Multimedia Systems (2005)
DOI 10.1007/s00530-005-0191-6

REGULAR PAPER

Mohamed Hefeeda · Ahsan Habib · Dongyan Xu · Bharat Bhargava · Boyan Botev

CollectCast: A peer-to-peer service for media streaming

Published online: 14 October 2005

Abstract We present CollectCast, a peer-to-peer (P2P) service for media streaming where a receiver peer is served by multiple sender peers. CollectCast operates at the application level but infers underlying network properties to correlate end-to-end connections between peers. The salient features of CollectCast include: (1) a novel multisender selection method that exploits the performance *correlation and dependency* among connections between different candidate senders and the receiver, (2) a customization of *network tomography* techniques and demonstration of improved practicality and efficiency, and (3) an aggregation-based P2P streaming mechanism that sustains receiver-side quality in the presence of sender/network dynamics and degradation. We have performed both real-world (on PlanetLab) and simulation evaluation of CollectCast. Our simulation results show that for a receiver, CollectCast makes better selection of multiple senders than other methods that do not infer underlying network properties. Our PlanetLab experiments are performed using a P2P media streaming application (called PROMISE) which we developed on top of CollectCast. Both packet-level and frame-level performance of MPEG-4 video streaming demonstrates the practicality and effectiveness of CollectCast.

Keywords Peer-to-peer systems · Multimedia streaming

M. Hefeeda
School of Computing Science, Simon Fraser University, Surrey, BC V3T 2W1, Canada
E-mail: mhefeeda@cs.sfu.ca

A. Habib
School of Information and Management Systems,
University of California, Berkeley, CA 94702, USA
E-mail: habib@sims.berkeley.edu

D. Xu (✉) · B. Bhargava · B. Botev
Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA
E-mail: {dxu, bb, botev}@cs.purdue.edu

1 Introduction

Peer-to-peer (P2P) systems have gained tremendous momentum in recent years. In a P2P system, peers communicate directly with each other for the sharing of data as well as other resources such as storage and CPU cycles. Paralleling research in other aspects of P2P, such as lookup [23, 26, 30], storage [11, 27], and multicast [1, 9, 32], we in this paper focus on P2P media streaming, which poses more stringent bandwidth requirement than general file sharing. More specifically, as first addressed in our earlier work [33], for a media streaming session with playback rate R_0, a single sending peer may *not be able* or *willing* to contribute the full outbound bandwidth of R_0. As a result, we propose a P2P streaming model where one receiving peer *collects* media data from *multiple* sending peers – each contributing a streaming rate *lower than* R_0. We also note that downloading the entire media file before playback is not always a good alternative, because of the potentially large media file size and thus long downloading time.

Under the "multiple-to-one" P2P streaming model, we present our solution to the following challenge: in a highly diverse and dynamic P2P network, how to select, monitor, and possibly switch sending peers in a P2P streaming session, so that the best possible streaming quality can be achieved and sustained for the receiver? The dynamics and diversity are reflected in both peers and network connections between peers: (1) a sender may stop contributing to a P2P streaming session, (2) the outbound bandwidth contributed by a sender may change, (3) the connection between a sender and the receiver may exhibit changing end-to-end bandwidth and loss rate, and (4) most importantly, the underlying network topology determines that the end-to-end connections between the senders and the receiver *are not independent* of each other with respect to their bandwidth and loss rate. As a result, the quality of a P2P streaming session depends on (1) judicious selection of senders, (2) constant monitoring of sender/network status, and (3) timely switching of senders when the sender or network fails or seriously degrades. Unfortunately, previous works in P2P streaming do not systematically address these requirements. For example, some simply assume that a receiver receives media data from only *one* sender [2, 9, 32]. For works that do assume the multiple-to-one streaming model [17, 20], there is no study on the selection of *multiple senders*.

As our solution, we present *CollectCast*, a multiple-to-one P2P streaming service. CollectCast operates at the application level but infers underlying network properties (topology and performance) to *correlate* end-to-end connections between candidate senders and the receiver. Different from other multiple-to-one network services such as Concast [6], each CollectCast session involves two sets of senders: the *standby senders* and the *active senders*. Members of the two sets may change dynamically during the session. CollectCast reflects the P2P philosophy of dynamically and opportunistically aggregating limited capacity of peers to perform a task (streaming) traditionally performed by a dedicated entity (a media server).

Our main contributions in CollectCast include the following: (1) a novel multisender selection method that exploits the performance *correlation* among multiple end-to-end connections between candidate senders and receiver, (2) customization of network tomography techniques and demonstration of improved practicality and efficiency, and (3) an aggregation-based P2P streaming mechanism that sustains receiver-side quality in the presence of sender/network dynamics and degradation. We have performed both PlanetLab-based and simulation-based evaluation of CollectCast. Our simulation results show that for a receiver, CollectCast makes better selection of multiple senders than methods that do not infer underlying network properties. Our PlanetLab experiments are performed using a P2P media streaming application (called PROMISE) which we developed on top of CollectCast. Both packet-level and frame-level performance of MPEG-4 video streaming demonstrate the practicality and effectiveness of CollectCast.

The rest of the paper is organized as follows. An overview of CollectCast describing its main components and operations is given in Sect. 2. The following four sections present key aspects of CollectCast design: *multisender selection* in Sect. 3, *topology inference and labeling* in Sect. 4, *rate and data assignment* in Sect. 5, and *monitoring and adaptation* in Sect. 6. We evaluate CollectCast via simulation in Sect. 7. Section 8 describes an application (PROMISE) built on top of CollectCast and presents the performance results obtained from running PROMISE on PlanetLab nodes. Section 9 discusses related work, and Sect. 10 concludes the paper.

2 Overview of CollectCast

CollectCast is proposed as a P2P transport service supporting distributed streaming applications. As shown in Fig. 1, the CollectCast service is layered on top of a P2P lookup substrate and is comprised of four main components: (1) topology inference and labeling, (2) multisender selection, (3) rate and data assignment, and (4) monitoring and adaptation. Operations of these components are initiated and coordinated from the receiver-side in a streaming session.

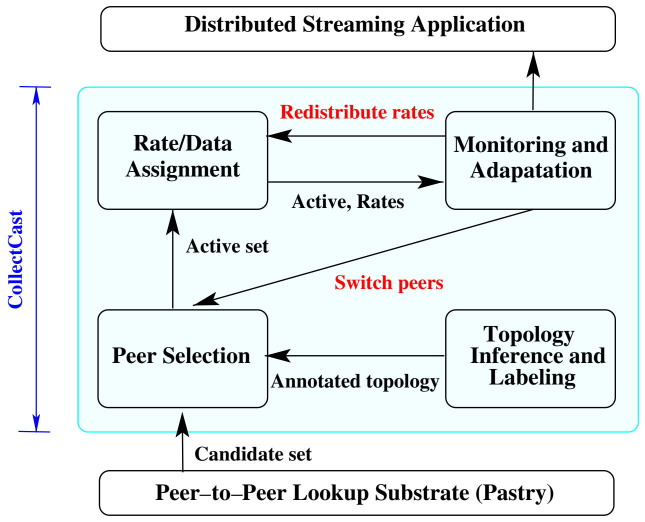

Fig. 1 Main components of CollectCast and their interactions. Numbers on the *arrows* indicate a typical sequence of establishing a streaming session.

The P2P lookup substrate manages peer membership and performs object look up. CollectCast is designed to be independent of the P2P substrate. Therefore, it can use substrates such as Pastry [26], Chord [30], or CAN [23]. We choose Pastry in CollectCast implementation because Pastry has an open-source Java implementation [13] with good portability. We note that current P2P lookup methods return one peer for each object lookup request. In our implementation, Pastry is extended to return multiple peers for each lookup request.

A CollectCast streaming session can be established in four steps, as shown in Fig. 1: (1) The receiver issues a lookup request to the P2P lookup substrate, which returns a set of *candidate sender peers* that have the requested media file. The set of candidates is usually small. In fact, as shown in our performance evaluation (Sect. 7.5), a set of no more than 20 candidates warrants good sender selection. (2) The topology inference and labeling component is invoked to *infer and annotate* the underlying topology that connects the candidate peers to the receiver. The topology is annotated with available bandwidth and packet loss rate on different segments. (3) The peer selection component uses the annotated topology to select the *active senders* from the candidate peers, such that the receiver obtains full streaming rate aggregated from the active senders. The rest of the candidate peers are kept in a *standby* sender set. (4) The rate and data assignment component is called to determine the rate and data assignment to each active sender.

Once the rates and data are assigned, the receiver establishes parallel connections, each from an active sender to the receiver. Two types of connections are established: A

UDP connection for data streaming[1] and a TCP connection for exchanging control messages. The monitoring and adaptation component oversees the streaming session to maintain receiver-side aggregate streaming rate. It measures the streaming rate and packet loss rate from each active sender. If the rate from a sender degrades due to peer failure or network congestion, the monitoring and adaptation component will try to redistribute the rate among the remaining active peers. If such redistribution is not possible, a peer switching will be performed by the peer selection component to replace poor-performing peer(s) with peer(s) from the standby set.

3 Multisender selection in CollectCast

The key component of CollectCast is the selection of active senders. This section presents the proposed tomography-based multisender selection method and discusses its advantages over methods that only use end-to-end performance observation between the candidate senders and the receiver. In particular, we show that for *multisender* selection, it is important to avoid selecting senders that share the same low-performance underlying network path.

