Multimedia Tools and Applications manuscript No.
(will be inserted by the editor)

CStream: Neighborhood Bandwidth Aggregation for Better Video Streaming

Thangam Seenivasan · Mark Claypool

Received: date / Accepted: date

Abstract Despite the popularity of watching videos online, challenges still remain in video streaming in many scenarios. Limited home broadband and mobile phone 3G bandwidths mean many users stream videos at compromised quality. To provide additional bandwidth for streaming, we propose *CStream*, a system that aggregates bandwidth from multiple cooperating users in a neighborhood environment for better video streaming. CStream exploits the fact that wireless devices have multiple network interfaces and connects cooperating users with a wireless ad-hoc network to aggregate their unused downlink Internet bandwidth. CStream dynamically generates a streaming plan to stream a single video using multiple connections, continuously adapting to changes in the neighborhood and variations in the available bandwidth. CStream is developed and evaluated on a test bed of computers, allowing for a detailed, controlled evaluation of performance. Analysis of the results shows a linear increase in throughput over single-connection streaming and improved video quality as the number of cooperating users in a neighborhood increase.

Keywords video streaming · wireless · bandwidth aggregation

1 Introduction

The popularity of video streaming systems has grown tremendously in the past few years. Sites like YouTube[1] support user generated video content and contribute a significant amount of Internet traffic [GALM07]. According to a 2009 survey by Cisco, video accounts for 25% of total Internet traffic and is expected to contribute about 50% of Internet traffic by 2012 [oEE09].

The quality of video streaming is largely dependent on the downlink Internet bandwidth available to the end user. Although Web sites like YouTube support high quality videos, due to the limited Internet bandwidth available today, many users stream videos at low quality and compromise on their user experience. Systems like Orb[2] dynamically adapt the quality of videos based on the available bandwidth, but still end up streaming videos at low quality because of the limited bandwidth [KSCK10]. Additionally, video streaming in mobile devices, like smart phones, are becoming increasingly important for video delivery. Unfortunately, Internet bandwidth to mobile devices is expected to take a long time to meet the demand for real-time streaming. Hence, despite the popularity, providing bandwidth for high-quality Internet video streaming still remains a challenge in many scenarios.

To overcome the problem of limited bandwidth for streaming, we propose *CStream* (Collaborative Streaming), a system that aggregates bandwidth from multiple cooperating users in a neighborhood. The motivation for the system stems from the fact that although individual users may have limited bandwidth, the high density of Internet users in a neighborhood with only a fraction active at any given

Worcester Polytechnic Institute 100 Institute Road
Worcester, MA, 01609, USA
E-mail: thangam, claypool@cs.wpi.edu

[1] http://www.youtube.com
[2] http://www.orb.com

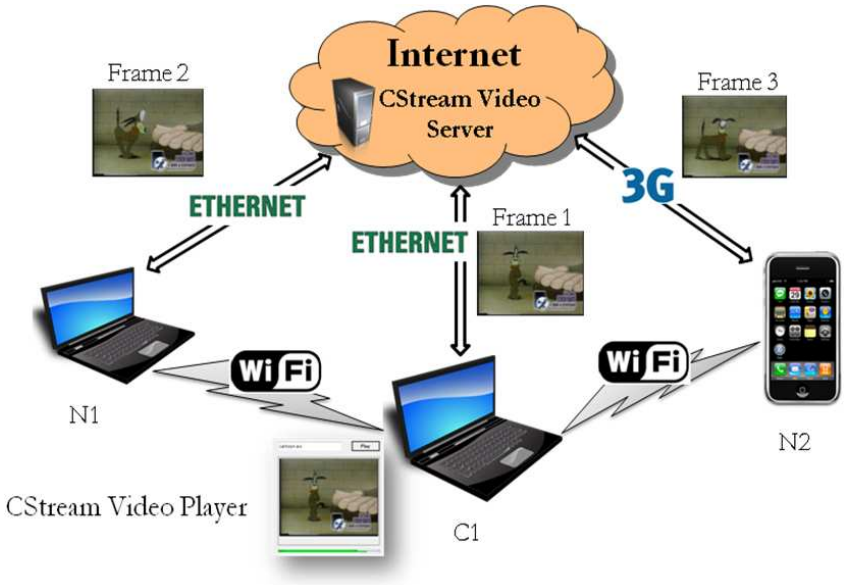

Fig. 1 Aggregating bandwidth from neighbor nodes for video streaming

time provides the opportunity to exploit the unused bandwidth to improve video streaming quality. When a user streams a video, CStream aggregates bandwidth by connecting to nearby cooperating users and uses their Internet connections in addition to the user's own connection.

Many video servers use layered encoding to support streaming in heterogeneous networks. Streaming additive layers when more bandwidth is available enhances video quality. CStream can exploit layered encoding of video by streaming different enhancement layers through different neighbors. As the number of neighbors contributing to the client bandwidth increases, CStream can stream videos at higher quality. In the current implementation, CStream does not use enhancement layers but streams individual frames through multiple links for better throughput and video quality.

Users increasingly use mobile devices to access the Internet. For example, many users at home and work use laptops or smart phones for Internet access. These mobile devices are typically equipped with multiple network interfaces to offer flexibility of Internet access. Laptops have both Ethernet and wireless 802.11 interfaces. Mobile phones may have 3G, 802.11 and Bluetooth interfaces. CStream exploits the fact that devices with multiple network interfaces can connect to other devices simultaneously while being connected to the Internet. Specifically, CStream uses the 802.11 wireless interface to connect to neighboring nodes to aggregate bandwidth. Consider a common scenario in a home community where users connect to their ISPs using laptops. CStream connects cooperating, but idle, neighboring laptops in a wireless ad-hoc network to aggregate bandwidth for video streaming. In another scenario at an airport, CStream connects idle, cooperating mobile phones together in an ad-hoc wireless network to aggregate their 3G bandwidth for better streaming.

The main components of the CStream system are the Video Server, the CStream Client (running the CStream Video Player) and the Neighbors running a simple support application. The Video Server encodes and stores the videos that can be requested by clients. A user requests a video through the CStream Client. The Client then creates an ad-hoc network, asking if any node in the neighborhood is willing to contribute bandwidth. Willing Neighbors having spare bandwidth connect to the ad-hoc network initiated by the Client. The Client informs the Video Server about the Neighbors. The server streams the video to the Client through all the available links (both Client and Neighbors). The Neighbors then act as proxies, sending the frames received from the Server to the Client through the ad-hoc network, thereby contributing additional bandwidth.

Figure 1 shows an example scenario streaming with CStream. Client C1 and Neighbor N1 are connected to Internet by Ethernet through their ISP. Neighbor N2 is a smart phone connected to Internet through 3G. All three nodes are nearby (i.e. with good wireless connections). C1 uses the CStream Video

Player to request a video. Neighbors N1 and N2 have spare bandwidth and, being willing to cooperate, connect to C1 using a wireless ad-hoc network. C1 informs the CStream Video Server about its active Neighbors and the server streams the video through all three links, thereby aggregating bandwidth to achieve higher throughput. For example, the server sends frame 1 directly to the Client, sends frame 2 through Neighbor N1 and frame 3 through Neighbor N2. While streaming, the system fully utilizes the available bandwidth in all three links and adapts dynamically to any change in bandwidth and dynamically adapts to changes in neighborhood.

CStream is developed and evaluated using a four-node test bed comprising a Video Server, a Client and two Neighbors. Running experiments in a closed environment enables us to control the bandwidth of the Client and Neighbors to the Video Server, allowing evaluation of CStream for various bandwidth settings. CStream is evaluated by varying the number of Neighbors, location of the Neighbor nodes (to vary the wireless throughput) and video content. CStream adaptiveness under dynamic conditions when Neighbors join and leave in the middle of video streaming is assessed. Performance metrics include aggregate throughput and video quality, measured by playout time, start-up delay and re-buffer events. Effectiveness of the streaming protocol is evaluated by measuring the Server's ability to effectively and proportionally use the bandwidth of the available links.

Analysis shows a near linear improvement in throughput and video quality as the number of Neighbor nodes increases. For example, when there is one cooperating Neighbor with the same bandwidth as the Client, the performance improves by almost 2x, and when there are two neighbors, the performance improves by almost 3x. As the number of Neighbors increases, the video start up delay, the playout time and the re-buffer events decrease almost linearly. Results show that the system effectively utilizes the available bandwidth and adapts quickly to neighbors joining and leaving. Also, the ratio of the frames distributed by the server across multiple links matches their ratios of bandwidth.

The contributions of the work include:

- A novel system for video streaming that connects neighboring nodes in an ad-hoc network to aggregate Internet bandwidth;
- A Video Plan Manager that determines how to stream video through multiple Internet connections, dynamically adapting to the changing neighborhood (nodes joining and leaving);
- A frame distribution scheme that distributes video frames across multiple connections and makes full utilization of the available bandwidth;
- Detailed performance evaluation of the entire CStream video streaming system over a range of video and network configurations, showing linear improvement in throughput and video quality as the number of cooperating Neighbors increases;
- Contribution of the CStream code for other researchers to use for comparative purposes or to extend for their own work. CStream is available for download online at: http://perform.wpi.edu/downloads/#cstream

The rest of this paper is organized as follows: Section 2 explores work related to CStream; Section 3 discusses the design and architecture of CStream; Section 4 explains the implementation details of CStream; Section 5 provides details on the experiments, including setup, results and analysis; Section 6 summarizes our conclusions; and Section 7 presents possible future work.

