Measurement of Commercial Peer-To-Peer Live Video Streaming

Shahzad Ali, Anket Mathur and Hui Zhang
School of Computer Science
Carnegie Mellon University

***Abstract*— Peer-to-peer technologies have proved to be effective for various bandwidth intensive, large scale applications such as file-transfer. For many years, there has been tremendous interest in academic environments for live video streaming as another application of P2P. Recently, a number of new commercial scale video streaming systems have cropped up. These systems differ from others in the type of content that they provide and attract a large number of users from across the globe. These are proprietary systems and very little is known about their architecture and behavior. This study is one of the first of its kind to analyze the performance and characteristics of P2P live streaming applications. In particular, we analyze PPLive and SOPCast, two of the most popular systems in this class. In this paper, we (1) present a framework in which to analyze these P2P applications from a single observable point, (2) analyze control traffic to present a probable operation model and (3) present analysis of resource usage, locality and stability of data distribution. We conclude that P2P live streaming has an even greater impact on network bandwidth utilization and control than P2P file transfer applications.**

I. Introduction

Peer-to-Peer (P2P) systems have been employed in a wide variety of distributed applications such as File Sharing and VOIP. The ubiquity, resilience and scalability provided by P2P systems make them ideal for large scale distributed applications. Recent studies have shown that the overall P2P traffic on the Internet has been constantly increasing [13], [14], [26].

Over the past few years, there has been a great deal of academic interest in the area of P2P video streaming applications. A lot of interesting research has been done and some encouraging results have been reported [1], [3]–[5], [7], [11], [12], [16]–[18], [21], [25], [28]. Most of these systems are available as prototype implementations and have been tested in limited research environments. The deployment experience of these applications is very limited with the only reported deployment scenario described in [6], [29].

Recently, a new breed of P2P video streaming applications has emerged [19], [20], [24], [27]. These are commercial grade systems that are broadcasting compelling content, capable of attracting large audiences. Due to the type of content they provide, some of these systems have gained tremendous popularity. Still, there is very little to no understanding about the protocols and behaviors of these applications. The systems, due to their commercial nature, have not been specified openly nor are available as open source.

In this paper, we will evaluate the performance of two of the most popular proprietary video streaming systems available, PPLive and SOPCast. Both these systems are based in Asia but have a significant user base in North America as well. These systems broadcast popular Asian and American content and attract a large number of users.

While studying these systems, we were faced with several challenges that stem from the commercial nature of these systems. These are *proprietary* systems, which means that their source code is not available and there is no specification of the protocols available. This forced us to conduct most of our studies treating the systems as black boxes. Another challenge we faced was due to the fact that there is no documentation or API that would enable the development of test scripts. This meant that we were unable to have a large number of vantage points in the system at any given point in time. Since the applications had to be manually started from the Graphical User Interface (GUI) and the channels manually selected, our measurement has to be performed in locations that we have physical access to.

To evaluate these systems, we had to setup experiments to give us insight into the operation of these systems from a small number of vantage points. We collected packet traces from runs of the systems under different environments and analyzed the data. First, we separated the control traffic from the data traffic in the collected traces. This allowed us to analyze the control traffic separately and reverse engineer the protocols used to get an understanding of how these systems work. We present an overview of how these systems operate based on our analysis of the control traffic. Next, we define a generic framework that can be used to evaluate the data distribution performance of such systems. For that purpose, we define metrics to highlight the key characteristics of the distribution plane. Finally, we evaluate the two systems using the framework formulated and present results.

II. Protocol Analysis

The systems we analyzed are proprietary and very little is known about the underlying protocols used in them. The systems use peer-to-peer technologies to support thousands of simultaneous clients. Before we proceed with the data analysis, we will outline the underlying protocols employed. We conducted simple experiments on a single host joining a system and then tuning into a channel, and collected packet traces for these cases. We were then able to separate the control traffic from the actual data traffic from the packet traces. We inspected the control packets visually to classify them into the different protocol messages. We were able to decode a number of these protocol messages and get a better understanding of the protocols.

Surprisingly, the two systems we analyzed showed remarkable similarity as far as their control protocols are concerned. We will present a general overview of the protocols used in both these systems and compare it with the system described in [29]. We will also present some specific examples from one of the protocols that we analyzed.

A. Experiment Setup

To analyze these applications, we had to collect packet traces from these systems. To make sense of the large amount of traffic data that was collected, we defined certain terms, such as *parents* and *children*. We also came up with concepts such as *distance* and metrics such as *cost of download* to better understand the behaviors of these applications. In this section, we will describe how we collected data, and define the terms and concepts that will be used in later sections.

B. Data Collection

Our test machines were all Intel Pentium 4 computers running Windows XP operation systems. The choice of machines was based on the physical access restriction. We collected packet traces for each of the experiments using Ethereal [10]. These traces include the 128 bytes of the packet including data to allow us to analyze the control packets. The types of machines we collected the traces on are listed below.

Name	**ISP**	**Bandwidth**	**NAT**
cmu	CMU - Pittsburgh	LAN	none
cmun	CMU - Pittsburgh	LAN	full-cone
pittc	Comcast - Pittsburgh	cable	symmetric
pittd	Verizon - Pittsburgh	DSL	none
chic	Ameritech - Chicago	cable	full-cone
dull	Comcast - Dulles	cable	full-cone

The length of the runs varies for the different experiments we performed. Most of the experiments were run for 2-3 hour durations. However, some of the experiments to gather control and connectivity information were run for shorter periods of time. The analysis presented later is done on snapshots of data collected in these traces.

We collected data from these sources in Oct 2005 and Dec 2005.

C. Protocol overview

The systems that we analyzed seem to be following a very similar approach in their operation. Some of the key components of these systems are presented below.

1) Software Update: Both systems analyzed have automatic software version checking and software update capabilities. The client upon start-up contacts a webserver to get the latest version of the software. If a newer version is available, the system can be directed to download that version and install it automatically. We know very little about the software version and update abilities of the system in [29].

2) Channel Lists: The system requires to get a list of channels available for the client to tune in to. There seem to be two modes by which a client can get a list of available channels. A webserver listing all the channels available is provided. By clicking on the link for the channel the user is interested in, the client application can start. The second method is by starting the application which in turn gets the channel list. The user can then click on a channel to tune into it.