In CollectCast, our tomography-based selection method infers the underlying network topology to find active senders that lead to the best aggregate streaming rate on the receiver side. For the sake of comparison, we also consider the noncorrelating *end-to-end* selection method commonly used in wide-area server selection [29, 35]. The end-to-end selection method probes the end-to-end "sender-receiver" connections without inferring the underlying network, treating these connections as totally *independent*.

To illustrate the two selection methods, consider the example shown in Figs. 2 and 3. Suppose that the candidate sender set consists of $P_1, P_2, \ldots, P_6$ while the receiver peer is P_r. In Fig. 2, the noncorrelating end-to-end method observes the quality of end-to-end connections from the candidate senders to the receiver. As a simplification in this example, the quality is in terms of available end-to-end bandwidth. Based on the end-to-end bandwidth probing results (as shown in the figure), the method selects peers P_3, P_5, P_6. However, this selection is not aware of the shared segment on the two network paths $P_5 \rightsquigarrow P_r$ and $P_6 \rightsquigarrow P_r$, which *cannot* support the aggregate rate from P_5 and P_6. As shown in Fig. 3, the tomography-based method infers the underlying network topology as well as the available bandwidth on different segments. As a result, it is able to make a more appropriate selection: P_2, P_3, and P_6. The selection decision will be explained in detail in Sect. 3.2.

3.1 Peer model

Before presenting the tomography-based multisender selection method, we first define our peer model. Peers are het-

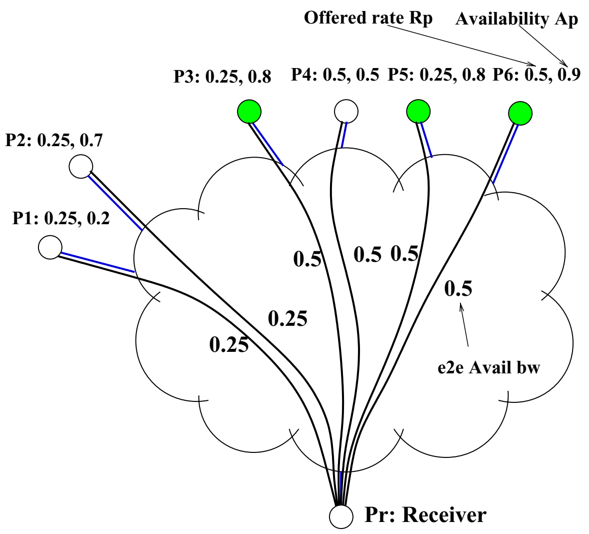

Fig. 2 Noncorrelating end-to-end sender selection. It treats sender – receiver connections as independent.

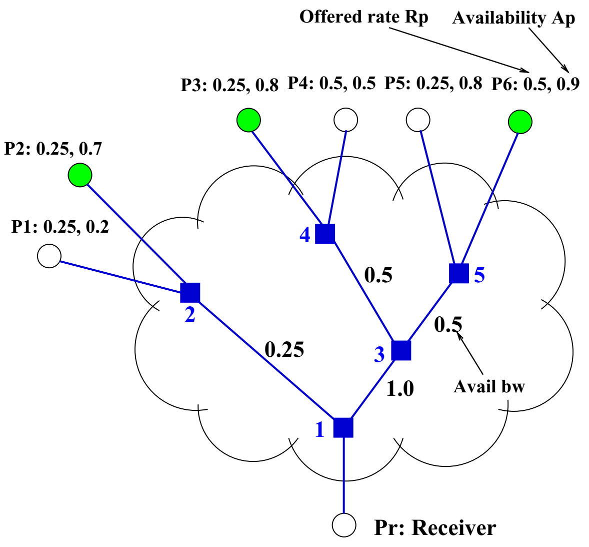

Fig. 3 Tomography-based sender selection in CollectCast. It constructs a virtual tree topology that correlates the sender – receiver connections

erogeneous in their capacity and availability and they do not exhibit server-like behavior. A peer p has two parameters: *offered rate* and *availability*. The offered rate R_p is the maximum sending rate that a peer can (or is willing to) contribute at any time. A lower bound on the offered rate ($R_\mathrm{p}^\mathrm{min}$) is imposed by the system to limit the maximum number of senders required to serve a CollectCast session, thus limiting the number of connections that a receiver needs to maintain. The offered rate is specified by the user when joining the system and adjusted by a CollectCast daemon if the first-hop network connection of p is lower than the user-specified rate.

The availability is a function that describes how individual peers use their machines, and hence how available the

[1] Adjusting the rate of the UDP connection to compete fairly with TCP traffic of other applications is discussed in [14].

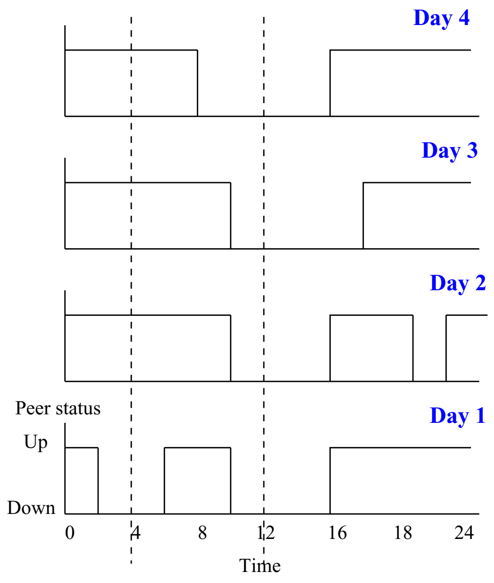

Fig. 4 The status of an example peer over a 24-hour period for 4 consecutive days

machines are to serve others. The availability varies with time of the day and from one peer to another. For instance, one peer may connect to the Internet and participate in the streaming system in the evening (e.g., between 6:00 PM and 10:00 PM) and then disconnect. Another peer may stay connected 24 hours, but use most of its upstream bandwidth between 8:00 AM and 9:00 AM. The availability information helps the selection algorithm to avoid soon-to-be-down peers. In order to model a peer's availability, a software daemon running on that peer collects information on the status of the peer over a moving window of several days (e.g., 1 week). The status is either "up" or "down". The meaning of "up" is that the peer is connected and can serve a request, while "down" means that the peer is not available because it is offline or is using its entire bandwidth. Figure 4 shows the status of a peer over 24-hour period for 4 consecutive days. Given the status information, we define the availability at time t as a random variable $\boldsymbol{A}_\mathrm{p}(t)$ which takes on two values: 1 ("up") and 0 ("down"). The probability distribution of $\boldsymbol{A}_\mathrm{p}(t)$ is determined by sampling the status information over several days at time t. For example, for $t = 4$, we see from Fig. 4 that the peer was up in days 2, 3, and 4 and was down in day 1. Thus, the peer is expected to be up with probability 3/4 and down with probability 1/4 at $t = 4$.

At the beginning of the selection process, each candidate peer reports to the receiver its offered rate and availability. As to be explained in the next section, the receiver will use the expected value of sender availability in the calculations. Therefore, only $E[\boldsymbol{A}_\mathrm{p}]$ value will be reported, not the entire distribution. Moreover, to be safe in the selection, a candidate peer reports the *minimum* expected availability value during the upcoming streaming session.

3.2 Tomography-based selection method

The input of the multisender selection method is the set of candidate senders returned by the P2P lookup substrate. The next step is performed by the topology inference and labeling component, which applies *network tomography* techniques to infer the topology connecting the candidate senders and the receiver. This step further annotates segments of the topology with network performance metrics. Details of this step will be presented in Sect. 4. The result of this step is a tree topology rooted at the receiver, as shown in Fig. 3. Note that, an edge in the inferred tree represents a network path segment, which may actually be a sequence of physical network links. Collapsing several links into one segment yields a compact representation of the topology. We also assume that routes from the candidate senders to the receiver do not change during the streaming session.

The last step of the multisender selection method further processes the inferred tree which we denote as $\mathcal{T}$. Table 1 lists the symbols used in the section. Each leaf node of $\mathcal{T}$ represents a peer p in the set of candidate senders $\mathbb{P}$ and has two attributes: offered rate R_p and availability $\boldsymbol{A}_\mathrm{p}$. An edge (or "segment") between node i and node j is denoted by $i \to j$, and is labeled with a quality metric $\boldsymbol{g}_{i\to j}$. We define the segment quality as a function of both packet loss rate and available bandwidth because they can be measured segment-wise [3] and they both impact the receiving rate, and hence impact the streaming quality. The quality of segment $i \to j$ is given by:

$$\boldsymbol{g}_{i\to j} = w_{i\to j}(1 - \boldsymbol{l}_{i\to j}), \tag{1}$$

where $\boldsymbol{l}_{i\to j}$ is the packet loss rate on $i \to j$, and $w_{i\to j}$ is a *weight* determined by the available bandwidth on $i \to j$ and the aggregate rate from peers sharing this segment if they are selected in the active set. The weight of segment $i \to j$ for a peer p is denoted by $w^{(p)}_{i\to j}$ and is given by:

$$w^{(p)}_{i\to j} = \min\left(1, \max\left(0, \left(b_{i\to j} - \sum_{s\in S, i\to j\in s\rightsquigarrow r} R_s\right)\Big/ R_\mathrm{p}\right)\right), \tag{2}$$

Table 1 List of symbols used in the paper

Symbol	Definition
R_p	Maximum sending rate from peer p
R_0	Playback rate of requested media data
$\mathbb{P}$	Set of candidate sender peers
$\boldsymbol{A}_\mathrm{p}$	Availability of peer p
$\boldsymbol{g}_{i\to j}$	Quality of network path segment $i \to j$
$\boldsymbol{l}_{i\to j}$	Packet loss rate of network path segment $i \to j$
$w^{(p)}_{i\to j}$	A weight factor determined by the available bandwidth on network path segment $i \to j$ for peer p
$\boldsymbol{G}_\mathrm{p}$	Quality of peer p in the session being established
α	Packet loss tolerance level

where $b_{i\to j}$ is the available bandwidth on segment $i \to j$, S is the set of peers selected to be in the active set thus far, and $s \rightsquigarrow r$ is the path from the sending peer s to the receiving peer r. The intuition behind this formulation is that, if a segment has a bandwidth equal to or higher than the aggregate rate contributed from peers sharing this segment, then this segment will not throttle this aggregate rate, and hence its weight is set to 1. Otherwise, the weight is a fraction proportional to the shortage in the bandwidth if peer p along with peers in S are chosen to serve. The example given later in this section explains numerically how to compute these weights.

