, a peer connects with the majority of them at the maximum number 35. It then sends to its neighbors the *bitfield* messages, which advertise the availability information of the pieces it already owns. Based on the *have* message collected from neighbors, each peer maintains an array of *interest* values, where each entry is the number of neighbors owning the corresponding piece. If a neighbor has pieces it needs, it informs the neighbor with an *interested* message.

The peer choking/unchoking is the decision process made by a peer about which of its “interested” neighbor it should send data to. It first sends *choke* message to most of its neighbors, which means it refuses to send the data. It then sends *unchoke* message to a small number (default value 4) of neighbors which have sent data to itself at the highest rate, a criteria best known as tit-for-tat. Once a peer becomes seed (the one that has the complete file), it unchokes the 4 neighbors with the highest downloading rates from itself, in order to speed up downloading of the entire swarm. Finally, it performs *optimistic unchoking* by sending *unchoke* message to a random peer, which is crucially important to bootstrap brand new peers with nothing to share yet.

The piece picking policy is executed by each peer that is unchoked. BitTorrent employs the rarest-first policy, in which the piece with the minimum interest value is chosen. If multiple pieces have the same minimum interest value, such as 1, then the tie is broken by randomly choosing one. An example illustrating this policy is given in Fig. 3 (a). The greatest benefit of this policy is that it helps promote the piece diversity of the entire swarm by help distributing the rare pieces. A diverse swarm can ensure concurrent downloading over multiple connections, thus increasing the aggregated downloading speed.

All above processes are repeated at different time granularities. The neighbor selection is renewed if the current active neighbor count drops to below a certain threshold (default value 20) due to peer leaving. The choking/unchoking process is repeated every 10 seconds and optimistic unchoking at every 30 seconds. The piece picking is executed every time an unchoked connection is available again to download a new piece.

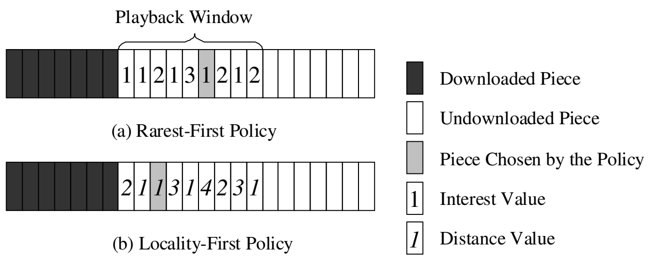


Fig. 3. BitTorrent Piece Picking Policies

2) Tracker Locality: To promote locality awareness, we replace the random peer selection with the selection that minimizes AS hop count metric. Upon a user request, the tracker sorts by the ascending order all other peers in the swarm by their distances to the requesting peer in terms of AS hop count. The tracker sends the prefix of this sorted list (e.g., first 50 peers) to the requesting peer. We note that this solution bears great similarity with the one proposed in [8], where 35 peers within the same ISP (AS hop count 0) are returned together with 15 other random peers.

We note that the tracker not only returns the peer list in the aforementioned fashion, but also attaches the distance values in an add-on field to the list, due to the following reason. The distance between two peers, if measured by AS hop count, does not change unless the peer change its ISP or the AS-level map is updated. Therefore, an existing peer can quickly accumulate such knowledge as it contacts more peers. However, a newly joined peer needs to learn its distance to other peers as early as possible in order for other locality-aware policies to function, which we will present immediately. While a peer can learn on its own the distance to other peers by probing them, the tracker is an ideal identity to collect and disseminate such information. By the same token, the AS-level map is best maintained by the tracker. The distance values can be learned through any commercial package able to map an IP address to the ISP it belongs to[36].

3) Choker Locality: We redefine the peer choking/unchoking policy by making a peer unchoke the 4 neighbors that are closest to itself in terms of AS hop count. Although the distance between two peers seldomly change, this policy will not result in the same selection of peers again and again. The primary reason is that the choking/unchoking decision is only made among interested neighbors, i.e., the peers whose piece collections are different from the local peer. Since piece collection of each peer will asymptotically grow in time, the set of interested neighbors will constantly change too. As such, this policy will enable a peer to exchange data with its close-by neighbors in a rotating fashion. The radius of this neighbor set is primarily determined by its population. The more peers are concentrated in a compact neighborhood, say a single ISP, the smaller radius the neighbor set needs to be.

