2010-05-26

CStream: Neighborhood Bandwidth Aggregation For Better Video Streaming

Thangam Vedagiri Seenivasan
Worcester Polytechnic Institute

CStream: Neighborhood Bandwidth Aggregation For Better Video Streaming

by

Thangam Vedagiri Seenivasan

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

_____

May 2010

APPROVED:

_____

Professor Mark Claypool, Thesis Advisor

_____

Professor Robert Kinicki, Thesis Reader

_____

Professor Michael Gennert, Head of Department

Abstract

Video streaming is an increasingly popular Internet application. However, despite its popularity, real-time video streaming still remains a challenge in many scenarios. Limited home broadband bandwidth and mobile phone 3G bandwidth means many users stream videos at low quality and compromise on their user experience. To overcome this problem, we propose CStream, a system that aggregates bandwidth from multiple co-operating users in a neighborhood environment for better video streaming. CStream exploits the fact that wireless devices have multiple network interfaces and connects co-operating users with a wireless ad-hoc network to aggregate their unused downlink Internet bandwidth to improve video quality. CStream dynamically generates a streaming plan to stream a single video using multiple connections and continuously adapts to changes in the neighborhood and variations in the available bandwidth. We have built CStream and evaluated it on a controlled test-bed of computers with various performance measures. The results show linear increase in throughput and improved video streaming quality as the number of cooperating users in a neighborhood increase.

Contents

1 Introduction

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

The quality of video streaming is mainly dependent on the downlink Internet bandwidth available to the end user. Although Websites 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 [oORB] 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. Additionally, video streaming in mobile devices, like smart phones, will be an important application in the future. The 3G Internet bandwidth [oWiki3G] offered in such devices is expected to take a long time to meet the demand for real-time streaming. Hence, despite the popularity, real-time video streaming that provides a good user experience still remains a challenge in many scenarios with the Internet downlink bandwidth as the major bottleneck.

To overcome this problem and enable users to stream quality videos in real-time, we propose CStream (Collaborative Streaming), a system that aggregates bandwidth from multiple co-operating users in a neighborhood. The motivation for the system stems from the fact that although individual users have limited bandwidth, the high density of Internet users in a neighborhood with only a small percentage active at any given 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 co-operating users and avail their Internet connection 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 the layered encoding of videos and streams 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 business 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 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 where in a home community, users connect to their ISPs using laptops. CStream connects these laptops in a wireless ad-hoc network to aggregate bandwidth for video streaming. Consider another scenario, an airport where there are many users with mobile smart phones. CStream connects these phones together through an ad-hoc wireless network to aggregate their 3G bandwidth for multimedia streaming.

The main components of the CStream system are the Video Server, the CStream Client (running 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 creates an ad-hoc network asking if any node in a neighborhood is willing to contribute bandwidth. 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 (Client and Neighbors). The Neighbors act as a proxy sending the frames received from the server to the Client through the ad-hoc network, thereby contributing additional bandwidth.

Figure 1.1 shows an example scenario of 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 connection. 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 the three links. The server sends frames through different 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 the three links and adapts dynamically to any change in bandwidth. The system also dynamically adapts to change in neighborhood. For example, when a new Neighbor joins the ad-hoc network, the system quickly uses the bandwidth of that Neighbor to improve streaming. Similarly, when a Neighbor

leaves in the middle of a streaming session, the system recovers and streams the lost frames through the active links.