2 Related Work

There has been considerable research on increasing Internet bandwidth and application throughput in recent years. One focus area is on effectively using the available bandwidth at a single network interface (for example, using download accelerators). With the proliferation of multi-homed devices, there has been an additional focus towards aggregating bandwidth from multiple Internet connections on a single device to improve application throughput. Recently, with the increasing popularity of devices with multiple interfaces such as smart phones, some research prototypes have focused on aggregating bandwidth across multiple devices to improve application throughput. Transport protocols beyond TCP and UDP can enable multi-link support, as well. Single file videos must be specially encoded to enable streaming over multiple links. This chapter describes some related work in these areas in detail and discusses how CStream differs from each.

2.1 Download Accelerators

Download accelerators and peer-to-peer (P2P) distribution systems can improve file download rates by getting parts of the file over multiple connections. Download accelerators improve the download rate on a single Internet link by opening multiple connections to mirrored servers in parallel and downloading different parts of a file simultaneously. The download performance improves because using a single, poorly provisioned server may severely impact the download performance, while downloading in parallel from multiple servers reduces the impact of a single bad connection. P2P networks can improve video stream rates by streaming different parts of a video from multiple nodes. In P2P networks, a client connects to multiple peers and parts of the file are downloaded from different peers. Similar to file download techniques accessing multiple servers, P2P networks provide link diversity thereby improving the download performance.

TRIBLER is a social based, P2P file sharing system that uses a paradigm of social networks to aid in content discovery, recommendation, and downloading [PWB+08]. TRIBLER can be deployed as a set of extensions to BitTorrent, to enable fast content discovery and improve download performance. Performance gains for downloading files are realized through a cooperative downloading protocol which provides incentives for donating upload bandwidth by peers. The authors show theoretical improvements up to 2x to 6x for ADSL links.

Rodriguez et al. implement a dynamic parallel-access scheme where clients connect to mirror sites using unicast TCP and dynamically request different pieces of a document from different sites, thus adapting to changing network and server conditions [RKB00]. Rodriguez et al. evaluated their scheme with various mirrored sites for different document sizes under different network/server conditions. The results show dramatic speedups in downloading a document, even when network or server conditions change rapidly and servers have different connection qualities.

Liu et al. review the state-of-the-art of peer-to-peer Internet video broadcast technologies [LRLZ08]. The authors describe the basic taxonomy of peer-to-peer broadcast and summarize the major issues associated with the design of broadcast overlays. They examine two approaches, namely tree-based and data-driven, and discuss their fundamental trade-offs and potential for large-scale deployment.

Both download accelerators and P2P use multiple connections and parallel downloads to enhance their download performance. Although these systems can improve download speeds over traditional client-server systems, they can never achieve capacity more than the downlink bandwidth available at the end-host. CStream overcomes the scenario where the user's Internet downlink is the bottleneck and increases streaming bandwidth through multiple network interfaces. CStream can achieve more than the user's downlink bandwidth since the bandwidth of the neighbor's Internet downlinks are aggregated. For CStream the bandwidth gain is limited by the number of collaborating Neighbors and not the user's downlink capacity as in download accelerators and P2P systems.

2.2 Network Sharing

Network sharing is a well-explored field. Wireless Mesh Networks provide connectivity to users in a neighborhood which do not have direct Internet access [AWW05]. In mesh networks, nodes in a neighborhood connect wirelessly to form a grid and share Internet access from one or a few nodes which do have an Internet connection. Wireless mesh networks are a cost effective way of increasing Internet connectivity, but can have scalability issues and suffer in performance as the number of nodes increases.

CStream is similar in theme to mesh networks in forming a neighborhood network, but unlike mesh networks which uses a single Internet connection per flow, CStream nodes aggregate bandwidth from all nearby nodes in addition to its own Internet connection and additional nodes all increase the bandwidth since only a single client in the CStream system is streaming a video at a time.

2.3 Stream Control Transmission Protocol

The Stream Control Transmission Protocol (SCTP) is a Transport Layer protocol, providing message-oriented communication like UDP, while ensuring reliable, in-sequence delivery with congestion control like TCP [(Ed07]. SCTP transports a given sequence of bytes as a message, rather than treating the

entire transmission as stream of bytes, as does TCP which requires streaming video in that applications must record their own markings to delineate messages. As in UDP, SCTP sends a message in one operation, and that exact message is passed to the receiving application in one operation. SCTP also supports multihoming in which either endpoint of a connection can have more than one IP address. While multihoming, and message-oriented, reliable communication are both core features desirable for CStream, the multihoming enabled by SCTP does not readily allow routing, which is done at the IP layer, across to a Neighbor's ISP connection.

Transport layer protocols, such as SCTP, and any other approach that aggregates bandwidth at the transport level, are often more robust and efficient than code implemented by individual applications at the application layer. However, without major operating systems support, application developers cannot make use of novel transport layer protocols on a variety of platforms. SCTP must be implemented in the operating system in order to be used and while SCTP is supported by modern versions of Linux and FreeBSD, there is no native kernel support for MacOS or Windows, making it impractical for the approach of sharing bandwidth with the typical Neighbor.

2.4 Bandwidth Aggregation

Bandwidth aggregation techniques seamlessly use multiple Internet connections to form a single, higher-bandwidth connection. The system presented by Chebrolu and Rao assumes multiple Internet connections on the same device through multiple interfaces and aggregates bandwidth across these interfaces for video streaming [CR06]. The number of network interfaces per device is limited (usually two) and the maximum capacity this system can achieve is the sum of the Internet bandwidths at these interfaces. CStream also does bandwidth aggregation but instead of combining bandwidth from network interfaces in a single device, CStream aggregates Internet bandwidth from multiple neighbors. Hence, CStream can achieve greater capacity than such multi-homed systems, only limited by the wireless capacity.

COMBINE aggregates bandwidth from multiple nearby phones by forming a wireless ad-hoc network between the nodes [APRT07]. The 3G bandwidth on the phones is then combined to improve download times over HTTP. The main focus of COMBINE is an incentive system that is based on battery energy cost, with an accounting system that pays and bills users based on their bandwidth contribution.

Link-alike is similar to COMBINE but is used to improve the upload capacity of the client [JJK$^+$08]. Link-alike tries to increase the upstream capacity for a client by aggregating the uplink capacities of the nodes in neighborhood. Link-alike addresses the challenges of operating in an environment that is lossy, broadcast in nature and half-duplex by using opportunistic wireless reception, a novel wireless broadcast rate control scheme, and preferential use of the wired downlink. Evaluation shows Link-alike provides significantly better throughput than previous solutions based on TCP or UDP unicast.

Since systems like COMBINE and Link-alike support only file transfer, they do not have specific constraints on the upload or the download times, and hence can use straight-forward work distribution techniques to assign a flow across multiple links. CStream supports real-time video streaming and distributes frames across multiple connections for improved video quality. CStream effectively utilizes the available bandwidth and dynamically adapts to neighbors joining and leaving.

2.5 Virtual Interfaces

Although CStream assumes there are multiple network interfaces on the client device, this is not a requirement. Technologies like Multinet allow a single wireless interface to act as multiple network interfaces, enabling them to connect to multiple nodes at the same time [CBB04]. For example, a node can connect to the Internet through an access point at the same time it connects to a neighboring node, both connections using the same wireless card. Multinet is transparent to the user and is agnostic to the upper layer protocols.

FatVAP is also system which enables a single wireless card to connect to multiple access points at the same time, aggregating the bandwidth available [SKK08]. FatVAP works with unmodified APs and is transparent to applications and the rest of the network stack. Evaluations show FatVAP can deliver about a 2.5-fold increase in the median throughput and about a 2.8-fold decrease in the median response time.

In the current implementation of CStream, the client devices have multiple interfaces, however by using technologies like Multinet and FatVAP, CStream can work on devices with only a single network interface.

2.6 Encoding Video for Multi-link

The encoding and scheduling requests to enable a single video to be streamed over multiple links is an ongoing research area. Typical approaches divide a complete file into smaller, fix-sized segments that can be retrieved independently [FNI04]. Ongoing throughput calculations can be used to make decisions on the segment-sizes.

Evensen et al. [EKG+11] present a client-side request scheduler for multi-link streaming by dividing videos into independent segments with constant duration. The segments can then be requested by the client to be sent over separate links, allowing bandwidth aggregation at the client. Segment sizes are based on throughput calculations on the links, allowing tuning to the individual link bandwidths. The focus on such work is complementary to that of CStream. Notably, segmentation and video scheduling techniques can be incorporated into the existing CStream framework, replacing the request method presented in our work, as appropriate.

3 Architecture

The CStream architecture has three distinct components: the Video Server which stores and streams the video content, the Client node which requests and plays the video, and the Neighbors which help to aggregate bandwidth for better video quality at the Client.

The current design assumes: 1) neighbors having spare bandwidth are willing to participate in CStream without any incentives, 2) the client and all neighbor nodes are trustworthy, and 3) both client and neighbors have installed the CStream software. These assumptions allow demonstration of the core CStream ideas, with relaxing of assumptions 1 and 2 left as future work.

3.1 Challenges

There are several challenges in building a system that aggregates bandwidth from nearby nodes through an ad-hoc wireless network to improve video streaming.

3.1.1 Neighbor Discovery and Maintenance

The Client and Neighbors need to find each other and connect through an ad-hoc wireless network. Once connected, the Client needs to have up-to-date knowledge of the active Neighbors willing to contribute bandwidth for streaming. Neighbors should be able to join and leave the system at any time during video streaming without major impact on the video stream (e.g. without halting the video connection).