The same unchoking policy applies to the seed. In this phase, the selection of unchoked peers will be much stabilized. A seed will keep unchoking 4 of its closest neighbors who still have not finished downloading. Only when a neighbor finishes downloading all pieces will the seed chooses another closest neighbor to unchoke. This is in sharp contrast with the original BitTorrent design, which gives priority to uploading speed. Giving priority to distance, our policy enables a seed to send pieces to its closest neighbors first. If all peers in the swarm have homogeneous uploading speeds, then peers closest to the seed are highly likely to be the first to finish downloading and become a seed too. This will result in the seeding of peers to be propagated by a growing radius centered around the original seed.

Finally, we keep the original BitTorrent optimistic unchoking policy intact, for the same purpose to help bootstrap brand new peers.

4) Piece Picker Locality: We propose a locality-first policy to encourage a peer to download pieces closest to itself. As shown in Fig. 3 (b), we introduce a *distance* value to each piece, which is the mean value of the distances of all peers possessing this piece. For example, if a piece is owned by three peers (i.e., its *interest* value is 3), and the AS hop counts from these peers to the downloading peer are 1, 2, 3, respectively, then the *distance* value associated with this piece should be 2. The locality-first policy chooses the piece with the smallest *distance* value. While the rarest-first policy promotes piece diversity within the swarm regardless the distance it travels, the locality-first policy encourages to distribute a piece by gradually enlarging its radius centered around the seed.

We note that the distance value could be easily calculated by a downloading peer, as long as it remembers its distances to its neighbors, which are passed by the tracker during the neighbor selection process. Upon receiving the *have* message from its neighbors regarding a particular piece, it will increment the *interest* value of this piece, meanwhile recalculating the average distance value of this piece by taking into account the *distance* value of the peer which just announced the have message.

5) Adaptation to Streaming Applications: To adapt BitTorrent to accommodate the streaming scenario, we must enforce it to support the "viewing-while-downloading" feature. The primary reason for the incompatibility between BitTorrent and the streaming scenario is its piece picking policy. Both the rarest-first and the locality-first policies ignore the position of a piece in the video playback. In the worst case, the first piece might not be downloaded until all other pieces are, which makes the video unable to play until the whole file is fetched.

Our solution is in accordance with existing proposals[27], [28], [29], which restrain the piece picking action within a window marching with the video playback. This window-based solution applies for any type of piece picking policies. In other words, we can deem the original rarest-first and locality-first policies as one extreme case of the window-based solution, where the window size equals to the entire file.

In order to keep up with the playback, the window must advance itself. In our solution, the window is automatically pushed forward whenever its leftmost piece is downloaded. Therefore, the window might advance ahead of the playback if the BitTorrent downloading speed is faster than the playback rate, or behind the playback otherwise. We note that this design, as well as all pure P2P-based solutions, are too primitive to guarantee smooth playback. To address this issue, [29] proposes a hybrid solution integrating P2P and client-server downloading. Here, a stream-watcher process monitors the downloading progress. If it falls behind the playback, it pushes the window by downloading the leftmost pieces from a video-on-demand server, until it catches up with the playback again. In addition, the stream-watcher also needs external information, such as the streaming rate of the video file, to push the window at a proper speed.

IV. Findings

We start by introducing the experiment setup, then present the performance results related to user experience, followed by locality-related results. Finally, we examine the results related with peer workload.

A. Experiment Setup

We make our changes to BitTorrent on its source code version 3.9.1 and deploy it over PlanetLab nodes. We use two PC machines at Vanderbilt University to be the original seed and BitTorrent tracker. Both machines run RedHat Linux, one with a Intel Xeon(TM) 2.80GHz CPU (seed) and the other with a Intel Pentium 4 2.80GHz CPU (tracker server). We experiment with locality-awareness features with each of them turned on individually, namely tracker locality, choker locality, and picker locality. In all runs of our experiment, we configure each peer, upon finishing downloading, to leave after 10 minutes, or stay as seed.