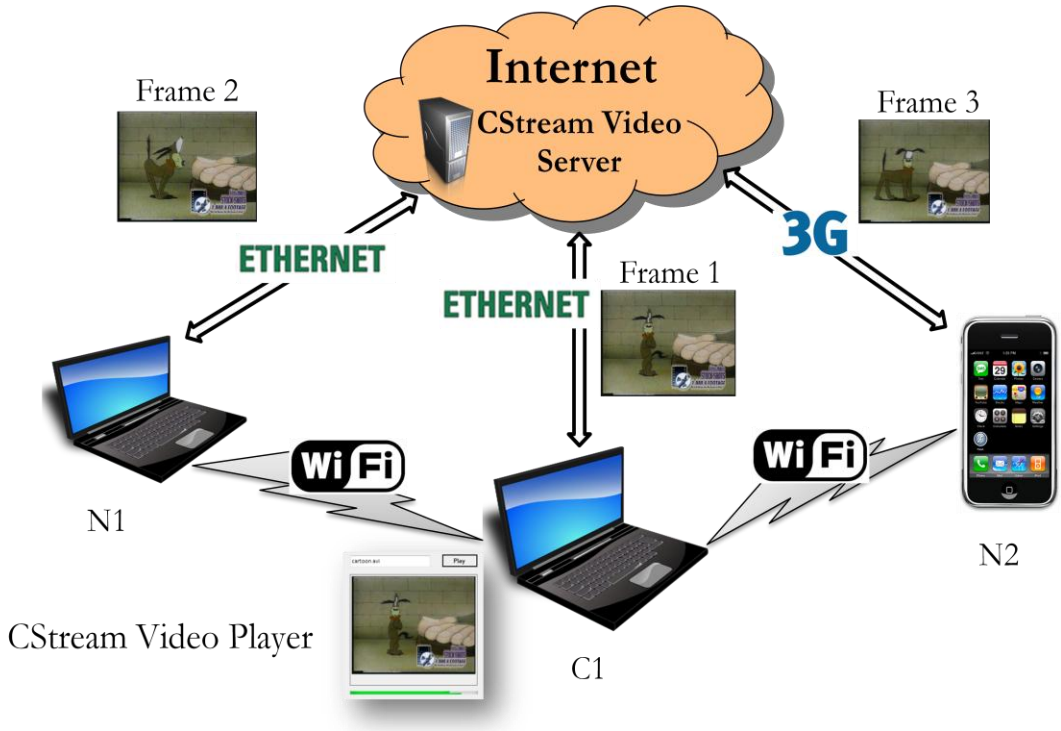

Figure 1.1: Aggregating bandwidth from neighboring nodes for video streaming

We build CStream and evaluate it using a four -node test-bed comprising a Video Server, a Client and two Neighbors. Testing CStream in a controlled environment allowed us to control the bandwidth of the Client and Neighbors to the Video Server. This helps us evaluate CStream for various bandwidth settings. CStream is evaluated varying the number of Neighbors, location of the Neighbor nodes (to vary the wireless throughput) and video content. We test the adaptiveness of CStream under dynamic conditions when Neighbors join and leave in the middle of video streaming. CStream System performance is evaluated using metrics such as aggregate throughput and video quality. To assess the video

quality, we measure the playout time, start-up delay and re-buffer events during streaming. Effectiveness of the streaming protocol is evaluated by measuring the server's ability to effectively and proportionally use the bandwidth of the available links.

We find 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 increase, 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 note that the ratio of the frames distributed by the server across multiple links perfectly matches their ratio of bandwidths.

The contributions of this thesis 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 and that dynamically adapts 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 video streaming system over a range of video and network configurations that shows linear improvement in throughput and video quality as the number of co-operating users increases.

The rest of the thesis is organized as follows. In Chapter 2, we explore related work and compare and contrast with CStream. Chapter 3 discusses the design and architecture of CStream. First, we list the challenges in building such a system and explain how the design meets the challenges. CStream design explains in detail the components of CStream, the Video Server, the Client and the Neighbor and discusses how the system works. Chapter 4 explains the implementation details of the CStream. We present the details of neighbor discovery and maintenance, frame distribution, dynamic plan handling and the buffering policy. Chapter 5 explains our experimental setup and presents detailed evaluation based on various performance measures such as aggregate throughput, video quality and frame distribution. CStream is evaluated varying the bandwidth, the number of Neighbors, the location of the Neighbors and video content. We present future work in Chapter 6 and conclusions in Chapter 7.

2 Related Work

There has been considerable research on improving Internet bandwidth and application throughput in recent years. Initially, the focus was on effectively using the available bandwidth at a single network interface (for example, using download accelerators). Later, with the proliferation of multi-homed devices, the focus shifted 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. This chapter describes some related work in these areas in detail and discusses how CStream differs from each of them.

2.1 Download Accelerators

Download accelerators and peer-to-peer (P2P) systems improve file download speed by getting parts of the file over multiple connections. Download accelerators [RKB00] 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 a single bad server selection may severely impact the download performance, while downloading in parallel from multiple servers reduces the impact of a bad server selection.

In [RKB00], the authors 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. They built a prototype of the dynamic parallel-access scheme as a JAVA client that takes the URL of the mirror servers as an input parameter. They 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. When all the servers used in the experiment have similar performance, then the speedup gain is very large. When the performances of the different servers are mismatched then the resulting speedup is not as significant when compared to the fastest server's performance. Even in this case, the parallel-access scheme achieves response times as low as the ones provided by the fastest server alone and at the same time eliminating the critical decision of server selection.

P2P networks can improve video stream rate by streaming different parts of a video from multiple nodes [LRLZ06]. In P2P networks, a client connects to multiple peers and parts of the file are downloaded from different peers. Similar to accessing multiple servers, P2P networks provide link diversity thereby improving the download performance.