3.1.2 Multi-path Streaming

The Video Server needs to be able to stream a single video through multiple nodes. The server needs to know of both the Client and Neighbor end-points to be able to send video frames from the same video to each, separately. To preserve the temporal aspects of the video, the server needs to distribute the frames based on the bandwidth available at each link and dynamically adapt to the changes in the bandwidth. An ideal distribution technique fully utilizes the available bandwidth and proportionally distribute frames based on the ratios of the available bandwidths.

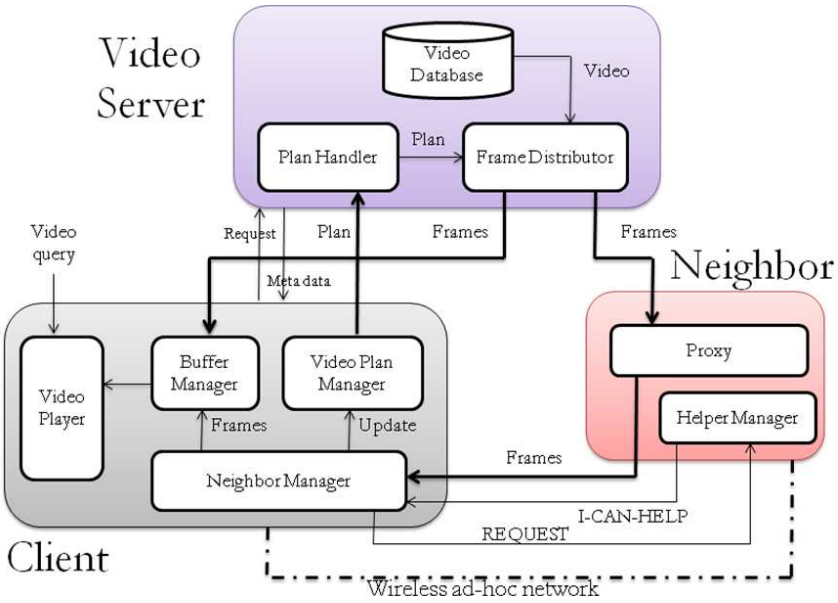

Fig. 2 CStream architecture

3.1.3 Dynamically Changing Neighborhood

The system should gracefully adapt to changes in the network neighborhood. Neighbors can then join and leave at any time during the video streaming, not just between video requests. When new Neighbors join, the system should adapt quickly to make use of the newly available extra bandwidth. When Neighbors abruptly leave, the system should recover from any frames lost, and continue playout over the remaining active links.

3.1.4 Buffering and Playing

The Client needs to buffer video frames received through different links which may result in shuffling of frame order, and play them out in their proper, temporal order. The system should have an appropriate buffering mechanism which decides whether to wait for the late frames or discard them, in addition to smoothing out variations in bandwidth.

3.2 Architecture

The CStream design addresses all the above challenges. Figure 2 shows the architecture of CStream. The system has three distinct components: the Video Server, the Client and the Neighbors. CStream assumes that multiple interfaces are present in the Client and Neighbors so that they can form a wireless ad-hoc network while still being connected to the Internet through their wired network interface.

In brief, CStream works as follows. The CStream Client forms a wireless ad-hoc network with Neighbors which are idle so that the Client can use their bandwidth. The Client periodically updates the Video Server with the Neighbor information and the Server streams the video to the Client using both the Client and the Neighbor bandwidth. In essence, a participating Neighbor simply forwards video frames received from the Server to the Client. Each component is explained in greater detail below, with implementation details deferred until Section 4.

3.2.1 Client

Users request a video using the custom Video Player running on the Client. The Client then sends the request to the Video Server, which replies with the video meta-data consisting of the number of frames

and the frame rate. The Client forms an ad-hoc network with the Neighbors and informs the Video Server about all the available links that should be used to stream the video. As the frames start arriving, they are buffered and later the video is played by the Video Player.

The Client has four main components:

Neighbor Manager The role of the Neighbor Manager is summarized below:

1. When a user requests a video, the Neighbor Manager creates an ad-hoc network and waits for Neighbors to join.
2. The Neighbor Manager periodically broadcasts REQUEST messages to find new Neighbors in the ad-hoc network which are willing to contribute bandwidth.
3. The Neighbor Manager keeps an updated knowledge of active Neighbors willing to help in streaming. The Neighbor Manager monitors for periodic heartbeat messages (I-CAN-HELP messages) from the Neighbors and keeps track of neighbors joining and leaving the neighborhood.
4. The Neighbor Manager periodically informs the Video Plan Manager about the active Neighbors and changes in neighborhood (new Neighbors joining and Neighbors leaving).
5. The Neighbor Manager receives video frames from the active Neighbors and forwards them to the Buffer Manager. The Neighbor Manager also keeps track of the frames received from each of the Neighbors and informs the Video Plan Manager.

Video Plan Manger The Video Plan Manager continually informs the Video Server about the latest streaming plan based on the current network neighborhood. The role of the Video Plan Manager is summarized below:

1. The Video Plan Manager constructs the streaming plan with Neighbor details and periodically updates the Plan Handler in the Video Server. The streaming plan consists of information about the Client and Neighbor links (IP address and port) that the Video Server can use to stream video.
2. When a new Neighbor, ready to contribute bandwidth, joins the ad-hoc network, the Video Plan Manager informs the Video Server about the Neighbor so that the Video Server can quickly use additional bandwidth to stream.
3. When a Neighbor leaves the network, the Video Plan Manager informs the Video Server so that the server can resend any lost frames and continue to stream through the remaining links.

Buffer Manager The Buffer Manager maintains the playout buffer that the Video Player uses to play the video.

1. Based on the meta-data response from the Video Server after the Client requests the video, the Buffer Manager initializes the playout buffer.
2. The Buffer Manager receives frames from the Video Server and stores them in the buffer.
3. The Buffer Manager also receives frames from Neighbors through the Neighbor Manager and stores them in the buffer.

Video Player The Video Player is the interface for the user in the CStream system:

1. A user can request a video using the Video Player in the Client.
2. The Video Player extracts frames from the playout buffer in the Buffer Manager and plays them.
3. The Video Player plays the video based on the meta-data of the video (in the current implementation, only the frame rate).
4. When the frame to be played is missing, the Video Player stops playout and waits for late frames to arrive. The stop and buffering mechanism is explained in detail in Section 4.

3.2.2 Neighbor

A Neighbor is idle and has spare bandwidth which it is willing to share with the Client to improve the Client's video streaming quality. Although Figure 2 shows only one instance of a Neighbor, multiple Neighbors can contribute bandwidth to a single Client. The role of each of the Neighbor components is explained below.

Helper Manager The role of the Helper Manager is:

1. When the Neighbor is willing to contribute bandwidth, it looks for an ad-hoc network created by a CStream Client and joins the network.
2. When the Helper Manager receives a REQUEST message from the Client, it starts sending I-CAN-HELP messages periodically to the Client. In the I-CAN-HELP messages, the Helper Manager puts the IP Address and the port which the Neighbor keeps open for the Video Server to stream the video.
3. When there is user activity or network activity (due to other applications), the Helper Manager stops sending I-CAN-HELP messages.
4. The Helper Manager disconnects from the CStream ad-hoc network when the user exits the CStream application.

Proxy The Proxy component in the Neighbor:

1. The Proxy keeps a port open that can be used to receive video from the Video Server.
2. The Proxy receives frames from the Video Server and forwards them to the Client through the ad-hoc network. Neighbors do not buffer the received frames and immediately forward any frames to the Client.

3.2.3 Video Server

The Video Server stores the videos that Clients can individually request. The Video Server streams a single video through the multiple links provided by the Client and Neighbors, adapting the streaming to changes in the network neighborhood.

Video Database The Video Database stores the uploaded videos that Clients can stream:

1. The videos are all encoded and stored in a single format, one that can be split temporally (i.e. along frame boundaries).
2. The Video Database splits a video into frames that are used as the unit for streaming, and allows queries to a specific frame in a video. The Frame Distributor extracts the frames from a video in the Video Database and distributes them through multiple links.

Plan Handler The Plan Handler's role is:

1. The Plan Handler receives a streaming plan from the Video Plan Manager in the Client. The Plan Handler initializes the Frame Distributor to stream according to the plan.
2. The Plan Handler receives plan updates from the Client about changes in the neighborhood. Upon being informed of a new Neighbor, the Plan Handler tells the Frame Distributor to include the new Neighbor in the streaming process. Similarly, upon being informed of a Neighbor that left in the middle of streaming, the Plan Handler tells the Frame Distributor to stop sending frames via that Neighbor.

Frame Distributor The Frame Distributor is the core component of the system that streams a single video through multiple links:

1. The Frame Distributor runs a frame assignment module that assigns frames in a video to stream through different links.
2. The Frame Distributor uses TCP to send the frames to the Client and the Neighbors.
3. The Frame Distributor adapts to change in bandwidth and effectively utilizes the links.
4. The Frame Distributor works with the Plan Handler to adapt to the changes in neighborhood. The Frame Distributor starts streaming to new neighbors when they join and recovers lost frames when a Neighbor leaves.

4 Implementation

This section presents implementation details of the protocols in the CStream system for: 1) Neighbor management, 2) Frame distribution, 3) Adaption to changes in neighborhood, and 4) Buffering and playout. Illustrative examples are used to explain each.