3) Bootstrap: In [29], the source of the video stream is described as a well-known location and all clients initially go through the source to join any channel. In the applications we analyzed, a webserver is used to get initialization information. This communication with a central webserver is done using HTTP using ASCII strings. The webserver or a component on the webserver maintains bootstrapping information for each channel that is passed along to the clients upon joining the channel.

4) Maintaining Peers: Once the channel selection is sent to the central server, a list of possible neighbors/peers is sent to the client. The list is usually on the order of 10's of hosts and the client attempts to connect with these hosts. During the course of the session, the client keeps getting regular updates about more neighbors from its existing neighbors. The client also re-broadcasts its current neighbor list to its neighbors. This mechanism is very similar to the gossip protocol used in [29] to keep an updated list of peers.

5) Requesting Data: As the client joins a session, it exchanges control messages with its peers using a gossip-like protocol. Although we were not able to completely decode the messages exchanged, we know that specific information about each neighbor is exchanged, referred to as the neighbor record. The record includes connectivity information and possibly a buffer map similar to [29]. The client then requests segments of data from some of its peers. The size of the segments as well as the scheduling algorithm used to request them is not clear from our analysis.

D. Separating Data and Control Traffic

The first step in the analysis of the data plane involves separation of control and data traffic. Control traffic can create noise in the analysis of the data plane and thus needs to be identified and eliminated. We used packet size as a simple mechanism to separate control and data traffic.

The graph in Figure 1 shows the packet size distribution for PPLive and SOPCast. In both cases, about 40% of the packets are 40 byte packets. These packets are pure acknowledgments of data packets sent and received. Another 50% of packets are greater than 1 KBytes, with the largest size packets being at 1400 bytes, which is approximately the Maximum Transmission Unit (MTU) for IP packets over Ethernet networks. There are about 10-20% packets that are in between these two ranges and we conclude that these are the control packets. The total amount of bandwidth used by the control packets is thus on the order of 5% of the total bandwidth.

Given the low frequency of the control packets as shown by their small numbers, we concluded that any pair of hosts that exchange more than 4KBps worth of data have data flow between them. We used this metric to define parent child relationships as described in Section III-D.

E. Specific Protocol Description

In this section, we will present the results of the protocol analysis specifically for PPLive. As mentioned earlier, the results for SOPCast are very similar and any difference will be mentioned.

1) Software Update: The application attempts to check its software version with a central server. This is done through an HTTP request sent to the centralized server. If there are a number of clients that startup at the same time, it is quite possible that the load on this centralized web server can become extremely large which might pose a concern for scalability.

The application on startup, opens a TCP connection on port 80 to a central server, update.pplive.com and sends an HTTP request **GET /update/pplive/update.inf**. We believe that if that file exists, PPLive assumes there is an update available. If it gets a **404 Not Found** response, it continues the startup process with the current software version.

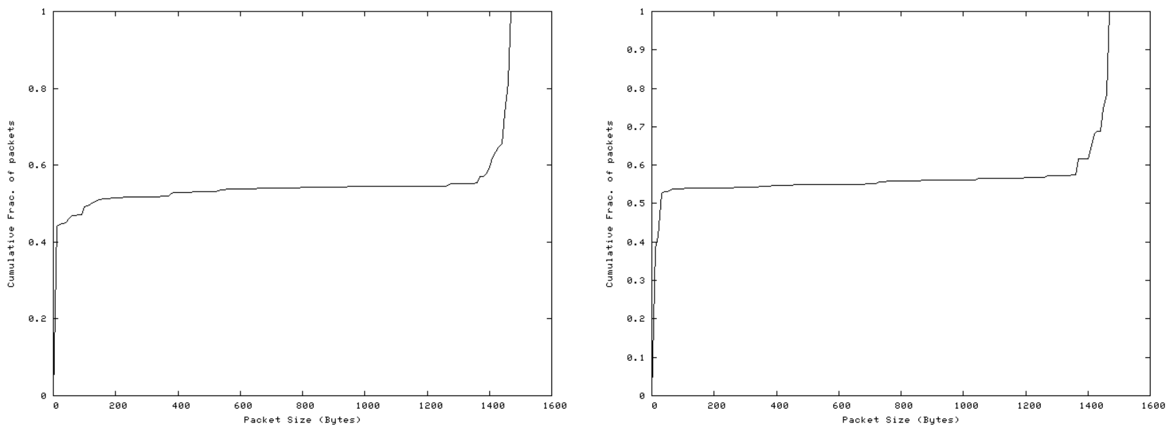
2) Channel list: The application next gets a list of channels from a centralized server. With the list of channels, there is information about how to connect to these channels, including Identifiers, trackers, etc. This could be another flash crowd point in the system if a large number of clients join at the same time.

The application opens a second TCP connection on port 80 to a central server, http://list.pplive.com/ and sends an HTTP request **GET /web/xml/all.xml**. This is an xml file containing the list of available channels. This list is retrieved once at the initial startup time and is written in the client hard disk as **channel.xml**. The format of the file is shown below.

```
<root>
  <channel>
    <ChannelGuid>{7D0F9A10-F9C3-4AAE-A2D6-0E54639BC715}</ChannelGuid>
    <BaseChannelGuid>{ceb266b8-11d4-4ab3-83bc-c4b502a3cb74}</BaseChannelGuid>
    <ChannelName>CCTV-5</ChannelName>
    <ChannelNameEn>CCTV-5</ChannelNameEn>
    <Author>..........</Author>
    <AuthorUrl>http://www.pplive.com</AuthorUrl>
    <PlayLink>
       pplive://211.162.0.46:8000/tracker?
       source={7D0F9A10-F9C3-4AAE-A2D6-0E54639BC715}&amp;type=udp&amp;
       chnname=CCTV-5&amp;engname=CCTV-5&amp;tracker=udpt://211.162.0.46:8000;
       tcpt://211.162.0.46:8000;
    </PlayLink>
    <PeerCount>0</PeerCount>
    <Bitrate>0</Bitrate>
    <PreViewUrl />
    <Quatity>100</Quatity>
    <Catalog>2</Catalog>
    <ProgramList1 />
    <ProgramList2 />
  </channel>
  <channel>
    ...
  </channel>
    ...
</root>
```

The channels are identified by a 16 byte identifier as well as a name. The `playlink` object points to a host with a tracker running. The tracker is the initial point where all clients connecting to the channel go to get a list of neighbors. The ability of the tracker to communicate through UDP and TCP is stated in this object along with the port number for the tracker.