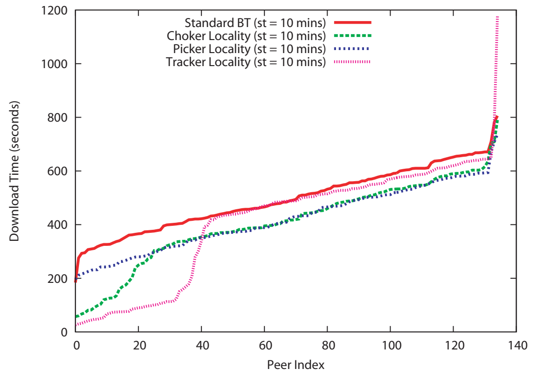
The test file is a flash video file downloaded from a video website, which lasts 28 minutes 28 seconds and is sized 61889761 bytes. For the purpose of simplicity, we determine its streaming rate to be the ratio of its size over playback length. On the day of September 5, 2007, this file has been requested for a total of 53165 times from the time point of 18:18:33 to 22:31:59. Such statistics is obtained by parsing the webpage hosting this video. Since we are not able to study its request pattern in a finer scale, we set the average request rate of our experiment to be the same as the average rate this file experienced during this period, which is around 1.5 request per second.

In the downloading scenario, we schedule all PlanetLab nodes to request this file at the same time. In the on-demand streaming scenario, we schedule each PlanetLab node to initiate a request based on this speed. All runs in the streaming scenario follow the same request sequence.

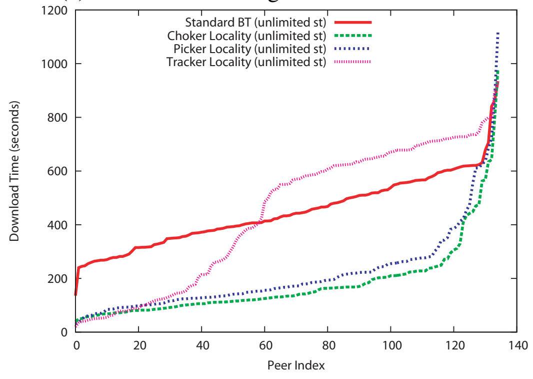
Due to various reasons such as machine failure or testbed administration, only part of PlanetLab nodes have successfully participated all runs of our experiment. Therefore, we only analyze and exhibit the results obtained from these nodes. Correspondingly, we measure the performance of the minimum-AS-hop strategy by only running it on these nodes in our simulation.

B. *User-Perceived Performance Results*

1) Downloading Time in Downloading Scenario: In Fig. 4, we show the downloading time in downloading scenario. Fig. 4 (a) shows the case with seeding time of 10 mins. In this case, the downloading time are very uneven with neighbor selection policy. Some peers need less than 200 seconds to finish downloading, but some others need nearly 1200 seconds. This may come from two facts: One is that tracker locality policy is very likely to partition the peers into some localized sets; another is that some peers might have limited network bandwidth. Choker locality and picker locality policies have more even downloading time. All of these policies download faster than standard BitTorrent. Fig. 4 (b) shows the case of unlimited seeding time. In this case, choker policy downloads faster than other policies. Again, the tracker locality policy makes a very uneven downloading time among peers. In both 10 mins seeding and unlimited seeding time cases, the choker locality policy gets the shortest and most even downloading time. From this two figures, we find a fact that continuing seeding after 10 mins seeding can not improve the downloading time of standard BitTorrent, but can significantly improve it for choker and picker locality policies.