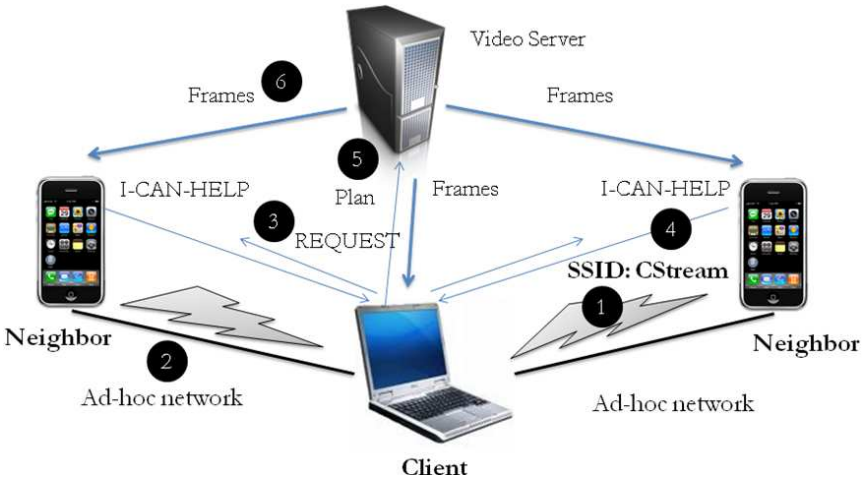

Fig. 3 Ad-hoc network formation and Neighbor management

4.1 Neighbor Management

This section explains the ad-hoc network formation and neighbor management. Specifically, we explain the implementation details of how the Neighbor Manager in the Client and the Helper Manager in the Neighbor form and manage the ad-hoc network.

Figure 3 shows an example scenario with a Client and two Neighbors. The user requests a video using the Client and the sequence of events step-by-step are shown with numbers in black circles.

Step 1: The Client creates an ad-hoc network with SSID `CStream`. If such a network already exists, the Client joins it. If the Client is already connected to the CStream ad-hoc network, it continues with Step 3. The same ad-hoc network can be maintained across multiple video requests. For instance, the Neighbors may be connected in the ad-hoc network all the time when they are idle even if a video is not requested. When there is local user activity, the Neighbor disconnects from the ad-hoc network and connects back when it becomes idle again. This method of maintaining the ad-hoc network helps reduce the streaming start-up delay of forming the neighborhood network.

Step 2: Neighbors which are near the Client are in the range of the CStream ad-hoc network. If they are running the CStream application and are idle, they join the network. Client and Neighbors get an IP address once they join the ad-hoc network.

Step 3: The Client broadcasts a REQUEST message in the ad-hoc network indicating that it needs to stream a video and is looking for neighbors to contribute bandwidth. In our implementation, IP broadcast in the ad-hoc network is used to broadcast the request.

Step 4: When a Neighbor in the ad-hoc network receives a REQUEST message, it starts responding to the Client with I-CAN-HELP messages. The I-CAN- HELP messages contain the Neighbor's IP address and port for the Video Server to stream the video via the Neighbor. The I-CAN-HELP messages are sent periodically to the Client (in our implementation, every second). The periodic I-CAN-HELP messages from the Neighbors are used by the Client to determine if a Neighbor is still actively participating in the streaming.

Using the I-CAN-HELP messages, the Neighbor Manager at the Client maintains a neighbor table with: Neighbor, IP address, Port, Last Update Time and Last Received Frame. For each Neighbor, the table stores the IP address and the port for streaming, the last time when the Client received an I-CAN-HELP message from the Neighbor and the last received frame from the Neighbor (used to recover lost frames when a Neighbor leaves). The Client decides that a Neighbor has left if it does not receive an I-CAN-HELP message for a specific amount of time (3 seconds in our implementation).

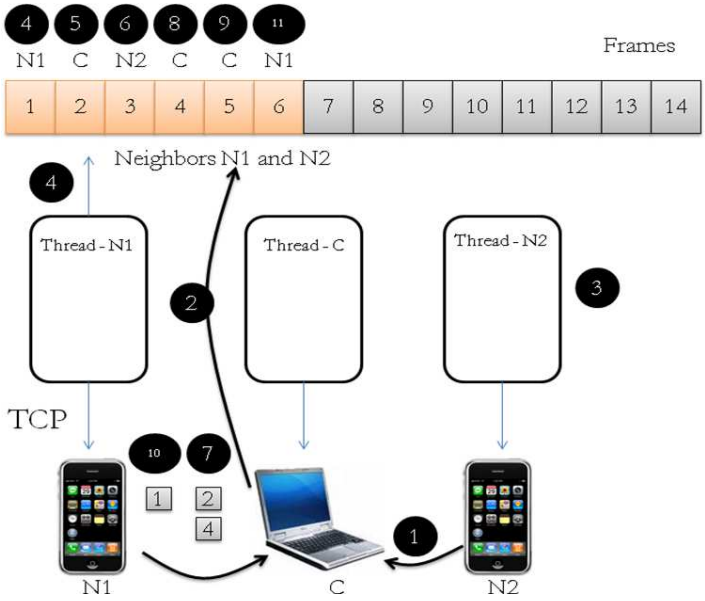

Fig. 4 Frame distribution example

Step 5: The Client sends the streaming plan periodically (every 500 msec in our implementation) to the Video Server. The streaming plan consists of the list of end points (IP address and port) to stream. The Client sends its own end point and the end point of the Neighbors that are active (i.e. last update time is less than 3 seconds). The streaming plan changes due to changes in the neighborhood (Neighbors joining and leaving) and the Client updates the Video Server with the new plan.

Step 6: Based on the streaming plan, the Video Server splits and streams the video across multiple links, described in Section 4.2.

4.2 Frame Distribution

The Video Server sends a single video through multiple links. The video files are stored in AVI format in the Video Database. While alternate video compression formats (e.g. H.264) would yield better compression rates, AVI allows demonstration and evaluation of CStream, with improvements due to alternate compression protocols left as future work. The Frame Distributor splits the video into frames and sends each frame through a single link. The flow sequence at the server after the Client sends the initial streaming plan is illustrated with an example in Figure 4. The steps of the protocol are shown in black circles. In the example there are two active Neighbors and a Client.

Step 1: As described in the Section 4.1, the Neighbors periodically send I-CAN-HELP messages to the Client. In this example, Neighbors N1 and N2 both send periodic I-CAN-HELP messages.

Step 2: The Client informs the Video Server about the Neighbors N1 and N2 as part of the streaming plan.

Step 3: The server has three links for streaming video. The server creates three threads to simultaneously send frames through the three links, where each thread services one link. The job of each thread is to fetch frames and send it down the link to the node. The server keeps all the frames in a Frame Queue and runs a simple frame assignment algorithm. Whenever a thread is ready to send a frame, it fetches the frame at the head of the Frame Queue and assigns the frame to that thread.

The frames are sent to the node using TCP since TCP provides automatic adaptation to the available bandwidth in each link. When there is spare bandwidth, the thread requests the next frame from the

Frame Queue and starts sending. In our implementation, the TCP sender window is small, about 32 KB for each node, and hence only about one frame for our source videos is accommodated in the sender buffer before fetching the next frame. This eliminates the problem of multiple frames being queued on the sender side when the Client and the Neighbor link are slow.

Step 4: Thread N1 requests a frame and frame 1 that is at the head of the queue is assigned to the thread. The thread starts sending frame 1 to the Neighbor N1.

Step 5: Thread C requests a frame and is assigned frame 2.

Step 6: Thread N2 requests a frame and is assigned frame 3.

Step 7: In the example, the Client has three times more bandwidth compared to the Neighbors, so frame 2 delivery is faster than frame 1 and frame 3.

Step 8: Since there is spare bandwidth in the Client link, thread C requests another frame and is assigned frame 4.

Step 9: Frame 4 is also sent to the Client and the thread C requests another frame and is assigned frame 5.

Step 10: Thread N2 finishes sending frame 1 to Neighbor N2, and N2 finishes forwarding frame 1 to the Client.

Step 11: N2 requests another frame and is assigned frame 6.

Thus, the system adapts to available bandwidth and distributes the frames to the links proportionally to their bandwidths.

4.3 Adapting to Changing Neighborhood

While streaming, especially during a long video, the number of Neighbors may change, mandating an adaption of the current streaming plan. This section explains the mechanisms CStream uses to adapt to these changes.

4.3.1 Neighbor Joining

The flow sequence of this Neighbor joining scenario is explained using Figure 5, a continuation of the example scenario shown in Figure 4. The Neighbor willing to donate downlink Internet bandwidth (Neighbor N3 in Figure 5) joins the `CStream` ad-hoc network. N3 then receives the periodic REQUEST broadcasts from the Client seeking streaming help.

Step 1: The Neighbor hearing the REQUEST messages starts responding to the Client with I-CAN-HELP messages.

Step 2: With a new entry in its Neighbor table, the Client updates the streaming plan, informing the Video Server about the new Neighbor.

Step 3: The server creates a new thread to send frames to the Neighbor N3.

Step 4: When a N3 thread requests a frame to send, it is assigned the next frame at the head of the queue (frame 7).

Thus, the CStream system adapts to new Neighbors joining and starts to use its bandwidth, quickly improving the overall throughput of the system.