3) Bootstrap Mechanism: The application already has information about the tracker for each channel from the channel list it downloaded earlier. As soon as the



(a) PPLive (b) SOPCast

Fig. 1. CDF of Packet Sizes

user selects a channel, the client sends a UDP (or TCP depending on the tracker information it already has) request to the tracker for that channel.

The reply from the tracker is a message that contains the list of initial hosts that need to be contacted. The list returned contains at most 50 hosts depending on the size of the group.

```
02 00 00 02 99 58 98 00 2d 5a 98 00 02 00 00 00 32      <- header
3b 40 c6 fa a4 0f 48 1f 00 00 00 00 18 00 07 00 35 00   <- Host1
da be 31 ad a4 0f 48 1f 00 00 00 00 00 00 34 07 01 00   <- Host2
...
...
da bf 29 6c a4 0f 48 1f 00 00 00 00 b4 ef 08 00 34 00   <- Host50
```

The header contains the number of hosts that are in the message which is 50 (0x32). Each host is represented as a structure, with the IP address 59.64.198.250 being the first four bytes (for the first host 0x3b40c6fa), the UDP port 4004 being the next two bytes (0xa40f) and the TCP port 8008 being the next two bytes (0x481f).

4) Tuning into the channel: Once the initial list of hosts is retrieved, the system tries to connect to some of these hosts to get data. The system also gets updates from these hosts and keeps increasing the list of hosts that it knows about. We observed that the initial list contained some addresses that are private. This can be a concern if most hosts that are part of a group connect through a NAT.

In our trace, the system sent the same UDP message to 39 of the 50 hosts in its initial list. This message contains the Channel ID of the channel the user has selected.

5 out of these 39 hosts the message is sent to were private addresses and were thus not able to reply. This is probably a bug in the software. The private addresses can easily be pruned by the system so that the clients don't have to try connecting to them. The system received UDP replies from 15 of the 34 remaining hosts. All the messages received are of different sizes but they all have the same format.

```
02 00 00 02 10 9a                                        <- header
0f 7d c3 f9 ae 4a a2 d6 0e 54 63 9b c7 15                <- ChannelID
0c 30 e9 6f 4d 8f 9f 45 be dd 42 db 96 9e 11 e7
84 47 6e 01 03                                           <- number of hosts
8c 6d 2a d4 a4 0f 48 1f 00 00 00 00 00 04 00 00 3c 00   <- Host1
...
...
cb ba ad 88 a4 0f 48 1f 00 00 00 00 00 04 0a 04 2e 00   <- Host3
```

The messages contain host information similar to the information reported from the tracker. This means that other hosts are reporting back their neighbors. This is typical of a gossip protocol.

The system then tries to open TCP connections to all 39 hosts (5 of which are private addresses) that it had earlier sent the UDP messages. The hosts that reply to the TCP connection setup are used to exchange the same information that was earlier exchanged over UDP. Thus the communication mechanism is moved from UDP to TCP. From that point onwards, there is no UDP communication with the host.

III. Methodology

To analyze the data collected, we defined key objects and relationships between them. We know that any run of our experiment yielded hundreds of host addresses, but some were identified as key hosts. We then defined metrics to compare the relationships between different runs of our experiments. We will present some of the definitions used to classify objects and relationships and highlight key metrics of interest.

A. Flow

We define a flow to be an exchange of TCP or UDP packets between the monitored host and another host. A flow between the monitored host A[1] and another host X is represented as a four-tuple $\{IP_A, P_{A_1}, IP_X, P_{X_1}\}$, where P_{A_1} is the TCP port on host A and P_{X_1} is a TCP port on host X. The flow has directionality such as the two flows $\{IP_A, P_{A_1}, IP_X, P_{X_1}\}$ and $\{IP_X, P_{X_1}, IP_A, P_{A_1}\}$ are treated as two separate flows.

B. Rate of a flow

The rate of a flow is the amount of data in bytes, that the flow is comprised of for a particular duration of time. Using the notation for a flow, assume $F(A_1, X_1) = \{IP_A, P_{A_1}, IP_X, P_{X_1}\}$ is a flow between hosts A and X. The size of a packet that is part of this flow can be represented as $P_l(A_1, X_1, t)$ where t is the timestamp at which the packet is recorded. The rate of the flow at time $t1$ can then be represented as

$$R(A_1, X_1, t1) = \frac{\sum_{t=t1}^{t1+\Delta} P_l(A_1, X_1, t)}{\Delta}$$

In our results we use a Δ of 20 seconds, so that bandwidth is calculated on non-overlapping intervals of 20 second each.

C. Duration of a flow

The duration of the flow is defined as the time between the first packet of the flow and the last packet seen for the flow. Note that this includes cases where a flow stops for a period of time and starts again.

D. Parent and Child relationships

This term defines a key relationship between observed hosts. The rate of all flows between A and X can be represented as a summation

$$R(A, X, t1) = \sum_{\forall A_1} \sum_{\forall X_1} R(A_1, X_1, t1)$$

An observed host A is classified as a **parent** of host X at time $t1$ if

$$R(A, X, t1) > R_p$$

where R_p is 4KBps in our experiments. We choose this limit as hosts that send less than that amount of data are likely to be exchanging control information rather than data. Similarly, an observed host B is a **child** of host X at time $t1$ if

$$R(X, B, t1) > R_c$$

where R_c is 4KBps in our experiments.

[1]Any notion of a host in this thesis, refers to the IP address of the host. Thus A will be used to describe the host A and the IP address of A IP$_A$.

E. Distance

We also analyzed the locality of the resulting distribution tree by using estimates of distances between hosts. We used a database of subnets and their longitude and latitude to calculate the cartesian distance between a pair of hosts. The distances were calculated by performing a longest prefix match of the IP addresses to subnets in the database. In cases where the prefixes did not match, we did not use that distance value in our calculations. From all our results, we observe that 80% of the hosts match the geographical database.