In the final step of the peer selection process, we formulate the selection problem as an optimization problem stated as follows. Given the inferred tree $\mathcal{T}$, find the set of active peers $\mathbb{P}^{\text{actv}} \subseteq \mathbb{P}$ that maximizes the expected aggregated rate at the receiver, provided that the receiver inbound bandwidth is not exceeded. Mathematically, this can be formulated as: find $\mathbb{P}^{\text{actv}}$ that

$$\text{Maximizes} \quad E\left[\sum_{p\in\mathbb{P}^{\text{actv}}} \boldsymbol{G}_{\text{p}} R_{\text{p}}\right] \tag{3}$$

$$\text{Subject to} \quad R_{\text{l}} \leq \sum_{p\in\mathbb{P}^{\text{actv}}} R_{\text{p}} \leq R_{\text{u}}, \tag{4}$$

where R_{p} is the offered rate of peer p, R_{l} and R_{u} are the lower and upper rate targets (Sect. 5 shows how they are determined), and $\boldsymbol{G}_{\text{p}}$ is the quality of peer p for the streaming session. $\boldsymbol{G}_{\text{p}}$ is a function of the availability of peer p and the quality of all segments comprising the path $p \rightsquigarrow r$, and is given by:

$$\begin{aligned} \boldsymbol{G}_{\text{p}} &= \boldsymbol{A}_{\text{p}} \prod_{i\to j\in p\rightsquigarrow r} \boldsymbol{g}_{i\to j} \\ &= \boldsymbol{A}_{\text{p}} \prod_{i\to j\in p\rightsquigarrow r} w^{(p)}_{i\to j}(1-\boldsymbol{l}_{i\to j}). \end{aligned} \tag{5}$$

A high expected quality value (close to 1) indicates that the peer is likely to provide good and sustained sending rate: it is unlikely to stop sending packets and the packets will be transmitted through network paths of low dropping probability.

Given the problem formulation above, finding the best active set $\widehat{\mathbb{P}}^{\text{actv}}$ is straightforward. Figure 5 describes an algorithm to determine $\widehat{\mathbb{P}}^{\text{actv}}$ given the inferred tree $\mathcal{T}$. The algorithm starts by enumerating all possible peer sets that satisfy the constraints in Eq. (4). Different orderings of peers are considered different sets. The number of possible peer sets is not too large, because the input (the candidate set) to the algorithm is fairly small (10–20 peers, as our simulation shows in Sect. 7.5) from which we choose 3–5 active peers (also shown in Sect. 7.5). Many sets will be immediately disqualified by the constraints. For each qualified set, the algorithm determines the expected aggregate rate (lines 5–13 in Fig. 5) as follows. The expected quality ($E[\boldsymbol{G}_{\text{p}}]$) of each peer in the set is computed using Eqs. (2) and (5), which are

Selection Algorithm

```
1.  Enumerate all sets that satisfy constraints in Eq. (4): $\mathbb{P}^1, \mathbb{P}^2, \ldots, \mathbb{P}^M$
2.  $\widehat{\mathbb{P}}^{actv} = null;\ \widehat{ER} = 0$
3.  for each $\mathbb{P}^m, 1 \leq m \leq M$ do
4.      Set $dif_{i\to j} = b_{i\to j}, \forall i \to j \in \mathcal{T}$
5.      $ER = 0$
6.      for each $p \in \mathbb{P}^m$ do
7.          $\overline{\boldsymbol{G}_p} = \overline{\boldsymbol{A}}_p$
8.          for each segment $i \to j \in p \rightsquigarrow r$ do
9.              $\overline{\boldsymbol{G}_p} = \overline{\boldsymbol{G}_p} \times \min(1, (1-\bar{\boldsymbol{l}}_{i\to j}) \times dif_{i\to j}/R_p)$
10.             $dif_{i\to j} = \max(0, dif_{i\to j} - R_p)$
11.         endfor
12.         $ER = ER + \overline{\boldsymbol{G}_p} \times R_p$
13.     endfor
14.     if $ER < \widehat{ER}$ then
15.         $\widehat{ER} = ER$
16.         $\widehat{\mathbb{P}}^{actv} = \mathbb{P}^m$
17. endfor
18. return $\widehat{\mathbb{P}}^{actv}$
```

Fig. 5 Pseudo code for selecting the best active sender set

implemented by line 7 and the for loop in lines 8–11. Then, the expected quality of the peer multiplied by its offered rate is added to the expected aggregate rate of the set (line 12). The algorithm finds the set with the maximum expected aggregate rate (lines 14–16) and returns it. The complexity of the algorithm is not a concern, because: (1) the input size is small, and (2) the algorithm is invoked only a few times: at the beginning of the session and when a peer switching is needed. Therefore, although designing more efficient selection algorithms is possible, we believe that the payoff will not be significant.

To demonstrate how the tomography-based selection technique works, we provide the details of selecting the best peers in the topology shown in Fig. 3. We set $R_{\text{l}} = R_{\text{u}} = R_0$ and the packet loss rate in all path segments to 0 to simplify the discussion. The playback rate R_0 is 1 Mb/s. The possible active sets that satisfy the constraints in Eq. (4) are: $\{P_4, P_6\}$, $\{P_3, P_5, P_6\}$, $\{P_2, P_5, P_6\}$, $\{P_1, P_5, P_6\}$, $\{P_3, P_4, P_5\}$, $\{P_2, P_4, P_5\}$, $\{P_1, P_4, P_5\}$, $\{P_1, P_3, P_4\}$, $\{P_2, P_3, P_4\}$, $\{P_2, P_3, P_6\}$, $\{P_1, P_3, P_6\}$, $\{P_1, P_2, P_4\}$, $\{P_1, P_2, P_6\}$, and $\{P_1, P_2, P_3, P_5\}$. Note that, different orderings of peers are not shown, but are considered in the calculations. The expected aggregated rate is then computed for every set. For instance, the expected aggregated rate for $\{P_3, P_5, P_6\}$ is $1 \times .8 + 1 \times .8 + .25/.50 \times .9 = 2.05$. P_5 and P_6 have a shared segment ($5 \to 3$) of bandwidth .5. If we assign $w^{(P_5)}_{5\to 3} = 1$ (because the available bandwidth on the path is greater than P_5's offered rate), P_6 will get a left-over bandwidth of 0.25, which makes the weight $w^{(P_6)}_{5\to 3} = 0.25/0.50$. If we assign the $w^{(P_6)}_{5\to 3} = 1$, P_5 gets a weight of 0 because no bandwidth is left for this peer on the shared segment. The expected rate of all possible sets are 1.4, 2.05, 1.95, 1.45, 1.85, 2.0, 1.5, 1.75, 1.25, 2.4, 1.9, 1.2, 1.6, and 2.3, respectively. The highest aggregate rate (2.4) comes from the set $\{P_2, P_3, P_6\}$.

3.3 End-to-end selection method

For comparison with the tomography-based sender selection method, we describe a selection technique which we refer to as the end-to-end selection method. The end-to-end selection exploits no information about the path segments shared among peers. It uses the end-to-end path available bandwidth and packet loss rate in addition to peer availability in choosing peers. While end-to-end selection may impose less probing overhead, it does not perform as well as the tomography-based selection, as demonstrated by our performance results in Sects. 7 and 8.

We can formulate the end-to-end selection as a special case of the tomography-based selection as follows. Instead of writing the peer quality as in Eq. (5), we write it as: $\boldsymbol{G}_\mathrm{p} = \boldsymbol{A}_\mathrm{p} w_{p\rightsquigarrow r}(1-\boldsymbol{l}_{p\rightsquigarrow r})$, where $w_{p\rightsquigarrow r}$ is the *path* weight and $\boldsymbol{l}_{p\rightsquigarrow r}$ is the end-to-end packet loss random variable. Computing the path weight is much easier in this case and is given by:

$$w_{p\rightsquigarrow r} = \begin{cases} 1, & R_\mathrm{p} \le b_{p\rightsquigarrow r} \\ \frac{R_\mathrm{p} - b_{p\rightsquigarrow r}}{R_\mathrm{p}}, & \text{otherwise.} \end{cases} \tag{6}$$

Using this formulation, the expected rate maximization problem can be solved in a way similar to the one in Sect. 3.2. To show how the calculations are performed, we solve the example in Sect. 3.2 using the end-to-end selection techniques. The possible active sets are still the same. The end-to-end selection utilizes the availability of peers and the path available bandwidth to calculate the expected rate. For example, the expected rate of the set $\{P_3, P_5, P_6\}$ is $1 \times .8 + 1 \times .8 + 1 \times .9 = 2.5$. The corresponding expected rate of all possible sets are 1.4, 2.5, 2.4, 1.9, 2.1, 2.0, 1.5, 2.0, 1.5, 2.4, 1.9, 1.4, 1.8, and 2.5, respectively. The maximum expected rate is 2.5, which is supplied by peer sets $\{P_3, P_5, P_6\}$ and $\{P_1, P_2, P_3, P_5\}$. Either of them can be taken, but we prefer the set with fewer peers to reduce the overhead of maintaining multiple concurrent connections.