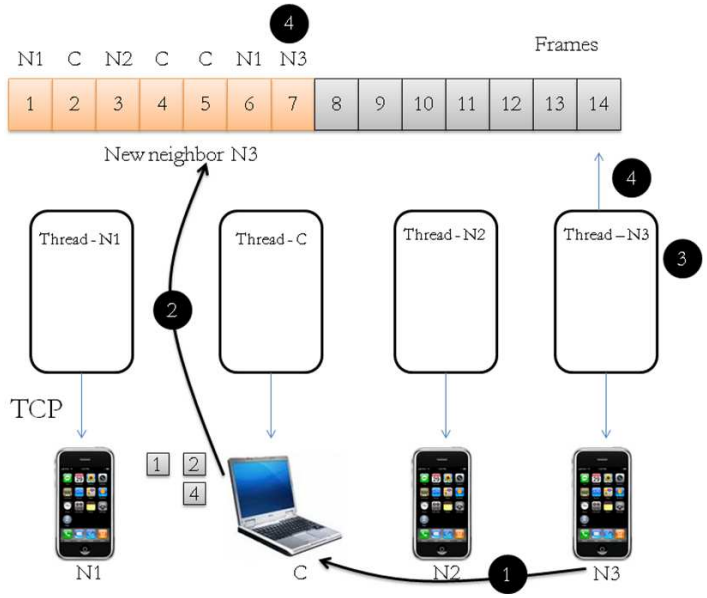

Fig. 5 Adapting frame distribution to Neighbor joining example

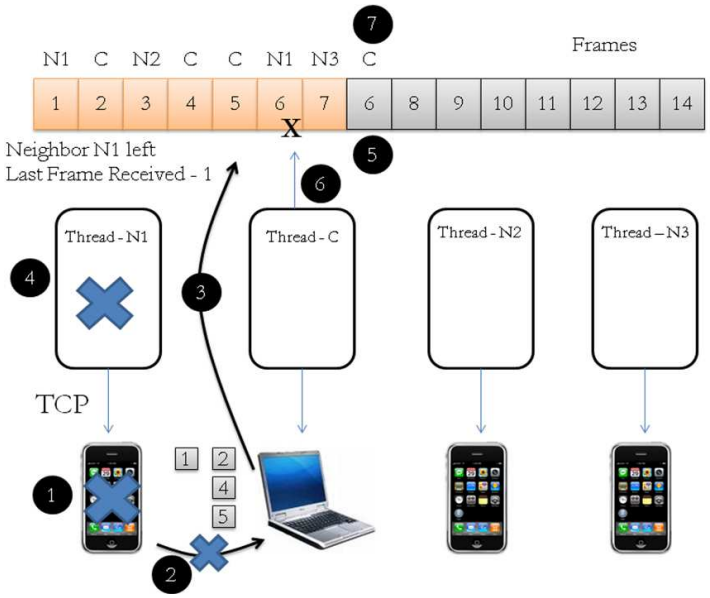

Fig. 6 Adapting frame distribution to Neighbor leaving example

4.3.2 Neighbor Leaving

Continuing the example from Figure 5, suppose Neighbor N1 decides to leave, the following sequence of events takes place (shown in Figure 6).

Step 1: Neighbor N1 leaves. This is signaled by Neighbor N1 not sending I-CAN-HELP messages for 3 consecutive seconds. Since N1 was mid-way in receiving frame 6, the Client never received it.

Step 2: The Client stops receiving I-CAN-HELP messages from the Neighbor. When the last update time in the neighbor table exceeds 3 seconds, the Client decides that the Neighbor has left. As explained

previously, for every Neighbor the Client maintains the last frame received. For N1, the last frame received is frame 1.

Step 3: Since the neighborhood has changed, the Client informs the Video Server about the change. The Client informs the server that Neighbor N1 has left. The Client also informs the server that the last frame received from that Neighbor is frame 1, so that the server can re-send any frames after frame 1 that it might have sent to N1.

Step 4: The Server kills the thread N1 so that no more frames are distributed to it.

Step 5: The Server has been keeping track of the distribution of frames to links. For example, the server keeps track that frame 1 and 6 were assigned to thread N1 in that order. Since the last frame received from the Neighbor is 1, the server knows it needs to redistribute the frames assigned to N1 by adding the frames to the head of the Frame Queue. Note that the system ensures that the queue is always sorted so that the frames are assigned in order. Here, the server inserts frame 6 into the queue.

Step 6: Assuming that the Client thread now has spare bandwidth, it requests another frame to send. The Server assigns frame 6 to the Client thread. Note that the Client may get duplicate frames in case a Neighbor intermittently leaves, rejoins and resumes transmission. These can be removed by the Video Player before playout.

Thus the system ensures that all the frames are delivered reliably and are never lost due to Neighbors leaving. This makes CStream adaptive to changes in the neighborhood.

4.4 Buffering and Playing

The Video Player in the Client plays the video as frames are being received, implementing a policy of stopping and waiting for late frames to arrive as opposed to discarding late frames. The Buffer Manager receives frames from the Server and through the Neighbor and stores them in the buffer for the Video Player to extract, decode and play. The buffering policy has the following features:

Initial Buffering: Before starting to play the video, the Video Player waits for some amount of frames to be initially buffered. In our implementation, the Video Player waits for two seconds of the video to be buffered before starting to play. That is, the Player waits for $2 \times f$ frames where f is the frame rate. The choice of two seconds worked well for our system to decrease the startup delay, while providing streaming without too many rebuffer events. The initial buffer length can be easily changed if future research provides indication of better values.

Stopping and Rebuffering: After the initial buffering, the Video Player plays the frames continuously as long as they are in the buffer. The Player plays the video at the appropriate frame rate, as reported in the meta-data when the Client first connected. When the frame to be played is not yet available in the buffer, the Video Player stops playing the video and triggers a rebuffer event. The Player then waits for the frame to arrive.

Playing after Rebuffering: During a rebuffer event, the Player stops and waits for the next two seconds of frames ($2 \times f$) to be received before starting the playout. Refilling the buffer attempts to reduce the total number of rebuffer events. If the total number of frames in the video is less than the current frame number $+ 2 \times fr$, it waits until all the frames are received before playing again.

To illustrate this with a numeric example, assume that the frame rate of a video is 15 frames per second and the total number of frames is 120. Before starting to play the video, the Player waits for the first 30 frames to be received (2 seconds of frames). Once they are received, the Player starts playout. Suppose the 45th frame is not available in the buffer when the Player is supposed to play it out. The Player then stops and triggers a rebuffer event. The Player waits until all the frames up to the 75th frame (next two seconds worth of videos frames) are received before restarting playout. Another rebuffer event at the 105th frame causes the Player to wait until the remaining frames, up to frame 120, are received before resuming playout.

Fig. 7 CStream user interface

5 Evaluation

In order to evaluate CStream, we build a complete system comprising the Video Server, Client and the Neighbor components. CStream is written in C# .NET with a code base of about 3000 lines. Source video files are stored in AVI format and CStream uses an open source AVI Video Library [oAV] to extract the frames from the source video files. Figure 7 shows a screen shot of the CStream Video Player. Users can request a video by filename using the textbox at the top of the video player. In addition to the video, the Player also shows the video status (*Buffering* or *Playing*), the number of Neighbors collaborating and their IP addresses. The Player also displays the aggregate throughput, the playout time and the number of rebuffer events.

5.1 Experimental Setup

The performance of CStream is evaluated in a controlled environment with a small test bed of computers. The bandwidth between the Video Server and the Client (and Neighbors) is controlled and the experiments are run for different bandwidth settings.

Figure 8 shows the experimental setup. The Video Server, Client and Neighbor machines are desktop PCs running Windows XP. All the machines have a Pentium 4, 2.8 GHz CPU with 1 GB RAM. The Video Server is connected to a PC running Linux acting as a bridge via a crossover cable and the Bridge uses class-based queuing (CBQ) for traffic shaping from Video Server. CBQ enables control of the bandwidth to specific destination IP addresses, allowing per flow control of the data rate from the Video Server to each Neighbor and the Client. The Bridge has a Pentium 4, 2.0 GHz CPU and 512 MB RAM running SuSE Linux 10.3 and has the prebuilt Netem module. The Client and the Neighbors connect to each other through a wireless ad-hoc network and all of the machines are on the Worcester Polytechnic Institute LAN.

5.1.1 Experimental Parameters

The performance of the CStream system is evaluated by varying the following parameters:

- *Number of Neighbor nodes:* The number of Neighbors is varied from zero to two.
- *Bandwidth of the nodes from the Video Server:* The available bandwidth on the link from the Video Server to the Client and the Neighbors is varied using CBQ for six different settings: 250 Kbps, 500 Kbps, 1 Mbps, 2 Mbps, 3 Mbps and 5 Mbps.
- *Video content:* Two different videos are used in the experiments, a short and a long video. The details of the videos are listed in Table 1.
- *Location of nodes:* The physical location of Neighbors with respect to the Client is changed to vary the wireless ad-hoc network connection quality and, hence, the bandwidth.

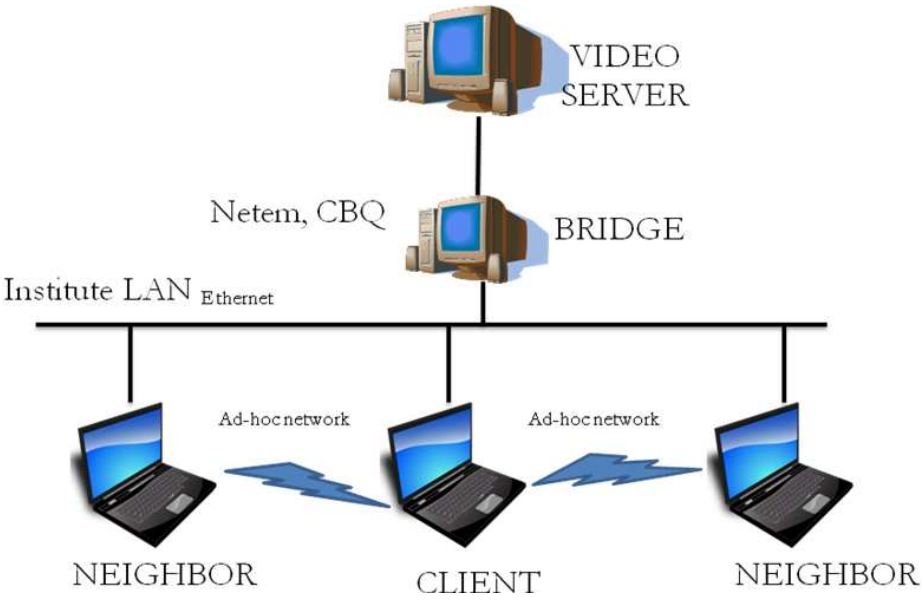

Fig. 8 Experimental setup

5.1.2 Performance Metrics

The Client records to a log the frame number, time of arrival, frame size and information about which link (Client or Neighbor) delivered the framed. Performance metrics are then calculated offline, after the experiments, using the log.