Once the latitude and longitude information is available for each host, we compute the cartesian distance between the two coordinates. If host A has co-ordinates (x_1, y_1) and host X has co-ordinates (x_2, y_2), then the distance between A and X in miles can be represented as

$$D(A, X) = k * \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

where k is the constant for the distance of 1 degree and is equal to approximately 69.2 miles.

F. Cost

We define a cost metric that will be used to analyze the distance property mentioned earlier. The cost metric is associated with download and is measured in miles per byte. We analyze how efficient the downloads are for the different cases in terms of proximity of the parents. To calculate cost, we use a simple weighted average of the distance between the parents and the measured host.

Let B_p be the total number of bytes received from a parent p, and D_p be the distance between the host and its parent p. The cost of download, C for the host can be represented as

$$C = \frac{\sum_{\forall p}(D_p \times B_p)}{\sum_{\forall p} B_p}$$

A similar metric can be used to describe the cost of data sent to children.

G. Stability

As most distribution strategies end up making a tree for distributing a unit of data, we need to characterize the stability of the resulting distribution tree. We define a stability of a distribution structure in terms of the frequency of changing the parents for a host. We have two slightly different notions for change in this regard.

We define a period Δ and look at the parents at a start time of t_0. We then compare that with the parents we observe at $t_1 = t_0 + \Delta$. Let the set of parents at any time t_n be equal to P_n. Let the cardinality of P_n

be represented as $|P_n|$. One definition of stability for a host at time t_n can be

$$S_n = 1 - \frac{|P_{n-1} \bigcap P_n|}{|P_n|}$$

This definition of stability looks at the rate of change of parents. As $S_n \to 0$, the structure becomes stable.

Another metric for stability can be defined to take into account the differences between the set of parents. Let the set A_n be equal to $P_n \setminus P_{n-1} = \{x \in P_n | x \notin P_{n-1}\}$ represent the elements that are in P_n but not in P_{n-1}. This refers to the parents that have been newly acquired. Similarly, the set L_n be equal to $P_{n-1} \setminus P_n = \{x \in P_{n-1} | x \notin P_n\}$ represent the parents that have been left. Then stability can be represented as

$$X_n = 1 - \frac{|P_n|}{|P_n| + |A_n| + |L_n|}$$

We used both these definitions in our analysis.

IV. Data Plane

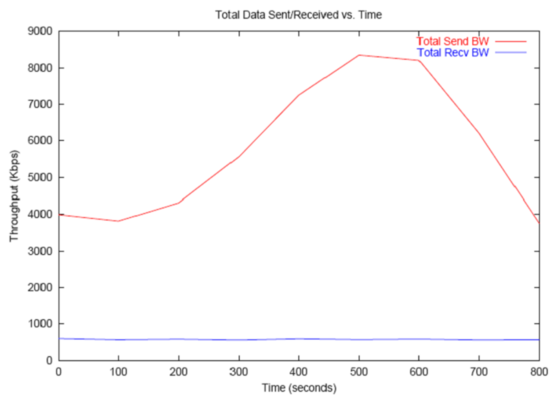
In this section, we present the data plane analysis of the application. Other applications analyzed exhibited similar data plane structures.

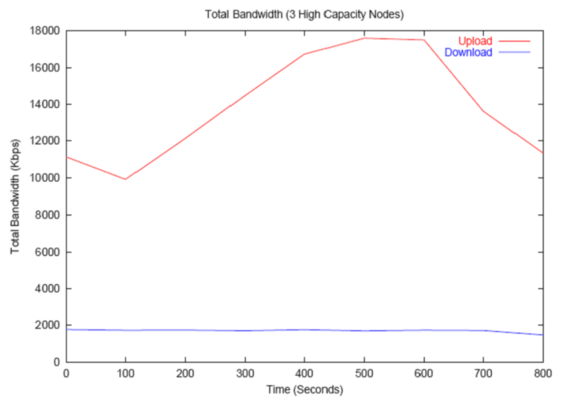
A. Network Resource Usage

The resource in terms of network bandwidth used is an important metric in analysing these applications. Most network environments are sensitive to network bandwidth utilization and administrators want some level of control over the bandwidth utilized by each member of the organization. Unlimited download or upload usage is often prohibited in controlled Internet environments. This is the major reason that Bit-Torrent has the concept of *fairness* where the amount of data uploaded from a host is propotional to the amount of data downloaded. This reduces the risk of freeloaders using up most of the network upload capacity without paying for it. In [15], the authors claim that ISP's need to be included into the model so that the data exchanged between the ISP's can be controlled. We analysed the network bandwidth utilization for these applications in various environments.

The two scenarios of interest for this analysis are a high bandwidth and a low bandwidth client. In both these types of clients, there was no NAT present. However, a high bandwidth node with a NAT showed results very similar to a low bandwidth node. As concluded earlier, NAT handling is inadequate in these application and even a high capacity node is considered *bad* from the point of view of the application if it has a NAT attached to it.

Figure 2 shows the graphs for sent and received bandwidth for *cmu*, *pittd* and *cmun* runs for App-B.





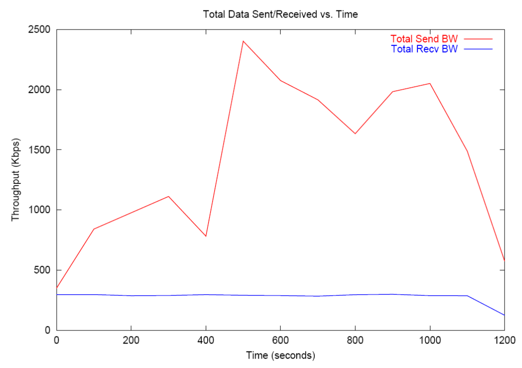
(a)cmu
(b)3 instances of cmu

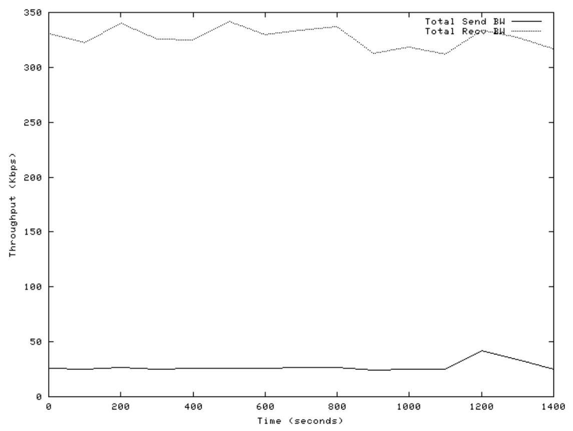
Fig. 2. Send and Receive rates for high capacity nodes