(a) Maximum Seeding Time = 10 minutes

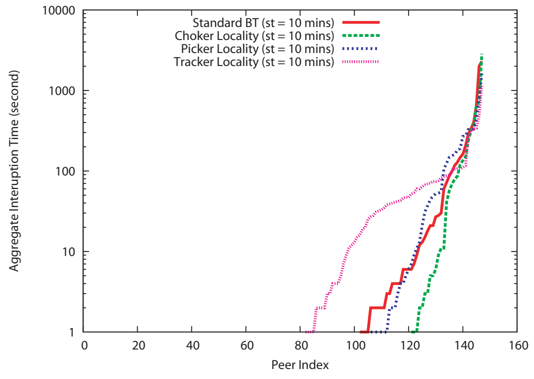


(b) Unlimited Seeding Time

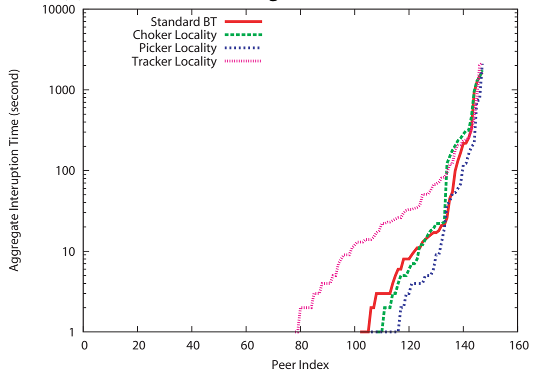
Fig. 4. Downloading Time (Downloading Scenario)

2) Interruptions in Streaming Scenario: In Fig. 5, we show the interruption time in streaming scenario. For each peer, during its streaming, we monitor the pieces received along the playback time. During the playback, if any piece is missing, the peer enters the "interruption" stage where the playback is starved. It will exit this stage when the missing pieces are received. The aggregated interruption time is the summation of time lengths of all interruptions experienced by the peer. This monitoring procedure starts from the point where the first piece is received.

This two figures show that in both 10 mins seeding and unlimited seeding time cases, the tracker locality policy makes the most number of peers suffering interruptions. The other three policies result in similar interruption experience. All these policies make more than 80% peers with no interruptions, and 90% peers with less than 100 seconds interruptions during the whole streaming process. From this two figures, we find that continuing seeding after 10 mins seeding can not further improve interruption time. The overall amounts of data uploaded by server are 1969MB, 2776MB, 1962MB and 3107MB for standard BitTorrent, choker locality, picker locality and tracker locality policies respectively. From these numbers, we can find that picker locality policy put similar load on server as standard BitTorrent, but choker locality and tracker locality policies put about 40% to 50% more burden on server than standard BitTorrent.



(a) Maximum Seeding Time = 10 minutes

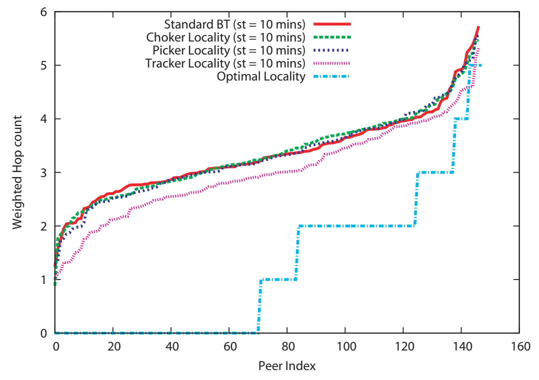


(b) Unlimited Seeding Time

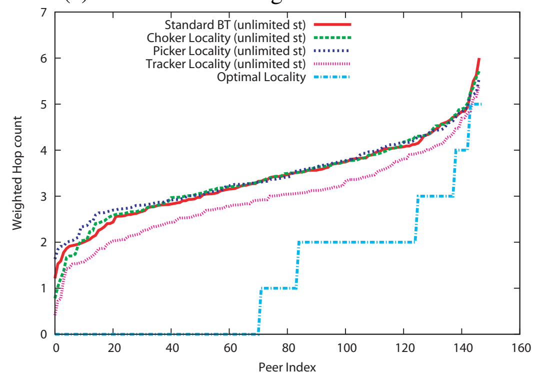
Fig. 5. Interruption (Streaming Scenario)

C. *Locality-Related Performance Results*

1) AS Hop Count: In Fig. 6, we show the weighted average hop count across all downloading paths of a peer in downloading scenario. Fig. 6 shows that tracker locality policy makes the shortest AS hop count, except the theoretical optimal bound by the minimum-AS-hop strategy. The other three policies achieve similar AS hop count. All these policies in unlimited seeding time case get a little lower hop count than in 10 mins seeding time case. This may come from the fact that peers have lower probability to download from close by peers in 10 mins seeding case than in unlimited seeding case.