4 Topology inference and labeling in CollectCast

The tomography-based selection algorithm of CollectCast (Sect. 3.2) relies on the inferred tree $\mathcal{T}$, which is a transformed version of the topology inferred and labeled through end-to-end probing techniques. Discovering the interior characteristics of the network by probing from its end points is called network tomography [10]. In this section, we describe our approach to inferring and labeling an approximate topology sufficient for peer selection. The contributions of this section are twofold. First, we modify current network tomography techniques to achieve significant reduction in probing overhead and convergence time. Second, we show through a concrete example (i.e., P2P streaming) the potential performance gain by applying network tomography techniques to distributed applications.

The inferred tree topology is constructed in three steps. In the first step, traceroute is used to build the physical topology connecting the candidate senders with the receiver. Then, consecutive links with no branching points are merged into one segment. We note that some routers do not support traceroute. This, however, does not severely harm the technique because we are not interested in the exact topology, but in the shared segments among peers.

In the second step, the inferred topology is annotated with available bandwidth. The end-to-end available bandwidth of a path is defined as the maximum rate that this path can provide to a flow, without reducing the rate of other traffic [15]. The link with the minimum available bandwidth (i.e., the tightest link) determines the path available bandwidth. Measuring the path available bandwidth is costly: one should keep increasing the probing traffic rate till at least it reaches (and probably exceeds) the available bandwidth on the tightest link. Measuring available bandwidth on individual path segments is even more costly. In CollectCast, we are not interested in the exact available bandwidth if it exceeds the media streaming rate. Therefore, we tradeoff the unnecessary accuracy for far less bandwidth measurement overhead.

We modify the basic available bandwidth measurement technique proposed by Jain and Dovrolis [15]. The authors show that the one-way delay difference of a periodic packet stream is a good indication of the end-to-end available bandwidth. The idea is that if the streaming rate is higher than the available bandwidth, the one-way delay difference will show a trend of increase. On the other hand, if the streaming rate is lower than the available bandwidth, the one-way delay difference will be zero. To measure the available bandwidth, the sender sends a stream of packets with a specific rate. The receiver measures the trend in the delay difference and decides whether the next stream rate should be increased or decreased by a factor of 2. The procedure continues till the available bandwidth estimate is within the desired range of accuracy. We make several adaptations to the basic procedure. First, instead of measuring the path available bandwidth, we test whether a path can accommodate the aggregated rate from peers sharing this path, which is at most R_0. We set the initial rate of the probing stream to the minimum possible offered rate ($R_\mathrm{p}^{\min}$), and we terminate whenever the stream rate reaches the minimum of R_0 and the aggregate rate from peers sharing the path. Second, since one peer may not be able to send at rate R_0, we coordinate the probing from multiple peers to get the same effect as probing from one sender. Third, we conservatively label all segments of a path with the value of its tightest segment. Finally, we use the actual media data to generate probing traffic (i.e., data from the media file that needs to be sent anyway).

To illustrate, consider the topology in Fig. 3. Let us estimate the bandwidth on the path segment $5 \rightarrow 3$. Peer P_5 sends a stream of packets (say 100 packets) with rate $R_0/8$. The receiver P_r notices that the delay differences are 0. Then P_5 increases its rate to $R/4$. Still no increasing trend in the delay differences, but P_5 cannot increase its rate anymore. Now P_r triggers P_6 to start sending at $R_0/4$ while P_5 is still sending, making the aggregate rate crossing $5 \rightarrow 3$ to be $R_0/2$. P_r measures the delay differences for the packet stream coming from P_5, that is, the stream coming from P_6 is considered as cross traffic to reduce the available bandwidth seen by P_5. P_6 keeps increasing its rate till it reaches its maximum ($R_0/2$) or P_r notices increasing delay differences. If the former happens, segment $5 \to 3$ will be considered to have an available bandwidth of $0.75R_0$, even though its actual available bandwidth might be higher. In the latter case, the actual available bandwidth on $5 \to 3$ can be determined as $0.5R_0$. The available bandwidth on $4 \to 3$ and $2 \to 1$ can be measured in a similar way. To measure the available bandwidth on $3 \to 1$, P_r will coordinate the probing from P_3, P_4, P_5 and P_6. As a final note, suppose that the available bandwidth on $3 \to 1$ is less than that on $5 \to 3$, say $R_0/4$. In this case, our technique will underestimate the available bandwidth on $5 \to 3$ because P_r will see increasing delay difference due to the tight link $3 \to 1$. This conservative estimation will make the expected rate computed from Eq. (3) even worse for the set of peers that share a tight segment, helping the selection algorithm to avoid them as output.

The third step in constructing the inferred tree is to annotate it with packet loss rate. Instead of explicitly probing for segment-wise loss rates, we leverage the information obtained during available bandwidth measurements. The receiver assigns the sending rate to each of the sending peers. It also determines which data packets should be sent by each peer. Therefore, it is easy to determine the loss rates on individual end-to-end paths. To compute the segment-wise loss rates, we use the recently proposed Bayesian inference using Gibbs sampling method [19]. The method models the network tomography (for segment-wise loss rates) as a Bayesian inference problem. Then, using the measured data and setting an initial distribution for the segment losses, the method iteratively computes the posterior distribution of the segment losses [19].

Constructing the inferred tree imposes two types of overhead: processing and communication. The communication overhead is due to sending probing packets. Since we send actual data packets as probes, we effectively introduce no communication overhead. The receiver, though, needs a larger buffer (in the order of seconds) to store these data packets for later use. The processing overhead is mainly due to topology inference and peer selection. This is not a major concern, given that the topology will typically be very small (10–20 nodes). We note that building the topology and determining the best active set will increase the start up delay of a streaming session. However, it is still in the order of seconds. Finally, the need for updating the topology will be infrequent, since the active set is expected to last for a relatively long period, because: (1) peers in this set are carefully chosen and are likely to have high availability, and (2) several Internet measurement studies (e.g., [36]) have shown fairly good stability of path properties such as loss, delay, and throughput.

5 Rate and data assignment in CollectCast

This section describes how CollectCast coordinates the active peers by assigning appropriate rate and media data to each of them.

Each media file is divided into data segments, each with a size of Δ packets. Data segments are encoded using forward error correction (FEC) to tolerate packet losses during transmission. FEC codes such as Reed–Solomon codes and Tornado codes [4] can be used. We use Tornado codes because they are faster to encode/decode, albeit with little decoding inefficiency [4]. We use the notation FEC(α) to indicate that up to $(\alpha - 1) \times 100\%$ packet loss rate can be tolerated. For instance, FEC(1.25) means that a data segment will be successfully reconstructed even if 25% of the packets are lost. α is the parameter that defines the current packet loss tolerance level in the system. α has two bounds: α_u, α_l, which are the maximum and minimum loss tolerance levels, respectively. These bounds impact the selection of active peers determined by solving the maximization problem (Sect. 3.2) because the bounds R_l, R_u in Eq. (4) are computed as: $R_u = \alpha_u R_0$ and $R_l = \alpha_l R_0$.

Data segments stored at peers are preencoded using FEC(α_u). A segment of Δ packets is encoded into $\Delta/(2 - \alpha_u)$ packets. For instance, FEC(1.25) on a segment of 120 packets results in 160 encoded packets, from which any 120 packets can reconstruct the original segment. Even though data segments are pre-encoded with α_u, we do not send at aggregated streaming rate of $\alpha_u R_0$ at all time. Instead, we adapt the sending rate dynamically based on the expected packet loss rate. Therefore, the number of redundant packets sent is proportional to the current packet loss rate. If the packet loss rate is low, only a small number of extra packets will be sent to save network bandwidth. The sending rate is set to αR_0, where $\alpha_l \le \alpha \le \alpha_u$ and α is given by:

$$\alpha = \max(\alpha_l, 1 + \min(\alpha_u, 1 + \overline{L}_{\sum})). \tag{7}$$

$\overline{L}_{\sum}$ is the the current expected aggregated loss rate and is computed as $\overline{L}_{\sum} = \sum_{p \in \mathbb{P}^{\text{actv}}} \overline{l}_{p \rightsquigarrow r} R_p / \sum_{p \in \mathbb{P}^{\text{actv}}} R_p$, where $\overline{l}_{p \rightsquigarrow r}$ is the expected loss rate on the path $p \rightsquigarrow r$. After computing the appropriate rate (αR_0), each peer p is assigned an actual sending rate $\widehat{R}_p$ proportional to its offered rate:

$$\widehat{R}_p = \frac{\alpha R_0}{\sum_{x \in \mathbb{P}^{\text{actv}}} R_x} R_p. \tag{8}$$

The active peers cooperate in sending every segment of the media file. Note that, since the active peers send at rate αR_0, they send only $\Delta/(2 - \alpha)$ packets out of the stored $\Delta/(2 - \alpha_u)$ packets. Each peer p is assigned a number of packets D_p to send in proportion to its actual streaming rate:

$$D_p = \left\lceil \frac{\Delta}{(2-\alpha)} \frac{\widehat{R}_p}{\alpha R_0} \right\rceil. \tag{9}$$

Example. Let $\alpha_l = 1.0625$, and $\alpha_u = 1.25$. Assume that the media file is divided into segments of 120 packets each. Encoding with FEC($\alpha_u = 1.25$), each encoded segment will have 160 packets. Suppose that the current active set has three peers P_1, P_2, P_3 with offered rates $R_{P_1} = R_0/2$, $R_{P_2} = R_0/4$, $R_{P_3} = R_0/2$, respectively. Assume that the current estimated α is 1.125. The assigned rates are: $\widehat{R}_{P_1} = 0.45$, $\widehat{R}_{P_2} = 0.225$, $\widehat{R}_{P_3} = 0.45$. The number of packets that need to be sent is 138, and the data assignment is: $D_{P_1} = 55$, $D_{P_2} = 28$, $D_{P_1} = 55$. P_1 sends packets with sequence numbers from 1 to 55, P_2 from 56 to 83, and P_3 from 84 to 138.