For Figure 2(a), the bandwidth sent from this node to other nodes is in the range of 5-8 Mbps. The bandwidth received by this node is around 500 Kbps which is approximately the rate of the video stream being played. One thing that is obvious from this graph is that there is no fairness, as the ratio of bandwidth sent to bandwidth received is 16:1. It seems as if there is no policy control on how much bandwidth can be uploaded from a particular host. To verify this, we conducted another experiement in which we ran multiple[2] *cmu* clients simultaneously to see if the total bandwidth uploaded from CMU is controlled. The results, as seen in Figure 2(b), indicate that no such policy bound exists and the bandwidth uploaded from CMU is as high at 18 Mbps.

Similar graphs were plotted for the *cmu* run of App-A. Figure 3(a) shows that the ratio between bandwidth sent and bandwidth received is 8:1. This ratio is The results look very similar to that of a low capacity node.

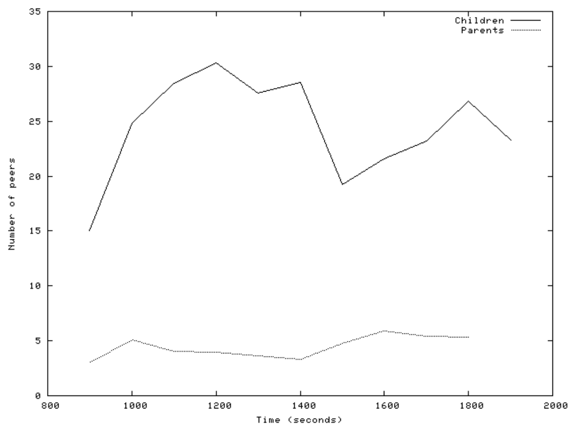
[2]In this case, we ran 3 separate clients simultaneously



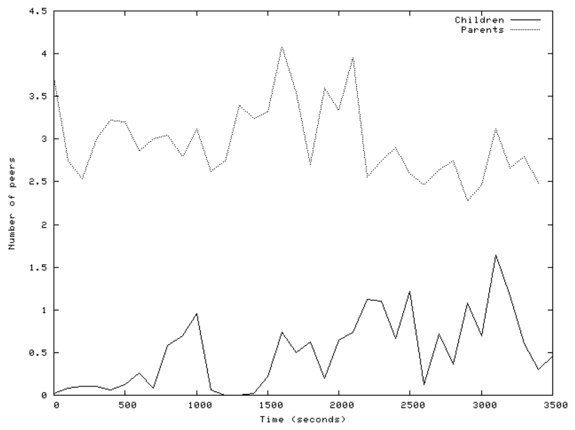


(a)cmu
(b)pittc

Fig. 3. Send and Receive rates for App-A



(a) cmu



(b) dull

Fig. 4. Number of Peers

This points to the very rudimentary handling of NAT's in this application. If the node is behind a NAT, its upload capacity is not used.

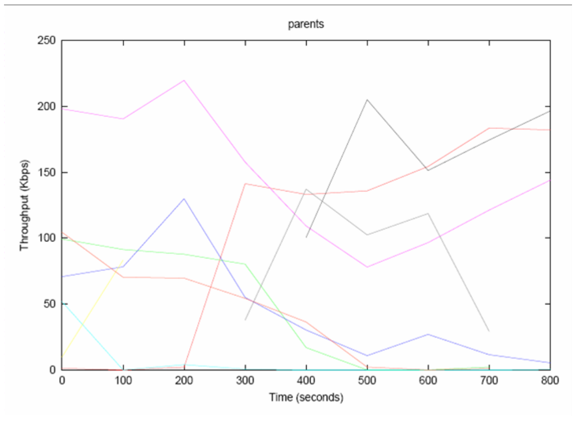
Another measure of the resource usage is the number of children that are supported by a node as a function of time. This is shown in Figure 4(a) for a high capacity and Figure 4(b) for a low capacity node. The number of parents[3] in both cases is about the same. If the node has a *good* parent, then from our observations, one such parent is enough to get the data. However, we see that in both cases the number of parents is 3 to 5. That is probably because of group dynamics where parents leaving the channel could cause disruption for children that do not have other parents already sending data. In such a scenario, looking for a new parent and establishing a connection with that parent could take valuable time and have an adverse effect on the performance observed by the client.

[3]As defined in Section III

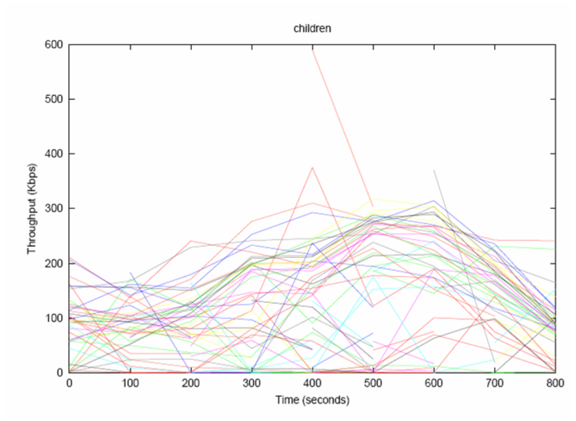
The number of children supported by the two types of nodes is very different. The high capacity node supports about 15-20 children, while the low capacity node supports no children. This points to the built in unfairness in the system.

In figure 5, we see the bandwidth received and sent by *cmu*. The parents send 100-200Kbps of data and there are 3-5 parents for this host. The number of children is much higher, each one is being sent about 100-200Kbps.