The performance of CStream is evaluated using the following metrics:

- *aggregate throughput (Kbps):* Aggregate throughput is the total application throughput at the Client, calculated as the ratio of the total size of the video downloaded to the total time taken to download all the frames.
- *playout time (sec):* The playout time is the total time taken to play the video, calculated as the sum of the startup delay, the time taken for rebuffer events and the playing time.
- *startup delay (sec):* The startup delay is the time taken to play the first frame in the video after the video request is sent to the Video Server.
- *rebuffer events:* The number of rebuffer events is the number of times the Video Player stopped to rebuffer frames after it first started playing.
- *Frame Distribution (%):* The frame distribution compares the ratio of the contribution of frames by the Client and the Neighbor compared to the ratio of their corresponding bandwidths.

	Short Video	Long Video
Content	Cartoon dog	Foreman
Length	8 seconds	33 seconds
Total frames	120	400
Frames per second	15	12
Resolution	320x240	176x144
Average frame size	85 Kbytes	68 Kbytes
Encoded bitrate	10 Mbps	6.3 Mbps

Table 1 Major features of videos used in testing

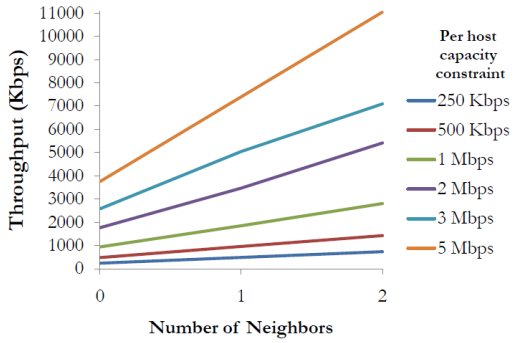

Fig. 9 Average aggregate throughput for short video

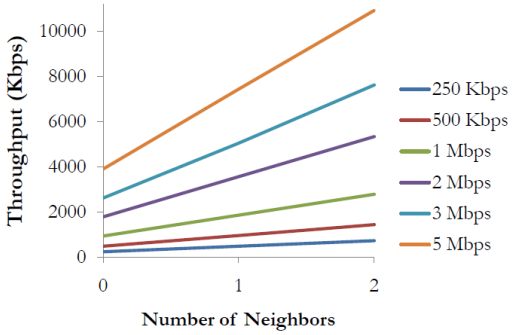

Fig. 10 Average aggregate throughput for long video

5.2 Results

Each video is run three times under each setting and all experiments present the average and standard deviation over thee three runs, unless otherwise noted. The bandwidth is set equally for the Client and the Neighbors for all the experiments, unless otherwise noted. Also, with the exception of the experiments studying the impact of wireless signal strength, the Neighbors are placed near to the Client so that they have excellent wireless signal strength. The minimum bandwidth of the wireless network for excellent signal strength measured using `iperf` is approximately 13 Mbps, significantly higher than the wired bandwidth.

5.2.1 Aggregate Throughput

The aggregate throughput observed at the Client both in the presence and absence of Neighbors is measured. For each bandwidth setting, the number of Neighbors is varied and the throughput computed. Figure 9 and Figure 10 show the throughput for each bandwidth setting with 0, 1 and 2 Neighbors for both the short and long videos, respectively.

Since the performance trends are the same for the short video as they are for the long video, for all subsequent analysis only data on the long video is shown.

To complement Figure 10, the aggregate throughput and exact ratio of the aggregate throughput with respect to the throughput with one client (no neighbors) is shown in Table 2. In general, the aggregate throughput at the Client increases linearly with an increase in Neighbors. At every bandwidth setting, the aggregate throughput approximately doubles that of just the Client alone when there is one Neighbor and it triples when there are two Neighbors. The standard deviations are not shown, but the largest standard error, for 2 Neighbors with 5 Mbps capacity, is only 12%.

Per Host	Aggregate Throughput (Multiple Increase)					
Bandwidth Constraint	0 Neighbors		1 Neighbor		2 Neighbors	
250	238	(1.00x)	483	(2.02x)	726	(3.05x)
500	476	(1.00x)	955	(2.00x)	1434	(3.00x)
1000	932	(1.00x)	1854	(1.99x)	2778	(2.98x)
2000	1777	(1.00x)	3558	(2.00x)	5338	(3.00x)
3000	2611	(1.00x)	5045	(1.93x)	7624	(2.91x)
5000	3902	(1.00x)	7437	(1.91x)	10925	(2.80x)

Table 2 Average aggregate throughput (Kbps) for long video, for a given per host bandwidth constraint and indicated number of neighbors. Numbers in parentheses represent the ratio of the achieved throughput with respect to the throughput with one client (no neighbors).

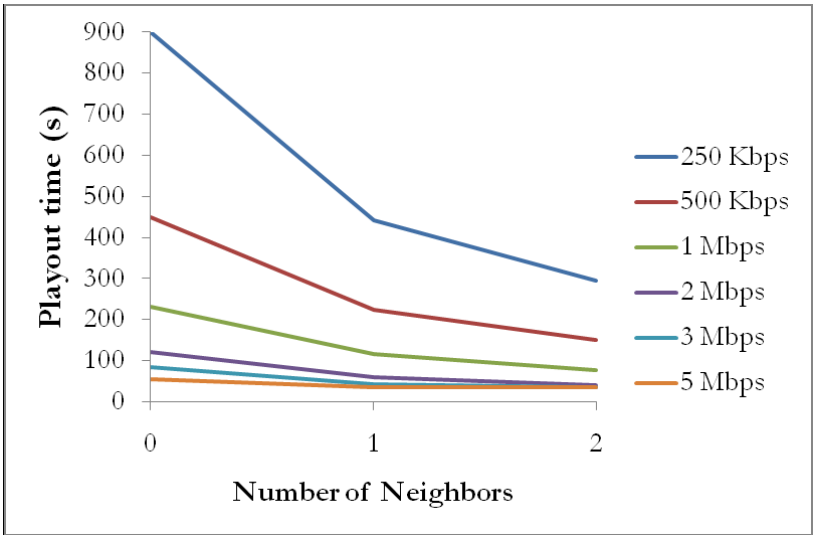

Fig. 11 Average playout time for long video

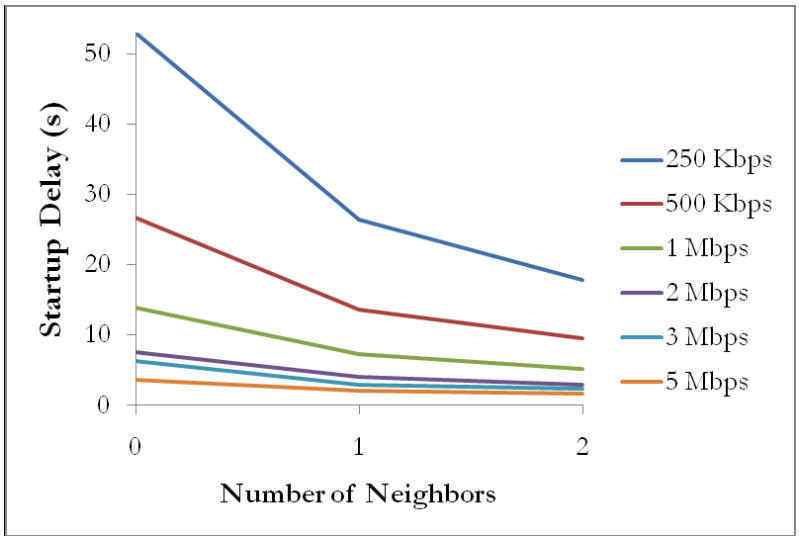

Fig. 12 Average startup delay time for long video

5.2.2 Playout Time

To quantify the quality of the video in different settings, playout time, startup delay and the number of rebuffer events are analyzed next.

Figure 11 shows the playout time for every bandwidth for the long video. For a given bandwidth setting, the playout time decreases multiplicatively with the increase in the number of neighbors. The largest standard deviation is 1.18 seconds (about 4%), when there are 0 neighbors and a 5 Mbps capacity.

5.2.3 Startup Delay

The startup delay is the time taken by the Video Player to play the first frame in the video after the video request is sent to the Video Server. In our implementation, the Video Player waits for two seconds of frames to arrive to fill the playout buffer before playing the first frame.