6 Monitoring and adaptation in CollectCast

Changes may occur during a CollectCast streaming session: peers may fail or network paths may become congested. To maintain good streaming quality on the receiver side, CollectCast performs dynamic adaptations. During the session, the receiver collects statistics on the loss rate and streaming rate from each sending peer. These statistics are used to update the inferred tree, which may later be used to *adjust* the active set.

6.1 Peer failure

A peer failure is detected in two ways: (1) from the TCP control channels established between the receiver and the sending peers (e.g., connection reset), and (2) from significant sending rate degradation. Once a failure is detected, the active set is adjusted by replacing the failed peer with new one(s). We choose the replacement peers using the tomography-based selection method (Sect. 3.2), provided that the currently good peers are part of the new active set. This may not yield the optimal solution, but it is more practical for two reasons. First, a freshly computed sender set may be totally different from the old one, which would require tearing down all current connections and establishing new ones. Second, the inferred tree may be outdated: for the standby peers, we use information gathered at the beginning of the streaming session. Thus, it is better to keep peers that are currently doing well. After determining the new active set, the receiver sends a control packet containing the new rate and data assignments to each active sender.

6.2 Network fluctuations

The receiver makes a switching decision after receiving each segment of the media file. A segment is in the order of a few seconds. Switching means one of two actions: (1) assigning new rates to the currently active peer set, or (2) adjusting the active set by adding or replacing peers. After receiving a segment, the receiver computes $\gamma = (R_\Sigma - R_0)/R_0$, where R_Σ is the aggregate rate measured during the last segment. A value of $\gamma < 0$ means that the network is dropping more than the current loss tolerance level α allows. In this case, the receiver tries to increase α to reach the desired R_0. It computes a new value for α using the updated topology. If the new α exceeds the upper bound α_u, a new active set is selected using the tomography-based selection method. Otherwise, a new rate and data assignment is computed using the new α. If γ is positive but less than a threshold (e.g., 0.1), no adaptation is needed. If γ is larger than the threshold, a new lower α will be adopted to reduce the FEC overhead.

7 Simulation results

In this section, we evaluate the performance of CollectCast using extensive simulations. We first present the setup and parameters used in the simulation. We then compare the performance of the tomography-based selection method with the end-to-end and random selection methods. The performance metrics are the aggregated streaming rate and packet loss rate on the receiver side. Finally, we assess the impact of peer availability on the size of the candidate peer set and estimate the average size of the active sender set during a streaming session.

7.1 Simulation setup

We simulate a hierarchical network topology with three levels. The highest level is composed of transit domains, which represent large Internet Service Providers (ISPs). Stub domains, which represent small ISPs, campus networks, moderate-size enterprise networks, and similar networks, are attached to the transit domains at the second level. Some links may exist among stub domains. At the lowest level, the end hosts (peers) are connected to stub routers. The first two levels are generated using the GT-ITM topology generation tool [5]. We then probabilistically add hosts to stub routers. Each experiment was run on several different topologies. The topologies used in the experiments have, on average, 600 routers and 1,000 hosts (peers). Imposing cross traffic over such a large topology is not feasible. Instead, we approximate the effect of cross traffic by: (1) attaching a stochastic loss model to the links, and (2) randomly setting the links' bandwidth. We use the two-state Markov loss model (a.k.a Gilbert model), which has been shown to model the Internet packet losses with a reasonable accuracy [16, 34]. In this model, the loss process is modeled as a Markov chain with two states: good and bad. In the good state, the probability of losing packets is very small and typically assumed to be zero. In the bad state, the probability of losing packets is assumed to be 1.0. The model has two parameters, which are the transition probabilities between the good and bad states. The available bandwidth on each link is chosen uniformly at random in the range $[0.25R_0, 1.5R_0]$. Peer's parameters are chosen to reflect the diversity in the P2P community [28]. The expected availability of peers ($\boldsymbol{A}_\mathrm{p}$) is distributed uniformly in the range $[0.1, 0.9]$. The offered rate (R_p) is also distributed uniformly in the range $[0.125R_0, 0.5R_0]$. The streaming session lasts for 60 min and the streaming rate R_0 is 1 Mb/s.

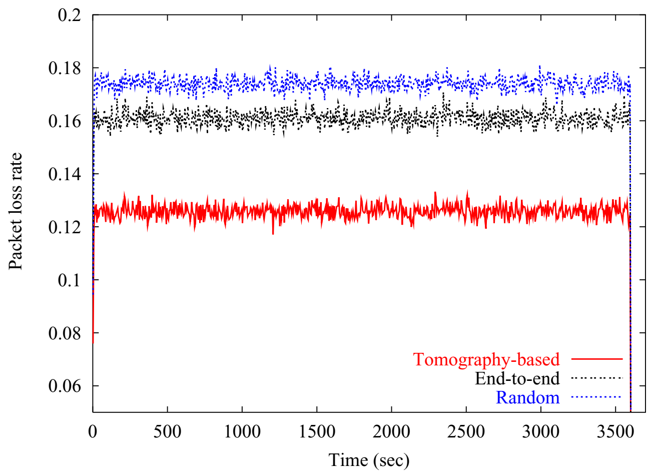

Fig. 6 Aggregated loss rate perceived by the receiver: no peer failures.

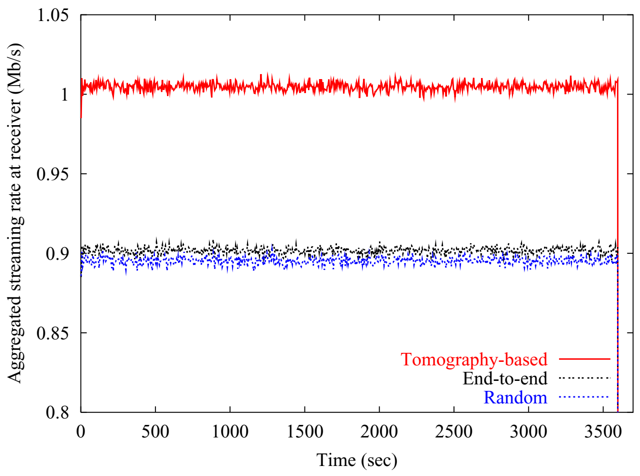

Fig. 7 Aggregated streaming rate at the receiver: no peer failures.

7.2 Performance of tomography-based selection

This section demonstrates the effectiveness of the tomography-based sender selection method. Two scenarios are simulated. In the first scenario, there are no peer failures, while in the second scenario, there are peer failures and switchings.

We simulate a streaming session as follows. First, we randomly select a number of candidate peers (e.g., 20 peers) and a receiver from the the 1,000-peer community. Then, we select the active sender set using either the random, end-to-end, or tomography-based selection method (Sect. 3). Each session is run three times with the same parameters, albeit each run with a different peer selection algorithm. Peers in the active set start streaming till a switching is needed. The loss tolerance level α_u is set to 1.2.

7.3 Results with no peer failures

Figure 6 depicts the aggregate loss rate seen by the receiver under the three selection methods. The tomography-based method achieves lower loss rate (13%) than end-to-end (17%) and random (18%) methods. The aggregated loss rate is high in this experiment because we set the available bandwidth on the links in the range [0.25, 1.5] Mb/s in order to stress the selection techniques. The aggregated streaming rate perceived by the receiver is shown in Fig. 7. The tomography-based technique yields a steady rate of 1.0 Mb/s, which achieves full-quality playback. The end-to-end technique performs better than the random technique. However, neither of them can achieve the rate for full-quality playback.

7.4 Results with peer failures

We simulate peer failures as follows. We schedule a fixed number of failure "trials" at random times throughout the streaming session. Upon each failure trial, a peer is selected randomly from the active set and we fail it probabilistically according to its availability: we generate a random number between 0 and 1. If this number is greater than the peer's expected availability, the peer fails. Otherwise, the peer remains active and the session continues normally till the next failure trial. The intuition behind this failing method is that if we have many failure trials, each peer will get enough opportunities to be tested. The fraction of "no-failure" trials will approximately be its expected availability. Figures 8 and 9 show the aggregated loss rate and the aggregated streaming rate, respectively, in the presence of peer failures. The tomography-based method performs better than the other two methods, achieving a lower loss rate while maintaining full quality. Note that, in Fig. 9, the aggregated rate is slowly decreasing as the session progresses. This is because as the time elapses, more peers fail and the selection method is left with fewer peers in the standby set to choose from.