Another interesting observation made during this analysis was of the structure of the data plane. We observe that the number of hosts that are communicated with are much greater than the number of parents or children. It appears that the delivery of the video stream is done through an unstructured data plane where connectivity is maintained through randomness. It has been known through work done in [8] and [9] that the probability

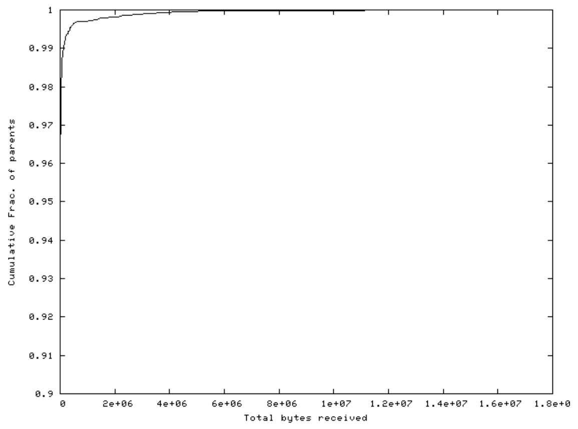


(a) parents

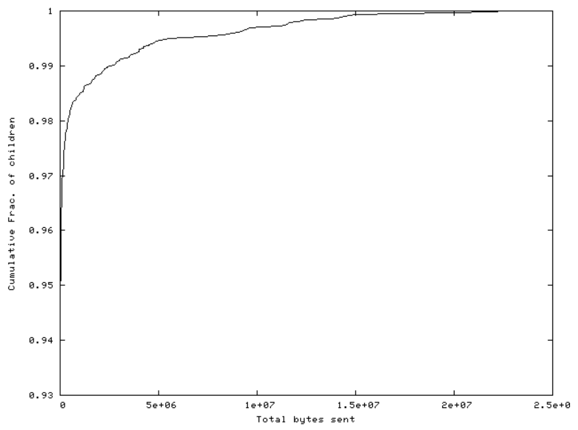


(b) children

Fig. 5. Bandwidth to Peers for high capacity node



(a) parents



(b) children

Fig. 6. Fraction of parent and child relationships for a high capacity node

of connectivity for a random graph increases as the outdegree of a node increases. Figure 6(a) plots the cumulative fraction of peers that the client received data from versus the total number of bytes received. Figure 6(b) plots the same graph, this time for peers that data was sent to versus total number of bytes sent. The figures show that the host in our example, *cmu*, communicated with a large number of other hosts even though actual parent and child relationships established are low. About 1% of the hosts communicated with became parents, while only about 2% of them became children. The large number of communication links established point to a random structure where the root is the source of the broadcast and all other hosts try to keep connected to the root via large number of links.

B. Locality of Peers

As mentioned in Section IV-A, the data plane appears to be constructed randomly. However it is not possible to analyze the entire structure as we have limited visibility into the data plane structure. We tried to analyze the three levels of the structure that are available to us through our measurement. This includes the parents and children of a host. Figure 7 shows the visibility we have to the entire structure.

Even with the limited visibility we have to the entire data plane structure, we are able to do some analysis on the relationship between the three levels. We use the concepts of distance and cost introduced in Section III for this analysis.

We analyze the cost of data received at a high capacity and a low capacity node. Figure 8 shows the cost of download for a few cases. Figure 8(a) plots the average cost of download per byte versus time for the *cmu* run. We can see that the cost of downloading data is about 13K miles per byte. That is approximately the distance across the pacific ocean. On further visual investigation of the parents, it is clear that almost all of the parents

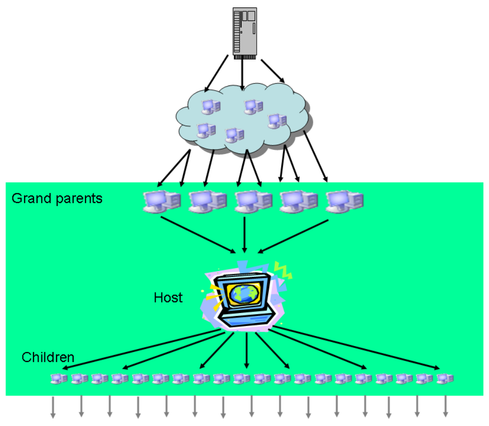


Fig. 7. Visibility of the data plane

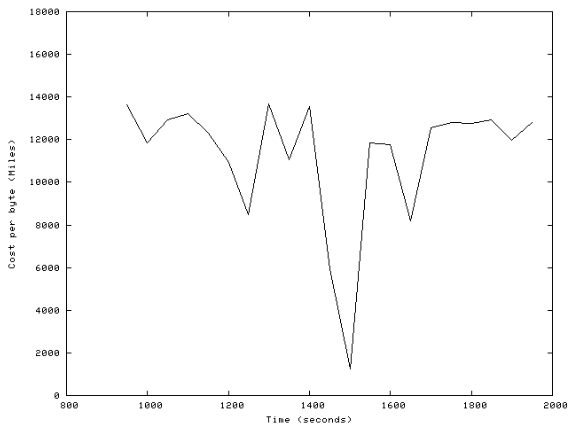
are hosts in Asia[4]. Figure 8(b) is a similar figure for a run of *chic*. In this case, the cost of download is mostly under 8K miles with some spikes at 12K miles. Upon further investigation of the parents, it seems like most of the parents of this host were other hosts in America within 6K miles, while there were a few times where the prominent parents were in Asia (12K miles).

To explain this difference between the parent selection for each host, we have to assume certain properties of the system. We believe that the system chooses parents solely based on performance. The high capacity host, being in a very favorable environment gets good performance when it connects to the parents in Asia. Therefore, it stays with those parents and does not change to closer, local parents. On the other hand, the low capacity node is in a bandwidth restricted environment going through an ISP that has strict contracts about bandwidth with other ISP's. This host, therefore, does not get good performance from parents in Asia and therefore has to choose parents that are closer. Some of these closer parents are within the same ISP as the host itself and thus provide a much better performance.

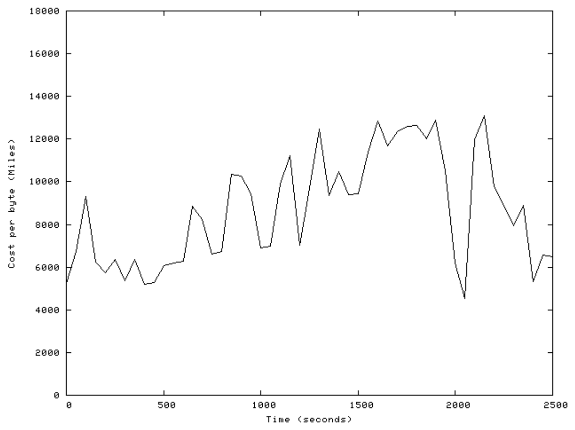
We can also look at the high capacity node's children and see their cost and distribution. Figure 9(a) shows the cost of uploading data to the children for the host *cmu*. The average cost is around 5K miles, but the graph in Figure 9(b) clearly shows what the distribution of children looks like. This graph is a CDF of child distances for the *cmu* host. It is clear from this graph that about 60% of the children are in Asia (13K miles) while the other 40% are either in the US or Europe.