[LRLZ06] reviews the state-of-the-art of peer-to-peer Internet video broadcast technologies. 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-off and potential for large-scale deployment. In a

tree-based approach, peers are organized into structures (typically trees) for delivering data, with each data packet being disseminated using the same structure. Data-driven overlay designs contrast to tree-based designs in that they do not construct and maintain an explicit structure for delivering data. Instead they use the availability of data to guide the data flow.

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. In CStream, we consider the scenario where the user Internet downlink is the bottleneck and increase its application bandwidth through multiple connections. CStream can achieve more than the user downlink bandwidth since the bandwidth of the neighbors is aggregated to improve video streaming. For CStream the bandwidth gain is limited by the number of collaborating neighbors and not the client downlink capacity as in download accelerators and P2P systems.

2.2 Network Sharing

Network sharing is a well-explored field. Wireless Mesh Network (WMN) [AWW05] provides connectivity to users in a neighborhood which do not have direct Internet access. 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 consists of mesh routers, mesh clients and gateways. Mesh routers are dedicated nodes that operate in ad-hoc mode, usually stationary to support meshing. Gateways are nodes that have an Internet connection. Mesh clients can be stationary or mobile and communicate peer to peer with the mesh routers and gateways. There are three types of mesh

networks: infrastructure supported mesh networks, client wireless mesh networks and hybrid mesh networks. In infrastructure mesh networks, mesh routers provide the infrastructure for the clients. In client mesh networks, client nodes form a peer-to-peer network for extending Internet connectivity and perform actual routing. Hybrid meshing is a combination of infrastructure and client meshing. [AWW05] presents a detailed study on advances and challenges in wireless mesh networks. Wireless mesh networks are cost effective way of increasing Internet connectivity. They are self organizing, self healing and self configuring. WMNs provide redundancy and involve multi-hop routing to transfer data to a gateway and back to a node. WMN protocols assign one gateway per flow and routes all the packets in a flow through the same path. WMN has scalability issues and suffers in performance as many nodes join the mesh network. 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. CStream also splits a single video stream across multiple Internet connections.

2.3 Bandwidth Aggregation

There have been several research efforts recently on aggregating Internet bandwidth from multiple connections. Bandwidth aggregation techniques seamlessly use multiple Internet connections as if it is a fat connection. The system presented by Chebrolu [CR06] assumes multiple Internet connections on the same device through multiple interfaces and aggregates bandwidth across these interfaces for video streaming. An important aspect of an architecture that does bandwidth aggregation for real-time applications is the scheduling algorithm that partitions the traffic onto different interfaces such that the QoS requirements

of the application are met. [CR06] proposes Earliest Delivery Path First (EDPF), an algorithm that ensures packets meet their playback deadlines by scheduling packets based on the estimated delivery time of the packets. They show through analysis that EDPF performs close to an idealized Aggregated Single Link discipline, where the multiple interfaces are replaced by a single interface with the same aggregated bandwidth. Using a prototype implementation and simulations carried using video traces, they show performance improvement with EDPF scheduling over using just the highest bandwidth interface and other scheduling approaches based on weighted round robin.

The number of network interfaces per device is limited (usually two) and the maximum capacity these systems 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, it aggregates Internet bandwidth from multiple neighbors. Hence, CStream can achieve greater capacity than such multi-homed systems, only limited by the wireless capacity (up to 600 Mbps in IEEE 802.11n [80211N]).

Systems discussed thus far attempt to improve throughput by parallel access to mirrored servers or simultaneous usage of multiple network interfaces. Recent research has focused on exploiting bandwidth at nearby nodes to improve application performance. COMBINE [APRT07] is a research prototype which aggregates bandwidth from multiple nearby phones by forming a wireless ad-hoc network between the nodes. COMBINE aggregates the 3G bandwidth on phones to download large files using HTTP. It improves HTTP download by getting different chunks of a file through different links. COMBINE uses an adaptive workload distribution algorithm to farm out work across the participants in the collaboration group. COMBINE uses HTTP byte-range requests for parallel

downloads and hence does not require server support. The main focus of COMBINE is an incentive system that is based on battery energy cost. An accounting system pays and bills users based on their bandwidth contribution. COMBINE includes an energy-efficient protocol for nodes to discover each other, exchange their bids, and form a collaboration group. The authors have prototyped COMBINE on Windows XP and evaluated its performance on laptop-class devices equipped with 802.11b WLAN NICs and GPRS WWAN modems. They show near-linear speedups for group sizes of up to five nodes.