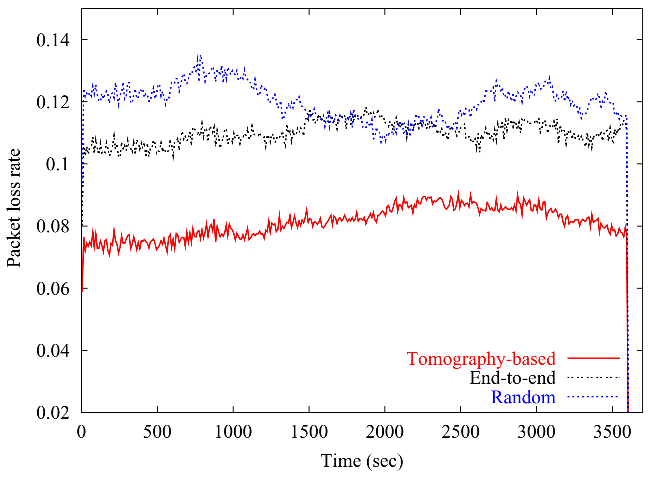

Fig. 8 Aggregated loss rate perceived by the receiver: with peer failures

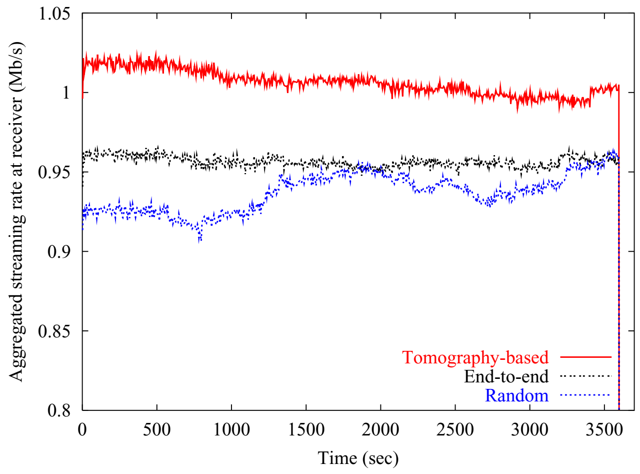

Fig. 9 Aggregated streaming rate at the receiver: with peer failures

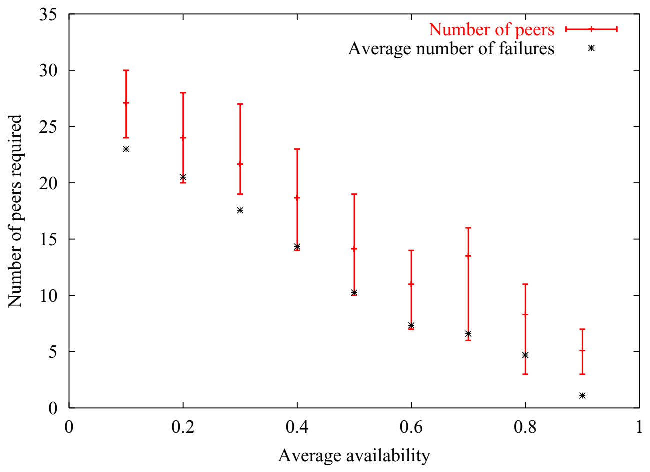

Fig. 10 Size of the candidate peers set required for different average peer availability values. The midpoint is the mean, the lower point is the minimum, and the top point is maximum number of peers required in the candidate set

7.5 Candidate set and active set sizes

In this section, we study two aspects of CollectCast. First, we assess the impact of peer availability on the size of the candidate peer set. The size of the candidate peer set is an important parameter because it allows us to configure the P2P lookup substrate to return an appropriate number of peers. If the size is too small, CollectCast may run out of peers during the streaming session because of peer failures. In this case, a new request is issued to the P2P lookup substrate to return more peers, which may cause long period of disruption. On the other hand, if the size is too large, the overhead incurred during the construction of the topology will be higher and the selection algorithm may take unnecessarily long time to determine the active set. Second, we estimate the average size of the active peer set during the streaming session. This indicates the average number of connections that a receiving peer may need to maintain concurrently.

7.6 Impact of peer availability on the size of the candidate set

We estimate the size of the candidate set for different values of peer average availability. We vary the average availability of peers from 0.1 to 0.9. A total of 25 failure trials are scheduled during each streaming session. If a failure trial is successful (i.e., a peer fails), a replacement peer (or peers) will be chosen. We run the simulation 10 times for each value of peer availability and count the total number of peers that are needed to complete the session. Figure 10 shows the impact of peer availability on the size of candidate set. The figure shows the average number of successful failure trials (out of 25) and the minimum, mean, and maximum number of peers required in the candidate set as the average availability grows from 0.1 to 0.9, over the 10 simulation runs. For example, for an average peer availability of 0.6, we need an average of 11 peers in the candidate set, and a maximum of 14 will guarantee that we will not run out of peers in the candidate set. Figure 10 shows that as the availability increases, the number of peers needed in the candidate set decreases.

7.7 Size of the active set

The receiving peer establishes concurrent connections with all peers in the active set. Each connection adds overhead on the receiver: more buffers are allocated and more control packets are sent. Using the same parameters as in Sect. 7.1, we conducted several experiments to estimate the average size of the active set. As shown in Fig. 11, we find that the average number of active peers is fairly small, less than four most of the time and it does not depend on the availability.

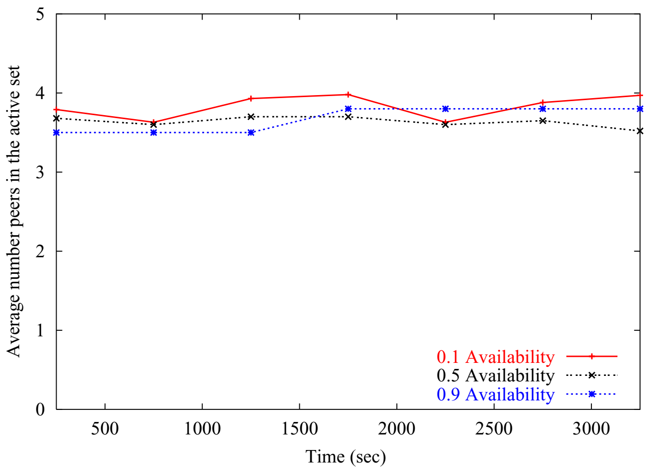

Fig. 11 Average number of senders in the active set under different average peer availability

Table 2 PlanetLab nodes used in our experiments

Node	Abbreviation	Location
planetlab3.millennium.berkeley.edu	ucb	West USA
planetlab1.ucsd.edu	ucsd	West USA
planlab1.cs.caltech.edu	caltech	West USA
planetlab-1.stanford.edu	stanford	West USA
planetlab1.cs.duke.edu	duke	East USA
planetlab1.lcs.mit.edu	mit	East USA
planetlab-1.cmcl.cs.cmu.edu	cmu	East USA
planetlab-01.bu.edu	bu	East USA
ricepl-1.cs.rice.edu	rice	South USA
planetlab1.cs.purdue.edu	purdue	Midwest USA
pads21.cs.nthu.edu.tw	taiwan	Taiwan, Asia
planetlab1.cs.unibo.it	bolonga	Italy, Europe
miranda.tkn.tu-berlin.de	berlin	Germany, Europe
planetlab-1.it.uu.se	sweden	Sweden, Europe
planetlab1.inria.fr	inria	France, Europe

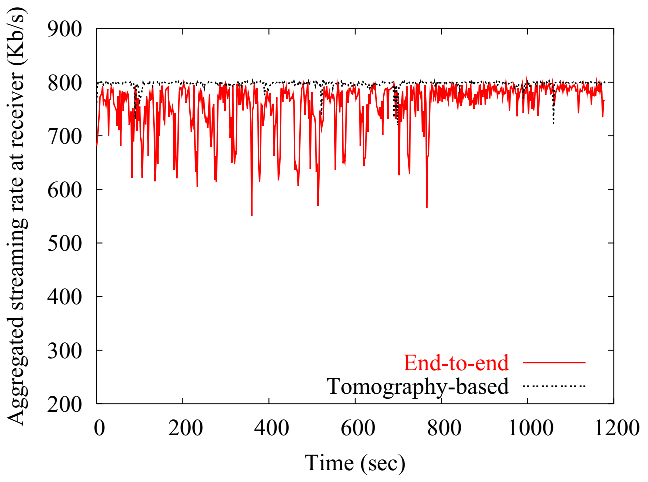

Fig. 12 Effect of peer selection methods on aggregated streaming rate

8 PlanetLab experiment results

To assess the performance of CollectCast in real environments, we have implemented a P2P media streaming system on top of CollectCast, called PROMISE. PROMISE has been tested in both local and wide area environments. In the implementation, we use Pastry as the P2P lookup substrate [13]. We have modified Pastry to support multiple peer lookup. For experiments in a wide area environment, we have installed PROMISE on 15 nodes of the PlanetLab testbed [22]. PlanetLab is composed of machines contributed from many academic and industrial research institutions. All machines run a customized version of Linux, and are centrally administered by the PlanetLab team. The PlanetLab nodes chosen for our experiments are distributed over different geographic locations. Table 2 lists all nodes, their locations, and their shortcut names used in this section.

We have conducted an extensive experimental study to assess the performance of PROMISE from several angles. In this section, we present two sets of results. The first set presents the packet-level performance, which considers the aggregated rate measured on the receiver side and how it changes over time. The second set addresses the frame-level performance, which focuses more on the perceived quality quantified in terms of the number of frames that either miss their deadlines or are lost. More results showing the impact of changing system parameters on the streaming quality, as well as how PROMISE handles peer failures and switchings can be found in [14].