This last set of results is very interesting as it points to the inefficiency of the systems. Data is sent from Asia to a host in the US, only to be sent back to other hosts in Asia. This fact also reaffirms the hypothesis of a random structure, that does not account for locality while making decisions about selecting parents for a particular client.

[4]mainly in China and Japan



(a) cmu



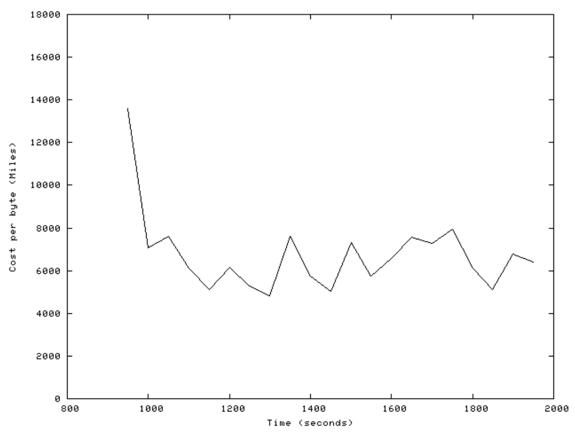
(b) chic

Fig. 8. Cost of download

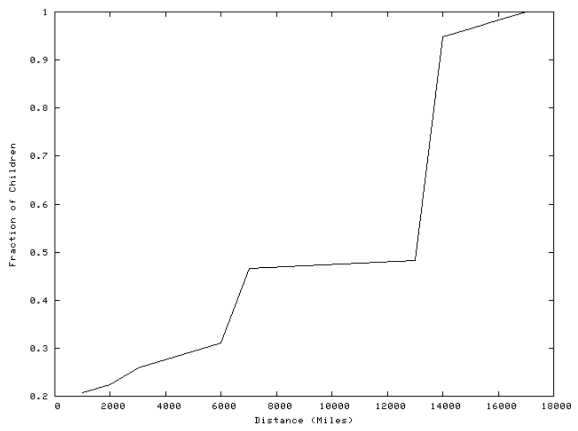
C. *Stability of the data plane*

We have concluded that the data plane structure of the applications is random, with the only selection criteria for a parent to child relationship being the performance received. We can also analyze the stability of this random structure by looking at the changes in the structure over time. In this analysis, we have to account for group dynamics that necessitate a change in the structure. We do not have an accurate measure of the group dynamics from the measurements performed. We still believe that the results are interesting and shed some light on the data plane. The stability of the data plane can be assessed by using the metrics defined in Section III-G.

Figure 10(a) plots the variable S_n versus time. Recall the S_n is a metric defined in Section III-G and measures

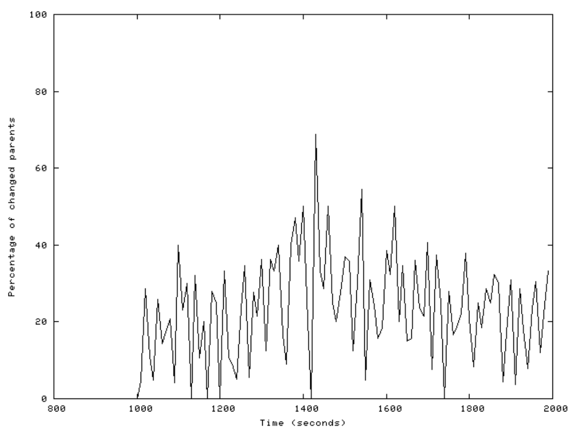


(a) Average

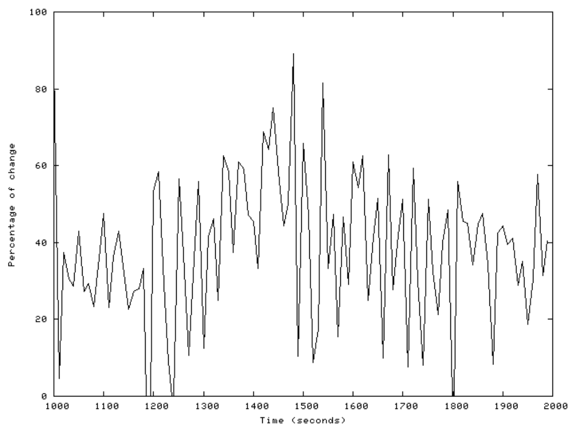


(b) CDF

Fig. 9. Cost of upload for high capacity node



(a)S_n



(b)X_n

Fig. 10. Percentage of parents changed

the rate of changes of parents over time. The graph shows the instability of parents for a run of *pittd* with an average of about 30% of changed parents between the intervals. This means that on average, 3 out of the 4-5 parents are different from one interval to the other. The graph in Figure 10(b) shows the stability using the X_n variable. Both these graphs point to the high rate of change of parents. Although we have not discounted for the group dynamics, we do not believe that all this variation can be explain through group dynamics and that in fact, most of this variation is due to the random nature of the data plane.

V. Summary of Results

As part of this analysis, we identified the major shortcomings and the challenges that applications in this class are likely to face.

A. Inefficient Distribution of Data

From the analysis of the control and data planes, we conclude that there is no planning or co-ordination involved in the decision making. A lot of important decisions, such as how to pick a parent, seem to follow a randomized greedy algorithm. The inefficiency is highlighted by the analysis of locality of upload and download for high capacity nodes. As noted earlier, test nodes in North America often download data from hosts in Asia, only to upload large amounts of data back to hosts in Asia.

B. Unfairness

It is the richly connected high capacity nodes that have to pay the price of the inefficiency in the system. In our analysis, each such node supported at least 15-20 children. Clearly, the system depends entirely on the availability of such nodes.