(a) Maximum Seeding Time = 10 minutes



(b) Unlimited Seeding Time

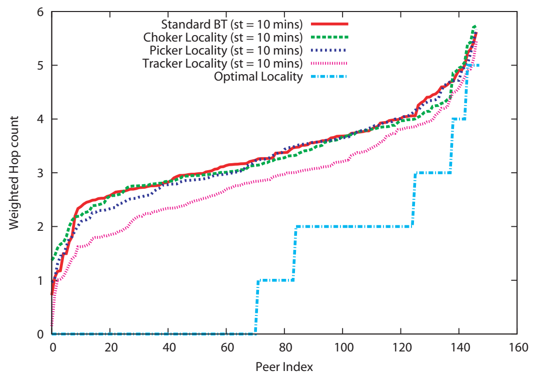
Fig. 6. AS Hop Count (Downloading Scenario)

In Fig. 7, we show the weighted average hop count across all downloading paths of a peer in streaming scenario. Fig. 7 shows that tracker locality policy makes the shortest AS hop count, except the theoretical optimal bound. Among the other three policies, the choker locality policy makes smaller AS hop count. Again, all these policies in 10 mins seeding time case get about 1 hop larger than in unlimited seeding time.

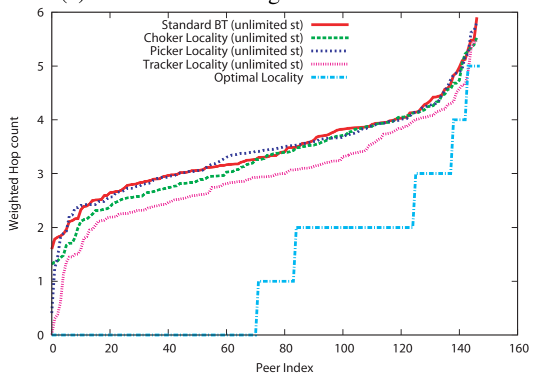
2) *Redundancy:* In Fig. 8 and 9, we show the normalized redundancy achieved in less than 80 ISPs, which host the PlanetLab peers in our experiment. Majority of them achieve the minimum value across all solutions, due to the fact each ISP only hosts one peer. For ISPs hosting multiple peers, all solutions are able to perform within 2 to 3 times of the optimal value achieved by the minimum-AS-hop strategy.

D. *Financial Impact on ISPs*

The minimum-AS-hop strategy only involves less than 150 ISPs, while other solutions involve between 200 and 250 ISPs. Among these ISPs, most of them just pass through traffics. So the overall financial gain/cost of these ISPs is zero. We also notice that the number of ISPs paying money is larger than the number of ISPs gaining money. The distribution of financial gain/cost is very uneven. Some ISPs cost a lot, such as AS 680(DFN-IP service G-WiN) which costs more than 1.7 GB data. And some ISPs gain a lot, such as AS 20965(The

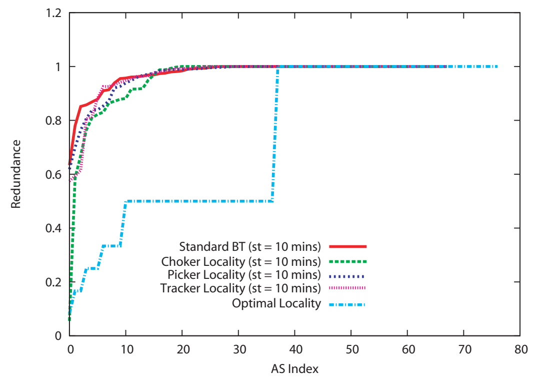


(a) Maximum Seeding Time = 10 minutes

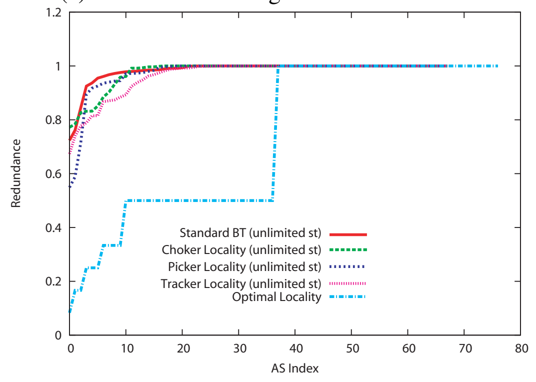


(b) Unlimited Seeding Time

Fig. 7. AS Hop Count (Streaming Scenario)