CStream is similar to COMBINE in forming an ad-hoc network with neighbor nodes and aggregating bandwidth from multiple neighbors. CStream applies bandwidth aggregation across multiple nodes to effectively improve video streaming performance. Video streaming is a high bandwidth application and unlike HTTP download, has real time constraints where the bandwidth gain from collaborating with neighbors can boost video performance tremendously.

Link-alike [JJKPS08] is similar to COMBINE but is used to improve the upload capacity of the client. Many photo, video sharing services and online backup services require users to upload large amounts of data to their Websites, but the asymmetric broadband connections prevalent in residential network pose a challenge to users who want to publish information. Link-alike tries to solve this problem by increasing the upstream capacity of the client by aggregating the uplink capacities of the nodes in neighborhood. Link-alike addresses the challenges of operating in an environment that is highly lossy, broadcast in nature and half-duplex. Link-alike uses opportunistic wireless reception, a novel wireless broadcast rate control scheme, and preferential use of the wired downlink. Through analytical and experimental evaluation, [JJKPS08] demonstrates that Link-alike provides significantly better throughput than previous solutions based

on TCP or UDP unicast. CStream also aggregates bandwidth in a neighborhood environment, but unlike Link-alike CStream is focused on aggregating downlink bandwidth for video streaming.

Since systems like COMBINE and Link-alike support only file transfer, they do not have any constraint on the upload or the download time, and hence 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.

2.4 Virtual Interfaces

Although CStream assumes multiple interfaces at a device, it is not a requirement. Technologies like Multinet (Virtual WiFi) [CBB04] make a single wireless interface act as multiple network interfaces enabling them to connect to multiple nodes at the same time. For example, a node can connect to the Internet through an access point at the same time as it can connect to a neighboring node, both connections using the same wireless card. Multinet continuously switches a single wireless card across multiple networks. The system is transparent to the user and is agnostic to the upper layer protocols. Multinet is implemented as a Windows driver and virtualizes a single wireless card into multiple interfaces.

FatVAP [KLBK08] is a system which enables a single wireless card to connect to multiple access points at the same time using similar techniques used in Multinet. FatVAP is implemented as an 802.11 Linux driver that aggregates the bandwidth available at accessible APs and also balances their loads. FatVAP chooses the APs that are worth connecting to and connects with each AP just

long enough to collect its available bandwidth. It ensures fast switching between APs without losing queued packets. FatVAP works with unmodified APs and is transparent to applications and the rest of the network stack. The authors evaluate FatVAP both in a lab, at hotspots and for residential deployments. FatVAP delivers a median throughput gain of 2.6x, and reduces the median response time by 2.8x.

Using approaches like Multinet or FatVAP, CStream can be effective for devices with only one network interface. In our current implementation, devices have multiple interfaces but using technologies like Multinet and FatVAP CStream can work on devices with single network interface.

2.5 Summary

To summarize, CStream differs from all the previous systems in application and design. To the best of our knowledge, CStream is the first system to focus on aggregating bandwidth from nearby nodes for video streaming. It streams a single video through multiple connections and dynamically adapts to changing neighborhood. We design and implement CStream and present detailed performance evaluation under various settings.

3 Design

This chapter presents the design details of the CStream. CStream is a proof of concept system to show the benefits of bandwidth aggregation in a neighborhood to improve the performance of a video streaming. There are several challenges in building a system that aggregates bandwidth from nearby nodes through an ad-hoc wireless network to improve video streaming. We list and explain the challenges below and show how our design addresses these challenges.

Currently CStream does not focus on security and incentive model. We assume that neighbors having spare bandwidth are willing to participate in CStream without any incentives. Also we assume that all the neighbor nodes collaborating in video streaming are trusted and do not deal with security issues. We assume that the Client do not use CStream to download any illegal content. Finally, we assume users of both Client and Neighbors have installed the CStream software.

3.1 Challenges

3.1.1 Neighbor Discovery and Maintenance

The Client and Neighbors should find each other and connect through an ad-hoc wireless network. Once connected, at any time the Client should have an updated knowledge of the active Neighbors willing to contribute bandwidth. Neighbors should be able to join and leave the system at any time during video streaming.

3.1.2 Multi-path streaming

The Video Server should be able to stream a single video through multiple nodes. The server needs to know the Client and Neighbor end points to be able to send the frames. The server needs to distribute the frames to be sent based on the bandwidth available at each link and it should dynamically adapt to the changes in the bandwidth. An ideal distribution protocol will fully utilize the available bandwidth and proportionally distribute frames based on the ratios of the available bandwidths.

3.1.3 Dynamically changing neighborhood

The system should adapt to a change in the network neighborhood. Neighbors can then join and leave at any time during the video streaming. When new neighbors join, the system should be able to adapt quickly and start using the newly available bandwidth. When a Neighbor leaves, the system should be adaptive and recover the lost frames and redistribute them across the remaining active links.