8.1 Packet-level performance

In this set of experiments, we focus on the raw aggregated received rate measured by the receiver. The setup is as follows. The receiver is located at the UC Berkeley peer. The remaining 14 peers constitute the set of candidate peers. We construct and annotate the tree topology connecting the candidate peers with the receiving peer. We construct the topology using the `tracepath` tool, which is similar to the `traceroute` tool but does not require superuser privileges. We measure the available bandwidth using `pathload` [21]. After annotating the topology, we choose the active peer set using two sender selection methods: tomography-based and end-to-end. The streaming session lasts 20 min. The playback rate R_0 is 800 Kb/s. Dynamic peer switching and FEC encoding are turned off. We repeat the streaming session five times and obtain the average aggregated streaming rate. The results are shown in Fig. 12. The aggregated streaming rate from peers selected by the end-to-end method varies widely and sometimes drops below 600 Kb/s, whereas the aggregated streaming rate from peers selected by the tomography-based method is smooth and rarely drops below 750 Kb/s. The reason is that the end-to-end method selected two peers (one at Caltech and the other at UCSD) that share a tight network segment, which could not support the aggregated rate from them both. The tomography-based method avoided that segment and chose a better active set containing two peers, one at Rice and one at UCSD.

8.2 Frame-level performance

In this set of experiments, we study the movie streaming quality using CollectCast. We quantify the quality by the number of frames that either miss their playback deadlines or are lost. We differentiate among the two cases because a larger initial buffering time could mitigate the first case, while it does not affect the second one (unless if we employ a retransmission technique). Moreover, higher values for α (loss tolerance level) may recover lost frames but it has little effect on late frames. We also study the impact of initial buffering on the quality, and compare the buffer size required by the tomography-based and the end-to-end sender selection methods.

We use video traces of several movies encoded using MPEG-4. The traces were obtained from [31]. We use the verbose versions of the traces. Each row of the trace file has

Table 3 MPEG-4 movie traces used in PlanetLab experiments

Movie title	Average rate (Kb/s)	Peak rate (Kb/s)	Size (Mbyte)	Streaming rate (R_0 Kb/s)
Star Wars IV	287.21	1874.00	43.08	400.00
The Firm	364.72	2020.40	54.71	400.00
Aladdin Cartoon	402.90	2559.80	60.44	400.00
From Dusk Till Dawn	576.12	3106.00	86.42	800.00

four entries: frame number, frame type (I, P, or B), frame playout time, and frame length in bytes. The frame playout time is relative to the first frame playout time, which is set to zero. The movie titles and some statistics are listed in Table 3. We stream only the first 20 min of each movie, that is, we stream 30,000 frames of each movie because all movies have a frame rate of 25 frames/s. The setup of these experiments is similar to the setup of the previous set of experiments, except that the FEC encoding is enabled. We set $\alpha = 1.2$ and the segment size equals 1 s playback. We record the arrival time of each single packet. After the termination of the streaming session, we determine the number of frames that would have missed their deadlines for a specific initial buffering time. To decide whether a frame f misses its deadline, we compare two values: f_{deadline} and f_{avail}. If f_{deadline} is greater than f_{avail}, f misses its deadline. The f_{deadline} is the sum of the frame playout time (read from the trace file) and the initial buffering time. The f_{avail} is the time at which all packets constituting f are successfully reconstructed by the FEC decoder and are available in the buffer.

Figure 13 shows the results from four different movies: Star Wars IV, The Firm, Aladdin Cartoon, and From Dusk Till Dawn. For each movie, we repeat the session five times and plot the average. The first observation is that peers selected by the tomography-based method require much less initial buffering in *all* four cases. To ensure full quality, i.e., no frame misses its deadline, the tomography-based method requires, on the average, less than half of the initial buffering required by the end-to-end method. The second observation is that the total number of frames that miss their deadlines depend on the movie characteristics and the streaming rate R_0. For example, in Fig. 13a, the initial buffering needed to ensure full quality is fairly small (about 10 s) for the tomography-based method. Also, the number of frames that missed their deadlines is relatively small for buffering less than 10 s. This is because the average and peak rates of the

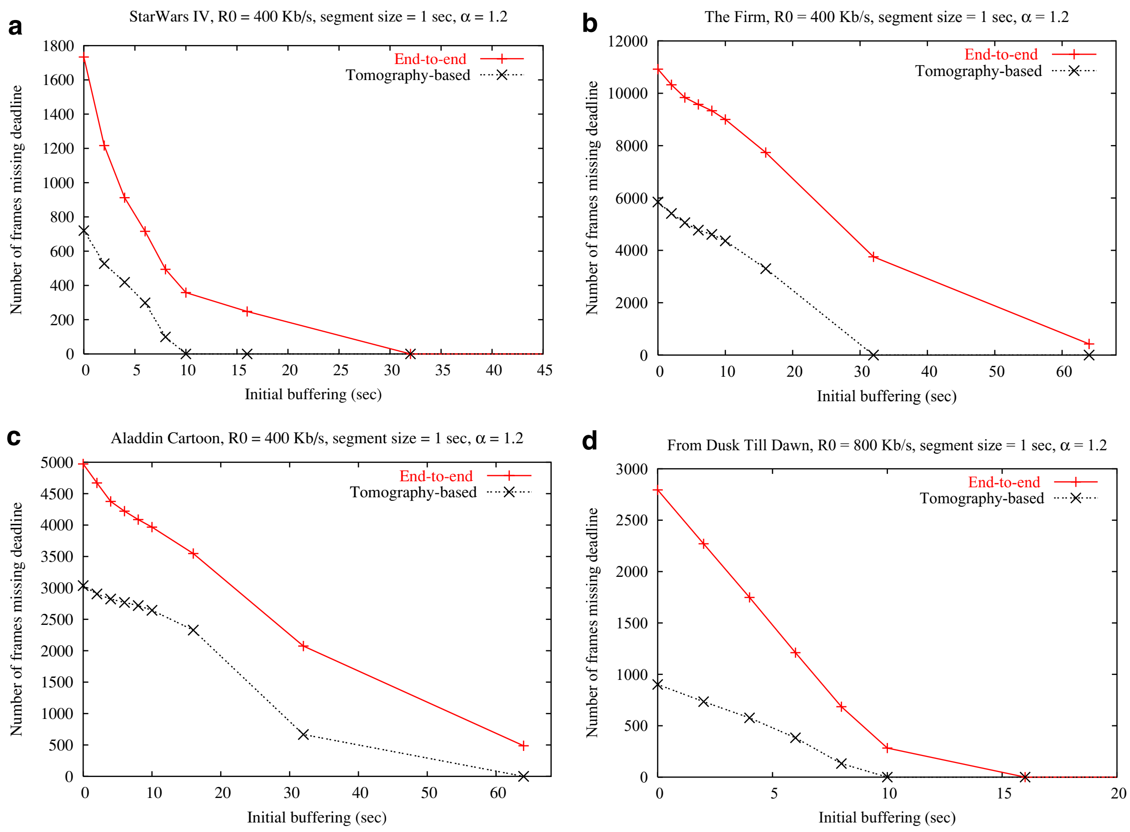

Fig. 13 Frame-level performance: initial buffering needed to ensure full quality. The tomography-based method requires much shorter initial buffering than the end-to-end method to ensure that all frames meet their deadlines. Traces from four different movies are used in the experiments

Star Wars IV movie are only 287.21 Kb/s and 1874.00 Kb/s, respectively, and we stream at $R_0 = 400$ Kb/s. In contrast, we need a larger initial buffering in the case of The Firm (Fig. 13b) and Aladdin Cartoon (Fig. 13c) because the average and peak rates are higher in these two cases. This implies that selecting the appropriate streaming rate for each movie has a direct impact on the playback quality.

9 Related work

In the past few years, the P2P paradigm has received tremendous attention from researchers. Two main categories of research can be identified: research on protocols and algorithms (such as searching and replication), and research on building P2P systems. The first category aims at building scalable and efficient P2P infrastructure (substrate), which could be used for systems in the second category. Lookup (or routing) protocols such as CAN [23], Chord [30], and Pastry [26] guarantee locating the requested object within a logarithmic number of steps, if the object exists in the system. However, network locality has not been amply exploited (except in case of Pastry). Examples of P2P systems include CFS [11] on top of Chord [30], and PAST [27] on top of Pastry [26]. Another example is Pixie [25]: a P2P content exchange architecture. Pixie aggregates requests from multiple peers and multicasts content to requesting peers. These systems do not target media streaming. Therefore, unlike CollectCast, they do not consider real-time and sending rate requirements for P2P data transport.

Application level multicast (ALM) is proposed to overcome the limited deployment of IP multicast. Each ALM-based system has its own protocol for building and maintaining the multicast tree. For example, both NICE [1] and Zigzag [32] adopt hierarchical distribution trees and therefore scale to a large number of peers. Narada [9], on the other hand, targets small scale multisender multireceiver applications. Narada maintains and optimizes a *mesh* that interconnects peers. The optimized mesh yields good performance but it imposes maintenance overhead. SpreadIt [12] constructs a distribution tree rooted at the sender for a live media streaming session. A new receiver joins by traversing the tree starting at the root till it reaches a node with sufficient remaining capacity. CoopNet [20] supports both live and on-demand streaming. It employs multidescription coding and constructs multiple distribution trees (one tree for each description) spanning all participants. SplitStream [7] provides a cooperative infrastructure that can be used to distribute large files (e.g., software updates) as well as streaming media. SplitStream is built on top of Scribe [8], a scalable publish-subscribe system that employs Pastry [26] as the lookup substrate. The content in SplitStream is divided into several *stripes*, each distributed by a separate tree. Different from these systems, CollectCast is not intended for multicast. As a complementary P2P service, CollectCast is proposed for media streaming from multiple senders to one receiver.