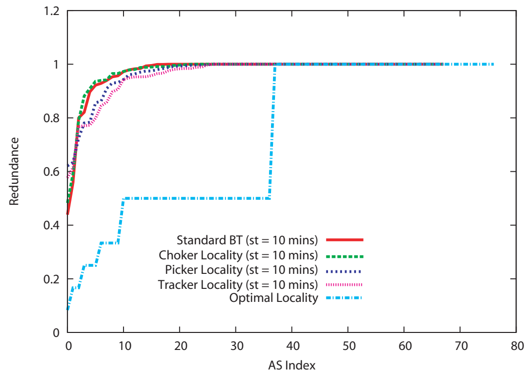


(a) Maximum Seeding Time = 10 minutes

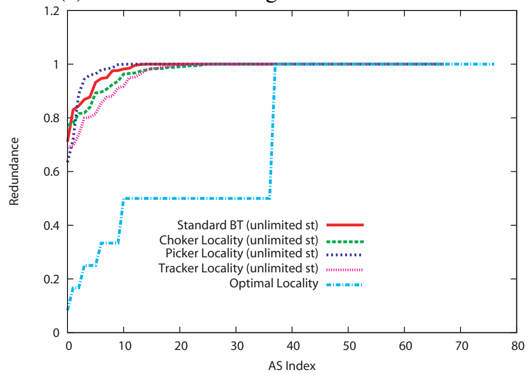


(b) Unlimited Seeding Time

Fig. 8. Normalized Redundancy (Downloading Scenario)

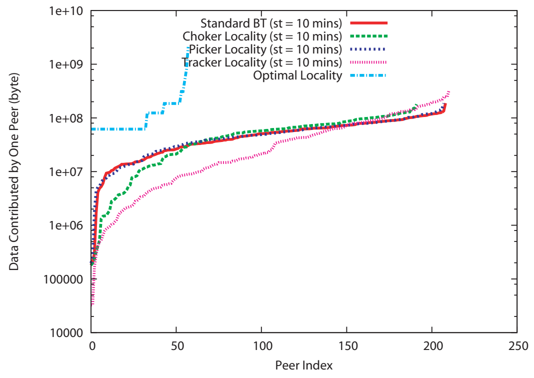


(a) Maximum Seeding Time = 10 minutes

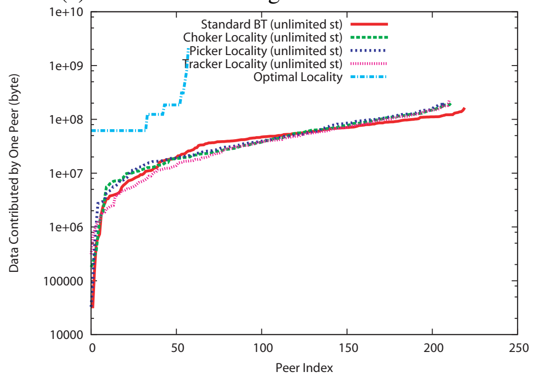


(b) Unlimited Seeding Time

Fig. 9. Normalized Redundancy (Streaming Scenario)