3.1.4 Buffering and Playing

The Client should be able to buffer the frames received through different links and play them. The underlying mechanism of receiving frames from different links and playing them should be opaque to the user. The system should have an appropriate buffering mechanism which decides whether to wait for the late frames or discard them.

3.2 Architecture

The CStream design addresses all the above challenges. Figure 3.1 shows the architecture of CStream. The system has three distinct components: the Video Server, the Client node which requests the video and the Neighbors which help to aggregate bandwidth for better video quality. CStream assumes that multiple interfaces are present in the Client and Neighbors so that they can form wireless ad-hoc network wireless communications while being connected to the Internet.

In brief, CStream works as follows. The CStream Client forms a wireless ad-hoc network with Neighbors which are idle so that it 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. A participating Neighbor simply acts as a proxy in delivering the frames from the server to the Client. We explain the role of each component in greater detail below. We defer the implementation details to Chapter 4.

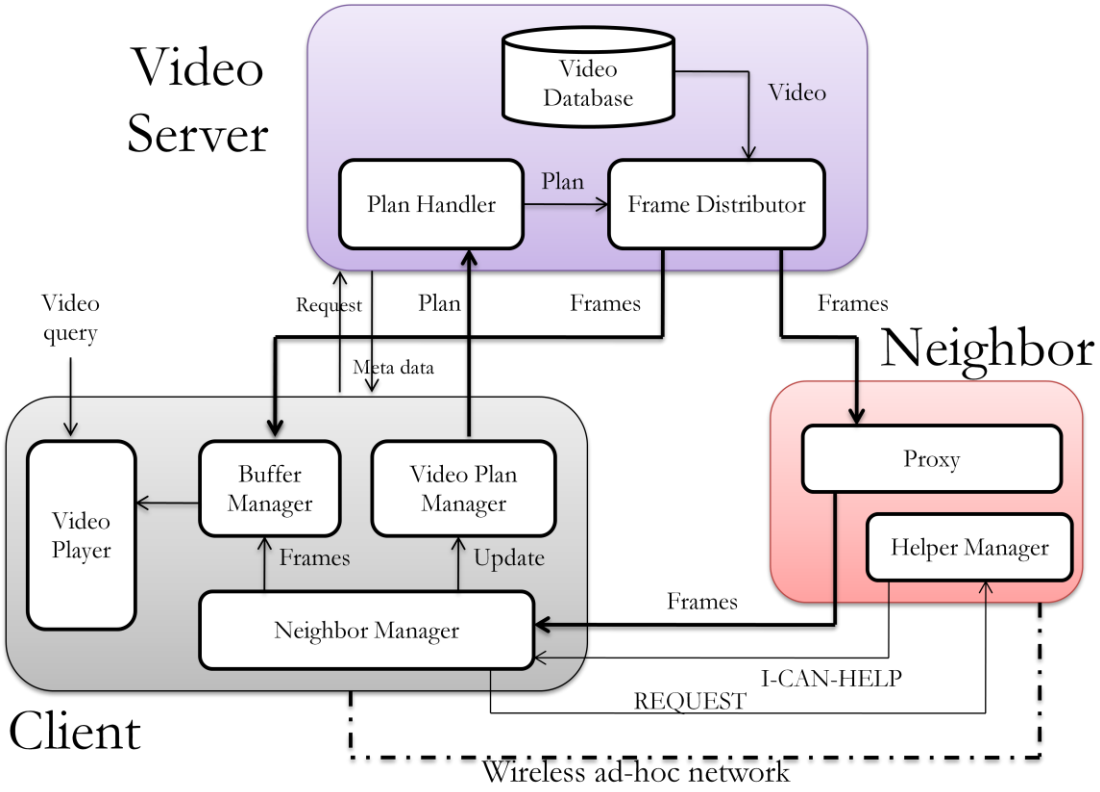

Figure 3.1: CStream architecture

3.2.1 Client

Users request video using the Video Player in the Client. The Client then sends the request to the Video Server. The Video Server replies back with the meta-data of the video which consists 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 the available links 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 and the role of each component is explained in detail below.

3.2.1.1 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 the ad-hoc network.
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 the system. The Neighbor Manager monitors for periodic heartbeat messages (I-CAN-HELP messages) from the Neighbors and constantly 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 it to the Buffer Manager. It keeps track of the frames received from each of the Neighbors and informs the Video Plan Manager.

3.2.1.2 Video Plan Manger

The Video Plan Manager is a core component of the Client that constantly informs the Video Server about the streaming plan. 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 to the Client.