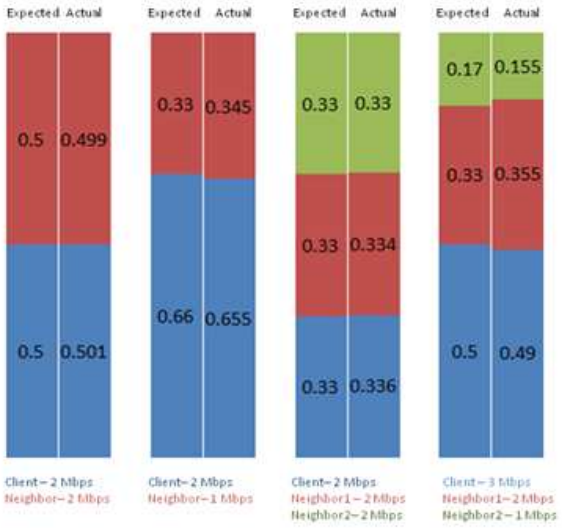

Fig. 13 Frame distribution versus ratio of bandwidth for long video

Figure 12 shows the average startup delay with the increase in number of neighbors for each bandwidth setting for the long video. Similarly to the playout time graph, there is a multiplicative improvement in the startup delay. The largest standard deviation is 1.81 seconds (about 30%), when there are 0 neighbors and a 3 Mbps capacity, but all other standard deviations were much less than 10%.

5.2.4 Rebuffer Events

Our final performance measure to assess the quality of the video is the number of rebuffer events.

Table 3 shows the average number of rebuffer events over three experimental runs for every bandwidth and Neighbor setting for the long video. The number of rebuffer events decreases as the number of neighbors increases for a given bandwidth setting. It can be noted that for the long video, either three 3 Mbps links or two 5 Mbps links are required to stream a video without any rebuffer events.

Per Host	Number of Neighbors		
Bandwidth Constraint	0	1	2
250 Kbps	15.0	14.3	14.0
500 Kbps	15.0	13.0	12.0
1 Mbps	13.0	11.0	9.0
2 Mbps	11.0	6.6	2.0
3 Mbps	9.0	2.6	0
5 Mbps	6.0	0	0

Table 3 Average number of rebuffering events

5.2.5 Frame Distribution

To study the effectiveness of the frame assignment scheme, the ratio of the frames received by the Client and the Neighbors during streaming is compared to their corresponding bandwidth settings. An ideal frame assignment scheme should distribute the frames proportionally based on the bandwidth of the links to minimize the total download time and hence achieve maximum throughput. For example, if there is one Neighbor with a bandwidth of 1 Mbps and the Client with a bandwidth of 2 Mbps, then ideally 2/3 of the video (frames) should be sent through the Client and 1/3 through the Neighbor to minimize the total download time.

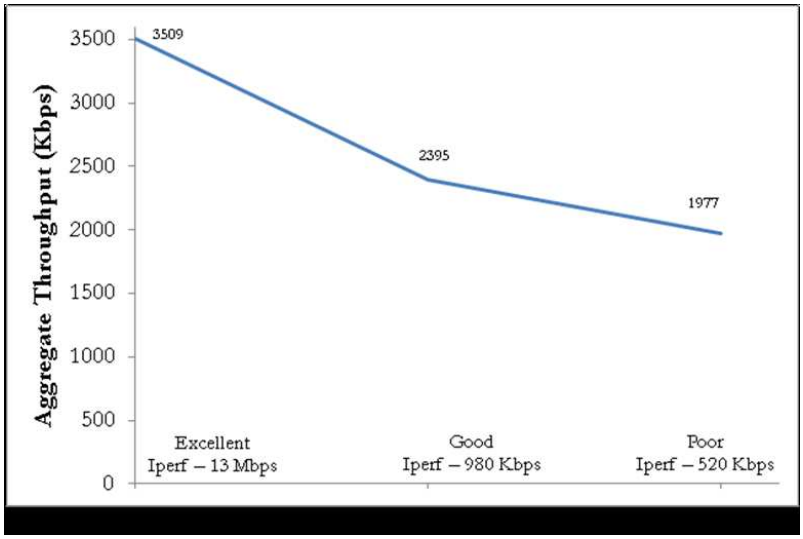

Fig. 14 Impact of wireless signal strength on aggregate throughput for long video (1 Neighbor with 2 Mbps capacity)

The total size of the frames downloaded by each node is compared to their ratio of the bandwidth settings. Figure 13 shows four different experiments with the long video.

In the first experiment, there was one Neighbor and the bandwidth of both the Neighbor and the Client is set to 2 Mbps. The graph shows the contribution of each node during streaming. As expected, both the nodes contributed around 50% of the total throughput, each. In the second experiment, the Client and the Neighbor have unequal bandwidth, with the Client set to to 2 Mbps and the Neighbor set to 1 Mbps. Again, in this scenario, the contribution of each node almost matches the ratio of their bandwidths. In the third and fourth experiments, the two Neighbors have equal and unequal bandwidths, respectively. In each experiment, the ratio of the video frames downloaded by each node matched the ratio of their bandwidths.

5.2.6 Impact of Wireless

In all the previous experiments, the wireless bandwidth was more than the wired bandwidth, and hence the overall throughput was limited by the aggregate wired bandwidths. The impact of wireless is studied by changing the location of the Neighbors with respect to the Client. In wireless, the throughput depends upon the signal strength among the nodes, which decreases as the distance between the nodes increases. For these experiments, the location of the Neighbor is changed from excellent, to good, and lastly to bad signal strength. The wireless condition was determined visually based on the number of bars in the "connect to a network" Windows dialog panel.

In our experiments, there was the Client and one Neighbor, both having a wired capacity of 2 Mbps. The long video is used for evaluating the Neighbor leaving and joining scenarios. Figure 14 shows the aggregate throughput obtained in each scenario, with the corresponding throughput of `iperf` shown along the horizontal access at the bottom. According to the `iperf` results, excellent good and poor received 13000 Kbps, 980 Kbps and 520 Kbps, respectively. Correspondingly, the aggregate throughput obtained by CStream drops as the distance between the nodes increases. Specifically, the bandwidth contributed by the Neighbor in the good and poor scenarios is constrained by the wireless bandwidth since the bandwidth contributed by the Neighbor in CStream is the minimum of the wired and wireless bandwidths.

5.2.7 Neighbors Joining and Leaving

Additionally, the impact in the overall throughput when Neighbors join and leave in the middle of a streaming session is studied. The measure of throughput for these experiments is the ratio of the total size of the previous 20 frames to the time taken to download them. 20 frames is about 2 seconds worth of

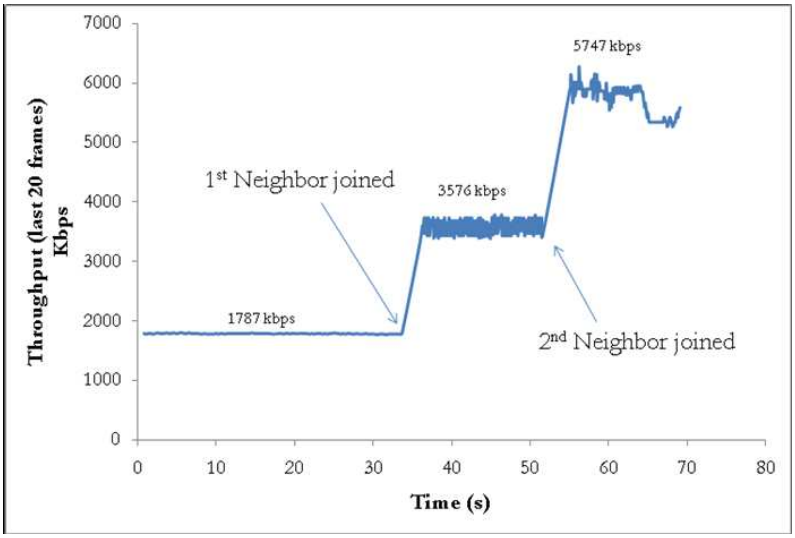

Fig. 15 Change in throughput when neighbors join

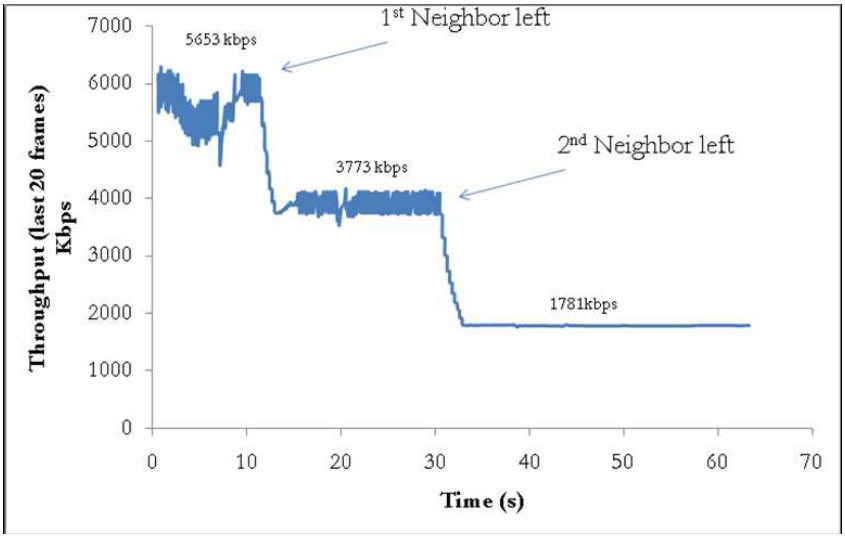

Fig. 16 Change in throughput when neighbors leave

video for the long video and about the size of the playout buffer. The throughput over time is measured at the Client as the neighborhood is changed.

Figure 15 shows a single experiment run where Client and the Neighbors had a bandwidth of 2 Mbps. The first Neighbor joined at around 35 seconds and the second Neighbor joined at around 55 seconds. It can be seen that the throughput almost doubled (from 1787 Kbps to 3576 Kbps) after the first Neighbor joined and almost tripled (up to 5747 Kbps) after the second Neighbor joined.