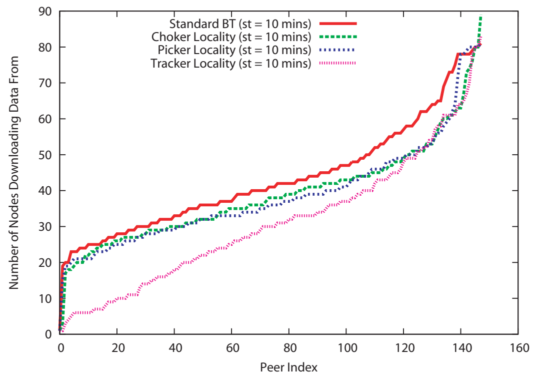


(a) Maximum Seeding Time = 10 minutes

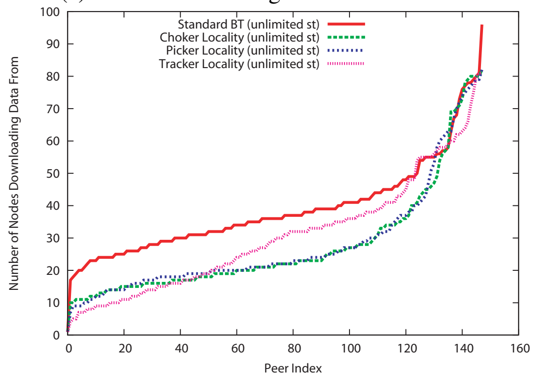


(b) Unlimited Seeding Time

Fig. 10. Traffic Uploaded per Peer (Downloading Scenario)



(a) Maximum Seeding Time = 10 minutes



(b) Unlimited Seeding Time

Fig. 11. Number of Downloading Neighbors per Peer (Downloading Scenario)

GEANT IP Service) which gains more than 1.3 GB data and AS 11537(ABILENE) which gains more than 1.8 GB data.

E. Peer Contributions

1) Traffic Uploaded per Peer: We show the amount of traffic uploaded by each peer in Fig. 10. The minimum-AS-hop strategy makes very small number(less than 70) of peers to contribute. Among all the other policies, the tracker locality policy makes the most uneven traffic distribution. Standard BitTorrent makes the most even traffic distribution. This may also come from the fact that tracker locality separates peers into localized sets. In summary, locality policies localize the traffic by paying the price of uneven traffic distribution.

2) Number of Downloading Neighbors per Peer: Obviously, every peer always downloads from a single neighbor in optimal policy. We show the number of peers each peer downloading data from in Fig. 11. Some peers download all data from one peer, while some other peers download from up to 80 peers. Most of peers download data from about 20 to 30 peers. The standard BitTorrent downloads from more peers than other policies. We can see that the more neighbors downloading from the more even the traffic distribution.

F. Findings

In downloading scenario, choker and picker locality policies can significantly reduce downloading time, while tracker locality policy achieves similar downloading time as standard BitTorrent. Tracker locality policy achieves the lowest AS hop count, but suffers most unbalanced peer load in terms of number of downloading neighbors and traffic uploaded per peer. So in downloading scenario, if shorter downloading time is of high priority, we should use choker or picker locality policies. If less inter-AS traffic is of high priority, we should use tracker locality policy.

In streaming scenario, Standard BitTorrent achieves similar disruption as choker and picker locality policies and less disruption than tracker locality policy. But it comes with much larger startup delay. The same as in downloading scenario, tracker locality policy achieves the lowest AS hop count. So in streaming scenario, if less playback disruption is of high priority, we should choose choker and picker locality policies. If less inter-AS traffic is of high priority, we should use tracker locality policy.

V. Conclusion

In this paper, we propose a set of locality-aware P2P solutions. In particular, we propose an optimal solution which returns a distribution structure with the minimum AS hop count. We also modify the BitTorrent system to embed locality-awareness into its neighbor selection, peer choking/unchoking, and piece selection processes. We evaluate the performance of these solutions, as well their impacts on ISPs on a real-world Internet AS topology derived from the PlanetLab testbed. While it clearly shows the advantage of locality-aware solutions at reducing inter-AS traffic and achieving shorter downloading time, they also demonstrate deficiency at evenly distributing peer workload as done by the traditional random strategy employed by BitTorrent. We also find that continuing seeding after 10 mins seeding can not improve the downloading time of standard BitTorrent, but can significantly improve it for locality policies. It can also reduce the number of connected peers for choker and piece picker locality policies. As such, our study suggests the necessity to consider, in the design of future P2P downloading and streaming solutions, the tradeoff between the goals of optimizing AS-related performance and achieving fairness among peers such as intra-AS traffic and peer burden fairness.

References

[1] "BitTorrent," http://bittorrent.com.

[2] "Skpe," http://skype.com.

[3] "PPLive," http://pplive.com.

[4] "UUSEE," http://uusee.com.

[5] T. Karagiannis and P. Rodriguez, and K. Papagiannaki , "Should Internet service providers fear peer-assisted content distribution," in *Internet Measurement Conference*, 2005.