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 Video Server can recover lost frames and stream through other active links.

3.2.1.3 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 the Neighbor through the Neighbor Manager and stores them in the buffer.

3.2.1.4 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 (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 the next chapter.

3.2.2 Neighbor

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

3.2.2.1 Helper Manager

The role of the Helper Manager is summarized below:

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.

3.2.2.2 Proxy

The Proxy component in the Neighbor functions as below:

1. The Proxy keeps a port open for the Video Server to stream video through it.
2. The Proxy receives frames from the Video Server and forwards it to the Client through the ad-hoc network. Neighbors do not buffer the received frames and immediately forward the frames to the Client.

3.2.3 Video Server

The Video Server stores the videos that Client can request. The Video Server streams a single video through multiple links. It adapts the streaming to changes in the network neighborhood. The role of each of the Video Server components is explained below.

3.2.3.1 Video Database

The Video Database stores the uploaded videos that Client nodes can stream:

1. The videos are encoded and stored in AVI format.
2. The Video Database splits a video into frames 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.

3.2.3.2 Plan Handler

The role of the Plan Handler is summarized below:

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. When the Client informs the Plan Handler about a new Neighbor, it updates the Frame Distributor to include the new Neighbor in the streaming process. When the Client informs the Plan Handler about a Neighbor that left in the middle of streaming, it updates the Frame Distributor to stop sending frames via that Neighbor.

Chapter 4 explains in detail how the system adapts to changes in neighborhood.

3.2.3.3 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 Client and the Neighbor.
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.

Chapter 4 explains in detail how the Frame Distributor effectively utilizes the bandwidth of the available links.

4 Implementation

This chapter presents implementation details of the protocols in the CStream system. Specifically, it discusses four areas: 1. Neighbor Management, 2. Frame Distribution, 3. Adapting to changes in neighborhood, 4. Buffering and Playing. We show illustrative examples to explain each of these areas.

4.1 Neighbor Management

This section explains the ad-hoc network formation and neighbor management. Specifically we explain the implementation details of Neighbor Manager in the Client and Helper Manager in the Neighbor and their interaction. An illustrative example to explain the sequence of flow in the implementation of CStream System is included.

Figure 4.1 shows an example scenario with a Client and two Neighbors. Assume that a user requests a video using the Client. We describe the sequence of events step by step using Figure 4.1. In the figure, the steps are show in black circles with the step number.

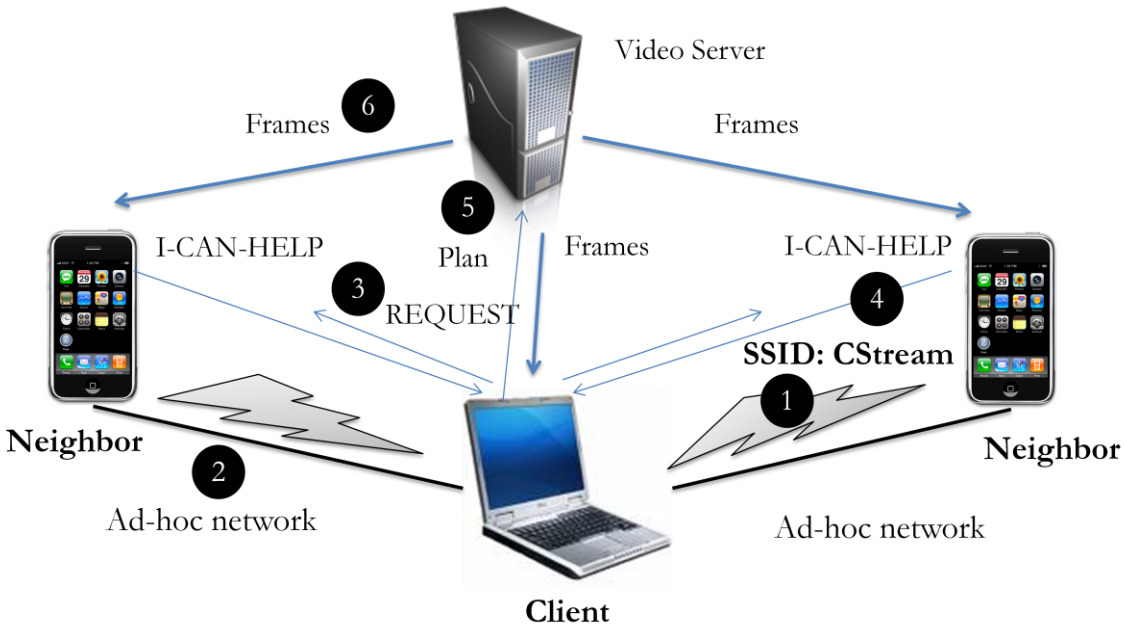

Figure 4.1: Ad-hoc network formation and neighbor management

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 even if a video is not requested. When it has user activity, the Neighbor disconnects from the ad-hoc network and connects back when it becomes idle again. This way of maintaining the ad-hoc network helps reduce the streaming start-up delay of forming the neighborhood network.