Many P2P data sharing and distribution systems implicitly assume that a sending peer is capable of supporting one or more receiving peers. However, it has been shown that peers are heterogeneous in their capability and/or willingness to contribute resources to other peers [28]. Few systems before CollectCast have considered the problem of selecting multiple supplying peers for a receiver, based on peer heterogeneity as well as network tomography information.

The distributed video streaming framework [17, 18] shows the feasibility and benefits of streaming from multiple servers to a single receiver. The receiver uses a rate allocation algorithm to determine the sending rate for each server to minimize the total packet loss. The rate allocation is based on estimates of the end-to-end loss rate and available bandwidth between the receiver and each server. However, the framework is not designed for P2P environments. Therefore, it does not address the selection and dynamic switching of senders.

Finally, Rodrigues and Biersack [24] show that parallel download of a large file from multiple replicated servers achieves significantly shorter download time. The subsets of a file supplied by each server are dynamically adjusted based on network conditions and server load. However, their work targets bulk file transfer, not real-time media streaming. Moreover, it does not consider the sender selection problem and it does not leverage network tomography techniques.

10 Conclusion and future work

This paper presents CollectCast, a novel P2P media streaming service that provides high-quality streaming in cooperative P2P environments. CollectCast optimizes the quality of each streaming session by carefully selecting senders and dynamically adapting to peer failures and network fluctuations. To select the best senders, CollectCast constructs the network topology connecting all candidate senders and the receiver. It then employs network tomography techniques to infer the loss rate and available bandwidth on each segment of the topology. Using this information, CollectCast maps the selection problem to a constrained optimization problem with the objective of maximizing the aggregated rate at the receiver. The constraints of the problem are defined by the lower and upper bounds of the receiver's inbound bandwidth.

We have evaluated CollectCast using packet-level simulations. Our simulations demonstrate that CollectCast achieves significant gain in the streaming quality in terms of streaming rate and packet loss rate even in the presence of peer failures and network fluctuations. In addition, we have implemented a P2P media streaming system called PROMISE on top of CollectCast and evaluated the system in PlanetLab. The performance results obtained from streaming several MPEG-4 video files show that: (1) the aggregated received rate is much smoother in streaming sessions that employ CollectCast than those that do not, and (2) the number of frames that miss their deadlines is reduced by about 50%. Our experimental and simulation results confirm that streming from multiple failure-prone peers to a receiver in a wide-area P2P environment is indeed feasible.

CollectCast can be extended beyond physical network characteristics and streaming applications. For example, CollectCast may take a peer's social properties such as credits and trustworthiness into consideration, which will lead to a logical topology formed by a set of candidate suppliers and a requester with its links labeled with trust-related metrics. This will enable security-sensitive applications to choose the best set of peers that will supply the most trusted data or service.

Acknowledgements We would like to thank the Associate Editor and the anonymous reviewers for their valuable comments. We also thank Jen-Yeu Chen of Purdue University for his helpful suggestions. This research is sponsored in part by the National Science Foundation grants ANI-0219110 and IIS-0209059 and by a gift from Microsoft Corporation.

References

1. Banerjee, S., Bhattacharjee, B., Kommareddy, C., Varghese, G.: Scalable application layer multicast. In: Proceedings of ACM SIGCOMM'02, pp. 205–220. Pittsburgh, PA (2002)
2. Bawa, M., Deshpande, H., Garcia-Molina, H.: Transience of peers and streaming media. First Workshop on Hot Topics in Networks (HotNets 2002)(2002)
3. Bestavros, A., Byers, J., Harfoush, K.: Inference and labeling of metric-induced network topologies. In: Proceedings of IEEE INFOCOM'02. New York (2002)
4. Byers, B., Luby, M., Mitzenmacher, M., Rege, A.: A digital fountain approach to reliable distribution of bulk data. In: Proceedings of ACM SIGCOMM'98, pp. 56–67. Vancouver, British Columbia (1998)
5. Calvert, K., Doar, M., Zegura, E.: Modeling Internet topology. In: IEEE Commun. Mag., **35**, 160–163 (1997)
6. Calvert, K., Griffioen, J., Mullins, B., Sehgal, A., Wen, S.: Concast: Design and implementation of an active network service. IEEE J. Sel. Area Commun. **19**(3), 426–437 (2001)
7. Castro, M., Druschel, A., Kermarrec, P., Nandi, A., Rowstron, A., Singh, A.: SplitStream: High-bandwidth content distribution in a cooperative environment. In: Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03). Berkeley, CA (2003)
8. Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE J. Sel. Areas Commun. (JSAC) **20**(8), 1489–1499 (2002)
9. Chu, Y., Rao, S., Seshan, S., Zhang, H.: A case for end system multicast. IEEE J. Sel. Areas Commun. (JSAC) **20**(8), 1456–1471 (2002)
10. Coates, M., Hero, R., Nowak, A., Yu, B.: Internet tomography. IEEE Signal Process. Mag. **19**(3) (2002)
11. Dabek, F., Kaashoek, M., Karger, D., Morris, D., Stoica, I.: Wide-area cooperative storage with CFS. In: Proceedings of ACM SOSP (2001)
12. Deshpande, H., Bawa, M., Garcia-Molina, H.: Streaming live media over peer-to-peer network. Technical report, Stanford University (2001)
13. Free pastry home page. http://www.cs.rice.edu/CS/Systems/Pastry
14. Hefeeda, M.: A Framework for Cost-effective Peer-to-Peer Content Distribution. PhD thesis, Department of Computing Sciences, Purdue University. West Lafayette (2004)
15. Jain, M., Dovrolis, C.: End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In: Proceedings of ACM SIGCOMM'02, pp. 295–308. Pittsburgh, PA (2002)
16. Markovski, V., Xue, F., Trajkovic, L.: Simulation and analysis of packet loss in user datagram protocol transfers. J. Supercomput. **20**(2), 175–196 (2001)
17. Nguyen, T., Zakhor, A.: Distributed video streaming over Internet. In: Proceedings of Multimedia Computing and Networking (MMCN'02). San Jose, CA (2002)
18. Nguyen, T., Zakhor, A.: Distributed video streaming with forward error correction. In: Proceedings of the Int'l Packetvideo Workshop (PV'02). Pittsburgh, PA (2002)
19. Padmanabhan, V., Qiu, L., Wang, H.: Server-based inference of Internet link lossiness. In: Proceedings of IEEE INFOCOM'03. San Francisco, CA (2003)
20. Padmanabhan, V., Wang, H., Chou, P., Sripanidkulchai, K.: Distributing streaming media content using cooperative networking. In: Proceedings of NOSSDAV'02. Miami Beach, FL (2002)
21. Pathload home page. http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/pathload.html/
22. Planetlab home page. http://www.planet-lab.org/
23. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of ACM SIGCOMM'01. San Diego, CA (2001)
24. Rodriguez, P., Biersack, E.: Dynamic parallel access to replicated content in the Internet. IEEE/ACM Transactions on Networking **10**(4), 455–465 (2002)
25. Rollins, S., Almeroth, K.: Pixie: A jukebox architecture to support efficient peer content exchange. In: Proceedings of ACM Multimedia. Juan Les Pins, France (2002)
26. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proceedings of 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). Heidelberg, Germany (2001)
27. Rowstron, A., Druschel, P.: Storage management in past, a large-scale, persistent peer-to-peer storage utility. In: Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01). Chateau Lake Louise, Banff, Canada (2001)
28. Saroiu, S., Gummadi, P., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking (MMCN'02). San Jose, CA (2002)
29. Stemm, M., Seshan, S., Katz, R.: A network measurement architecture for adaptive applications. In: Proceedings of INFOCOM'00. Tel-Aviv, Israel (2000)
30. Stoica, I., Morris, R., Kaashoek, M., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proceedings of ACM SIGCOMM'01. San Diego, CA (2001)
31. Traces, MPEG-4 movie. http://www-tkn.ee.tu-berlin.de/research/trace/ltvt.html
32. Tran, D., Hua, K., Do, T.: Zigzag: An efficient peer-to-peer scheme for media streaming. In: Proceedings of IEEE INFOCOM'03. San Francisco, CA (2003)
33. Xu, D., Hefeeda, M., Hambrusch, S., Bhargava, B.: On peer-to-peer media streaming. In: Proceedings of IEEE ICDCS'02. Vienna, Austria (2002)
34. Yajnik, M., Moon, S., Kurose, J., Towsley, D.: Measurement and modeling of the temporal dependence in packet loss. In: Proceedings of IEEE INFOCOM'99, pp. 345–352, York, NY (1999)
35. Zegura, E., Ammar, M., Fei, Z., Bhattacharjee, S.: Application-layer anycasting: A server selection architecture and use in a replicated web service. IEEE/ACM Trans. Netw. **8**(4) (2000)
36. Zhang, Y., Duffield, N., Paxon, V., Shenker, S.: On the constancy of Internet path properties. In: Proceedings of ACM SIGCOMM Internet Measurement Workshop. San Francisco, CA (2001)