Figure 16 shows a similar experiment but with Neighbors leaving. To start there are 2 Neighbors all with 2 Mbps bandwidth. The first Neighbor leaves at around 12 seconds and the second Neighbor leaves at around 32 seconds. The throughput drops from an average of 5653 Kbps to 3773 Kbps when the first Neighbor leaves and to 1781 Kbps when the second Neighbor leaves.

Both the experiments show that CStream dynamically handles changing neighborhood and effectively uses the available bandwidth. This adaptation happens automatically, in mid-stream without any intervention by the user.

6 Conclusions

Despite its popularity, video streaming still remains a challenge in many scenarios. The reception of high-bandwidth video can often be limited by broadband bandwidths to the home or by 3G bandwidths to smart phones. Two observations can be made about many streaming environments: 1) Since Internet devices are often on and connected even when not in use, there is often a high density of idle Internet connections available in many neighborhoods. Although the available bandwidth at one Client is limited, the total unused bandwidth available in a neighborhood can be quite high if aggregated together. And 2), most of today's wireless devices have multiple interfaces that enable them to connect to nearby devices in an ad-hoc fashion at the same time they are connected to the Internet. CStream leverages the above two facts to aggregate unused Internet bandwidth in a neighborhood for better video streaming.

This paper presents the design, implementation and evaluation of the CStream prototype, a proof of concept implementation of a collaborative streaming system to improve video streaming in a neighborhood environment. CStream connects nearby nodes in an ad-hoc network to aggregate the bandwidth available at each node. CStream streams a single video through all the available links to improve the overall video quality. A fully-functional, adaptive, CStream system including the Video Server, Client and the Neighbor is designed and implemented.

CStream is evaluated on a small test bed of computers allowing for a controlled but realistic setup of wired and wireless networks. Aggregate throughput achieved by CStream is measured, varying the number of Neighbors participating in video streaming. Video quality is measured in terms of total playout time, startup delay and number of rebuffer events.

The results show that when the Client and the Neighbors have equal bandwidth, the aggregate throughput achieved with CStream increases linearly with the increase in the number of Neighbors participating in the streaming. The startup time and time to stream and play the entire video decreases multiplicatively as the number of neighbors increases, accompanied by a sharp decrease in the number of rebuffer events. The ratio of the frames contributed by all nodes (both Client and Neighbor) is proportional to the available bandwidth. Degraded bandwidth due to possibly poor signal strength from Neighbor to Client is handled automatically by CStream. CStream can even adapt mid-stream to Neighbors joining and leaving the CStream system.

The contributions of this work include:

- A novel system for video streaming that connects neighboring nodes in an ad-hoc network to aggregate Internet bandwidth.
- Design of a system that streams video through multiple Internet connections and that dynamically adapts to the changing neighborhood (nodes joining and leaving).
- Description of a frame distribution scheme that distributes video frames across multiple connections and makes full utilization of the available bandwidth.
- Implementation of a fully-functioning CStream system including Client, Neighbor and Video Server. The Client includes a Video Player through which users can request and play videos.
- Detailed performance evaluation over a range of video and network configurations, showing a linear improvement in throughput and video quality as the number of cooperating Neighbors increases.
- Contribution of the CStream code[3] for use by other researchers.

7 Future Work

A natural next step in evaluation is to measure the performance of CStream in a real-world setting. Real-world deployment, say in an apartment complex, would help study ISP diversity, bandwidth optimizations and their effect on CStream performance. It would also provide data on the density of Neighbor nodes, their wireless signal strengths and Neighbor idle times.

Setting the TCP buffer size to approximately one frame limits problems with Neighbors with extremely low-bandwidth links or Neighbors leaving. However, since TCP throughput is dependent upon buffer sizes, an optimal buffer size may need to consider round-trip time, in addition to video delay, in choosing the "best" size. In addition, while the current CStream approach of sending a frame down a link when there

[3] CStream is available for download at http://perform.wpi.edu/downloads/#cstream

is available buffer capacity implicitly accommodates disparities in bandwidth among the links, additional considerations could consider variations in bandwidth or possibly the round-trip time of each link, too.

While CStream was implemented and evaluated on PCs, a possible extension is to use CStream to aggregate 3G bandwidth for smart phones and measure how it improves video streaming performance. The CStream Video Server could use the same code developed in this work, but the Client and Neighbor code would need to be ported to a smart phone. Alternatively, to just evaluate the implications of 3G performance, 3G modem cards could be deployed on laptops running the current CStream implementation.

The source videos were stored in AVI format in our CStream implementation to ease the implementation effort. CStream can be extended to support other video types with better compression, using the video properties to adjust the streaming plan. For example, for the I, P and B frames in MPEG the CStream plan may send the more important I frames directly to the client and send the P and B frames to the Neighbors.

In addition, the current implementation of CStream does not scale the video when the combined bandwidth of the participating Neighbors and the Client is not sufficient to support the streaming rate. CStream could be extended to include different temporal, quality or spatial scaling when there is not enough bandwidth to stream the video.

CStream assumes that all the Neighbors (and Clients) are trustworthy, so there is no focus on security or privacy. Thus, an area of future work is to examine the security issues the system needs to address. This could include the Video Server encrypting and signing video frames and the Client verifying the content to prevent man-in-the-middle attacks. There might be a need for a Client to maintain a trusted set of Neighbors as well as the ability to blacklist malicious Neighbors to control denial of service attacks.

Similarly it would be useful to come up with an incentive model to make CStream more practical and deployable rather than just relying upon the good will of Neighbors. A simple solution for incentives is to implement a tit-for-tat (TFT) based scheme as do streaming systems such as BitTorrent [LRLZ08]. In a TFT scheme, a node gains credit when it sends data to other peers and spends the credit to receive data from other peers. A more complex solution is to design a micropayment-based scheme similar to COMBINE [APRT07] to incentivize nodes to help for a fee. The payment scheme COMBINE includes a signed node of credit, termed an IOU indicating the amount of payment made. Providing proper incentives becomes more complicated when multiple Clients compete for resources donated by Neighbors, warranting additional study.

References

[APRT07] G. Ananthanarayanan, V. Padmanabhan, L. Ravindranath, and C. Thekkath. COMBINE: Leveraging the Power of Wireless Peers through Collaborative Downloading. In *Proceedings of ACM Mobisys*, San Juan, Puerto Rico, June 2007.

[AWW05] I. F. Akyildiz, X. Wang, and W. Wang. Wireless Mesh Networks: A Survey. *Elsevier Journal of Computer Networks*, 47(4):445–487, 2005.

[CBB04] R. Chandra, V. Bahl, and P. Bahl. Multinet: Connecting to Multiple IEEE 802.11 Networks using a Single Wireless Card. In *Proceedings of IEEE Infocom*, Hong Kong, March 2004.

[CR06] K. Chebrolu and R. Rao. Bandwidth Aggregation for Real Time Applications in Heterogeneous Wireless Networks. *IEEE Transactions on Mobile Computing*, 5(4):388 – 403, April 2006.

[(Ed07] Randall Stewart (Editor). Stream Control Transmission Protocol. *IETF Request for Comments (RFC) 4960*, September 2007.

[EKG+11] Kristian Evensen, Dominik Kaspar, Carsten Griwodz, Pl Halvorsen, Audun Hansen, and Paal Engelstad. Improving the Performance of Quality-Adaptive Video Streaming over Multiple Heterogeneous Access Networks. In *Proceedings of the Second ACM Multimedia Systems Conference (MMSys)*, pages 57 – 68, February 2011.

[FNI04] J. Funasaka, K. Nagayasu, and K. Ishida. Improvements on Block Size Control Method for Adaptive Parallel Downloading. In *Proceedings of the International Workshops on Distributed Computing Systems*, pages 648 – 653, Washington D.C., USA, March 2004.

[GALM07] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View From the Edge. In *In Proceedings of the ACM Internet Measurement Conference (IMC)*, San Diego, CA, USA, October 2007.

[JJK+08] S. Jakubczak, M. Jennings, M. Kaminsky, K. Papagiannaki, and S. Seshan. Link-alike: Using Wireless to Share Networking Resources in a Neighborhood. *ACM SIGMOBILE Mobile Computing and Communications Review*, 12(4), October 2008.

[KSCK10] Rabin Karki, Thangam Seenivasan, Mark Claypool, and Robert Kinicki. Performance Analysis of Home Streaming Video Using Orb. In *Proceedings of the 18th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Amsterdam, The Netherlands, June 2010.

[LRLZ08] J. Liu, S. Rao, B. Li, and H. Zhang. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. *Proceedings of the IEEE*, 96(1):11–24, January 2008.

[oAV] A Simple C# Wrapper for the AviFile Library. [Online] http://www.codeproject.com/KB/audio-video/avifilewrapper.aspx.

[oEE09] The Exabyte Era. [Online] http://www.cisco.com/web/IN/about/network/the_exabyte_era.html, September 2009.

[PWB+08] J. A. Pouwelse, P. Garbacki J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. van Steen, and H. J. Sips. TRIBLER: a Social-based peer-to-peer System. *Journal of Concurrency and Computation: Practice & Experience - Recent Advances in Peer-to-Peer Systems and Security*, (2), February 2008.

[RKB00] P. Rodriguez, A. Kripa, and E. W. Biersack. Parallel-access for Mirror Sites in the Internet. In *Proceedings of IEEE Infocom*, Tel Aviv, Israel, March 2000.

[SKK08] T. Badirkhanli S. Kandula, K. Lin and D. Katabi. FatVAP: Aggregating AP Backhaul Bandwidth. In *Proceedings of Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, April 2008.