Step 2: Neighbors which are nearby the Client are in the range of 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 that it needs to stream a video and is looking for neighbors to contribute bandwidth. In our implementation, we use IP broadcast in the ad-hoc network 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 Neighbors 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 alive.

The Neighbor Manager at the Client maintains a neighbor table with the following information using the I-CAN-HELP messages.

Neighbor	*IP address*	*Port*	*Last Update Time*	*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. As detailed in subsequent sections, the Last Received Frame value will be used to recover lost frames when a Neighbor leaves. The Client decides that a Neighbor has left if it does not receive I-CAN-HELP messages for a specific amount of time (3 seconds in our implementation).

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 (last update time is less than 3 seconds).The streaming plan changes to capture the 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. The frame distribution protocol is explained in the 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. The Frame Distributor splits the video into frames and sends each frame through a single link. The sequence of flow at the server after the Client sends the initial streaming plan is illustrated with an example in Figure 4.2. 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. Here Neighbor N1 and N2 sends 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.

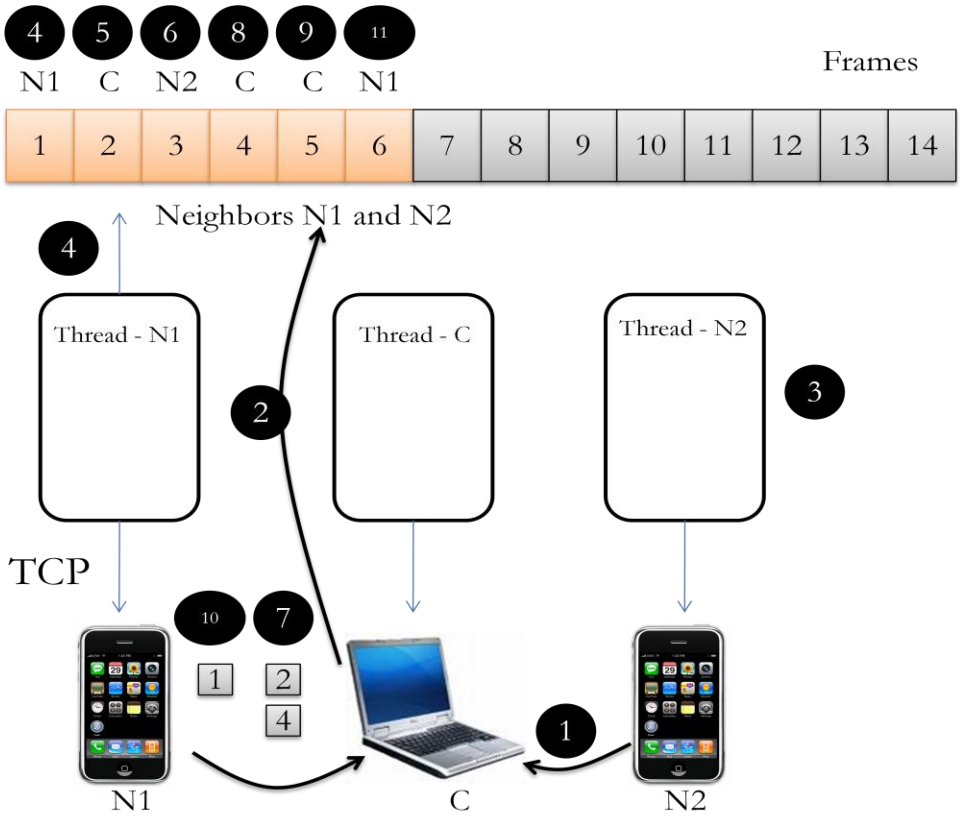

Figure 4.2: Frame Distribution

Step 3: Now the server has three links for streaming video. The server creates three threads to simultaneously send frames through the three links. Each thread services one node. The job of each thread is to fetch frames and send it to the node.

The server keeps all the frames in a Frame Queue and runs a simple frame assignment process. 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. We rely on TCP to adapt and effectively utilize 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 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: Suppose the Client has three times more bandwidth compared to the neighbors. Frame 2 gets sent faster than frame 1 and frame 3.

Step 8: Since there is spare bandwidth in the Client link, thread C requests for 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 finish sending frame 1 to Neighbor N2, N2 finishes forwarding frame 1 to the Client.