C. NAT handling

We have seen that the applications do not handle NAT's effectively. A host behind a NAT can receive traffic, but is unable to send any data to other hosts. These systems need to implement NAT identification and traversal using well-known protocols, such as STUN and TURN [22], [23] to deal with such connectivity restrictions.

D. Transport Protocol

The systems analyzed use HTTP as a ubiquitous mechanism to join the P2P network. However, most communication after that follows using TCP, with UDP being used sparingly. As most of the data transfers occur in small chunks, the delay properties of TCP for small amounts of data might not be ideal for real-time applications. Also, we see that the clients setup TCP connections with each other even if they are not receiving data on these connections yet. The overhead associated with setting up multiple TCP connections on a single host might be an issue in large networks.

E. Security

As seen by this analysis, most of the control protocols are not encrypted. This can lead to malicious attacks that can make key components of the system ineffective. The messages are sent over HTTP, UDP and TCP in plain text and can be used to interfere with the normal operation of the system.

VI. Related Work

Significant amount of research has been conducted in assessing the viability of peer-to-peer live video streaming, and some encouraging results have been reported [1], [3]–[5], [7], [11], [12], [16]–[18], [21], [25], [28]. However, most of these systems have been tested only in research environments, and the deployment experience of these applications is very limited.

We are aware of only one other study (conducted simultaneously with our work) that measures proprietary P2P video broadcast systems [30]. Perhaps due to limitation of space, they give only a brief overview of the system and discuss performance with respect to relatively few metrics. The results from our experiments were consistant with the ones reported by them. However, our work provides a more comprehensive analysis. For example, by analysing the rate of change of peers, we are able to quantify the stability of the system. In addition, our analysis of locality is more detailed. We calculate the cost of download or upload in terms of distance per byte, which gives a much better idea of the inefficiency in the system. Also, we provide a more thorough description of the control protocol used by the system.

VII. Conclusion

This study is one of the first attempts to measure commercial scale video broadcast applications that use peer-to-peer technology. These applications are different from others in their commercial content that can attract large numbers of users. We have gained valuable insight to the working of these applications by analyzing the control traffic in different scenarios. This helped us identify the key system components in applications of this class. Furthermore, a methodology is formulated to study the data planes of such applications from a single point of observation. Metrics such as resource usage, locality of download and stability of the distribution structure are defined, that can be used to study applications in this space.

In our study of these applications, we note that the greedy algorithms used and the locality independence of the underlying model leads to very high resource usage across key network boundaries. There is also a lack of tit-for-tat fairness in these applications which further lead to uneven distribution of bandwidth upload. We also observe that these systems make it difficult to implement fair bandwidth restrictions, which is a key component of a successful application. The data distribution structure is built randomly without any consideration of bandwidth. Even if such restrictions could be placed in the building of the data distribution structure, the instability observed in the resulting structure makes it challenging to implement a convergence in the data plane.

These systems prove the possibility of peer-to-peer random distribution structures for live video streaming. However, key questions as to the network usage, locality of download and stability of the distribution structure remain to be answered.

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, August 2002.

[2] Bit torrent. http://www.bittorrent.com/.

[3] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Content Distribution in Cooperative Environments. In *Proceedings of SOSP*, 2003.

[4] Y. Chawathe. Scattercast: An architecture for Internet broadcast distribution as an infrastructure service. Fall 2000. Ph.D. thesis.

[5] Y. Chu, S.G. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM Sigmetrics*, June 2000.

[6] Y.-H. Chu, A. Ganjam, T.S. Eugene, S. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an interet broadcast system based on overlay multicast. *USENIX Annual Techincal Conference*, June 2004.

[7] J. Albrecht D. Kostic, A. Rodriguez and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of SOSP*, 2003.

[8] Erdos and Renyi. On random graphs. *Publ. Math. Debrecen.*, 6:290–297, 1959.

[9] Erdos and Renyi. On the evolution of random graphs. *Magyar. Tud. Akad. Mat. Kut. Int. Kőzl.*, 5:17–61, 1960.

[10] Ethereal. http://www.ethereal.org/.

[11] P. Francis. Yoid: Your Own Internet Distribution, http://www.aciri.org/yoid/. April 2000.

[12] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, October 2000.

[13] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is P2P dying or just hiding. In *Proceedings of Globecom*, November 2004.

[14] T. Karagiannis, A. Broido, K. Claffy, and M. Faloutsos. Transport Layer Identification of P2P Traffic. In *International Measurement Conference*, October 2004.

[15] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should Internet Service Providers Fear Peer-Assisted Content Distribution. In *Proceedings of IMC*, pages 63–76, October 2005.

[16] J. Liebeherr and M. Nahas. Application-layer Multicast with Delaunay Triangulations. In *IEEE Globecom*, November 2001.

[17] A.M. Kermarrec M. Castro, P. Druschel and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communications Vol. 20 No. 8*, Oct 2002.

[18] V.N. Padmanabhan, H.J. Wang, and P.A Chou. Resilient Peer-to-peer Streaming. In *Proceedings of IEEE ICNP*, 2003.

[19] PPLive. http://www.pplive.com/.

[20] QQLive. http://tv.qq.com/.

[21] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level Multicast using Content-Addressable Networks. In *Proceedings of NGC*, 2001.

[22] J. Rosenberg, R. Mahy, and C. Huitema. Traversal Using Relay NAT (TURN). IETF Internet Draft, Febuary 2005.

[23] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. RFC 3489: STUN - Simple Traversal of UDP Through NATs, march 2003.

[24] SOPCast. http://www.sopcast.org/.

[25] S.Q.Zhuang, B.Y.Zhao, J.D.Kubiatowicz, and A.D.Joseph. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination, April 2001. Unpublished Report.

[26] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. In *International Measurement Conference*, October 2005.

[27] TVAnts. http://www.tvants.com/.

[28] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *Proceedings of Fourth International Workshop on Networked Group Communication (NGC)*, October 2002.

[29] X. Zhang, J.C. Liu, B. Li, and P. Yum. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *Proceedings of IEEE INFOCOM*, March 2005.

[30] X. Hei, C. Liang, J. Liang, Y. Liu, and K.W. Ross Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System. In *IPTV Workshop, International World Wide Web Conference*, May 2006.