[6] S. Ren and L. Guo and X. Zhang, "ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP," in *ICDCS*, 2006.

[7] C. Huang and J. Li and K. W. Ross, "Can internet video-on-demand be profitable?," in *ACM SIGCOMM*, 2007.

[8] R. Bindal and P. Cao and W. Chan and J. Medved, G. Suwala and T. Bates and A. Zhang , "Improving Traffic Locality in BitTorrent via Biased Neighbor Selection," in *ICDCS*, 2006.

[9] "Planetlab," http://www.planet-lab.org.

[10] Dongyu Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2004, pp. 367–378, ACM Press.

[11] X. Yang and G. Veciana, "Service capacity of peer to peer networks," in *Proc. of INFOCOM*, 2004.

[12] Laurent Massoule and Milan Vojnovic, "Coupon replication systems," in *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, New York, NY, USA, June 2005, vol. 33, pp. 2–13, ACM Press.

[13] A. Bharambe and C. Herley and V. Padmanabhan, "Some Observations on BitTorrent Performance," 2005.

[14] L. Guo and S. Chen and Z. Xiao and E. Tan and X. Ding and X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems," 2005.

[15] Y. Chu, R. Rao, and H. Zhang, "A case for end system multicast," in *Proc. of ACM SIGMETRICS*, 2000.

[16] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[17] H. Xie and Y. R. Yang and A. Krishnamurthy and Y. Liu and A. Silberschatz, "P4P: Provider Portal for Applications," in *Proc. of SIGCOMM*, 2008.

[18] D. R. Choffnes and F. E. Bustamante, "Taming the Torrent: A Practical Approach to Reducing Cross-ISP Traffic in P2P Systems," in *Proc. of SIGCOMM*, 2008.

[19] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[20] Y. Cui, B. Li and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," *IEEE Journal on Selected Areas of Communications, Special Issue on Recent Advances in Service Overlay Networks*, vol. 22, 2004.

[21] S. Jin and A. Bestavros, "Cache-and-Relay Streaming Media Delivery for Asynchronous Clients," in *Proc. of International Workshop on Networked Group Communication (NGC)*, 2002.

[22] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou, and Kunwadee Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, New York, NY, USA, 2002, pp. 177–186, ACM Press.

[23] J. Li and Y. Cui and B. Chang, "PeerStreaming: Design and Implementation of an On-Demand Distributed Streaming System with DRM Capabilities," *ACM/Springer Multimedia Systems Journal*, 2007.

[24] C. Wu and Baochun Li, "rStream: Resilient and Optimal Peer-to-Peer Streaming with Rateless Codes," *IEEE Transactions on Parallel and Distributed Systems*, 2007.

[25] C. Gkantsidis and J. Miller and P. Rodriguez, "Anatomy of a P2P Content Distribution System with Network Coding," in *IPTPS*, 2006.

[26] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Y. S. P. Yum, "Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming," in *INFOCOM 2005: 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 2005, vol. 3, pp. 2102–2111 vol. 3.

[27] C. Dana and D. Li and D. Harrison and C.N. Chuah, "BASS: BitTorrent Assisted Streaming System for Video-on-Demand," in *IEEE MMSP*, 2005.

[28] A. Vlavianos and M. Iliofotou and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications," in *IEEE INFOCOM*, 2006.

[29] Y. Choe and D. Schuff and J. Dyaberi and V. Pai, "Improving VoD Server Efficiency with BitTorrent," in *ACM Multimedia*, 2007.

[30] "BitTorrent DNA," http://www.bittorrent.com/dna/.

[31] L. Gao, "On inferring autonomous system relationships in the internet," *IEEE/ACM Transaction on Networking*, 2001.

[32] N. Spring and R. Mahajan and David. Wetherall and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Transaction on Networking*, 2004.

[33] X. Dimitropoulos and D. Krioukov and M. Fomenkov and B. Huffaker and Y. Hyun and k. claffy and G. Riley, "AS relationships: inference and validation," in *ACM SIGCOMM*, 2007.

[34] "Cymru," http://www.cymru.com.

[35] "Caida," http://www.caida.org.

[36] "IP2Location," http://www.ip2location.com.