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

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

4.3 Adapting to changing neighborhood

Sections 4.1 and 4.2 explained the basic mechanism used for streaming. Often a neighborhood may change and, hence, the system should adapt to Neighbors joining and leaving. This section explains the mechanisms CStream uses to adapt to these changes.

4.3.1 Neighbor Joining

We continue to use the same example in the section 4.2 to explain how the CStream system adapts to new Neighbors joining the network.

When a new Neighbor running CStream comes into the vicinity of the existing ad-hoc network (Neighbor N3 figure 5), the Neighbor joins the 'CStream' ad-hoc network. It receives the periodic REQUEST broadcasts from the Client seeking help. The flow sequence of Neighbor joining scenario is explained step by step using Figure 4.3, a continuation of the example scenario shown in Figure 4.2.

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.

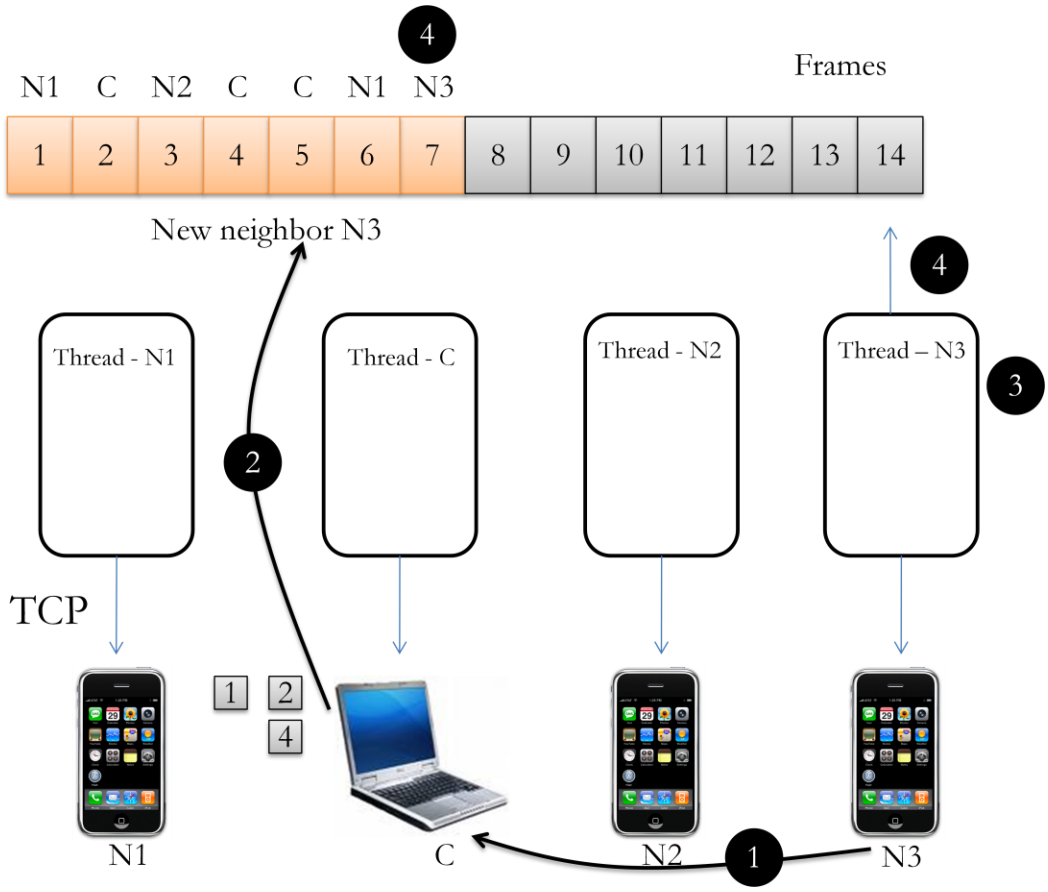

Figure 4.3: Adapting frame distribution to neighbor joining

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.

4.3.2 Neighbor Leaving

We will continue on the example in the section 4.3.1 to explain how the system adapts when an existing Neighbor leaves. Suppose Neighbor N1 leaves, the following describes the sequence of events:

Step 1: Neighbor N1 leaves. This is signaled by Neighbor N1 not sending I-CAN-HELP messages for three 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 about the last frame received from that Neighbor so that the server can recover and redistribute lost frames. Here, the last frame received from N1 is frame 1.

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

Step 5: The Server keeps 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. Now, since the last frame received from the Neighbor is 1, the server knows it needs to redistribute the rest of the frames assigned to N1. It inserts these frames into 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 back to the queue.

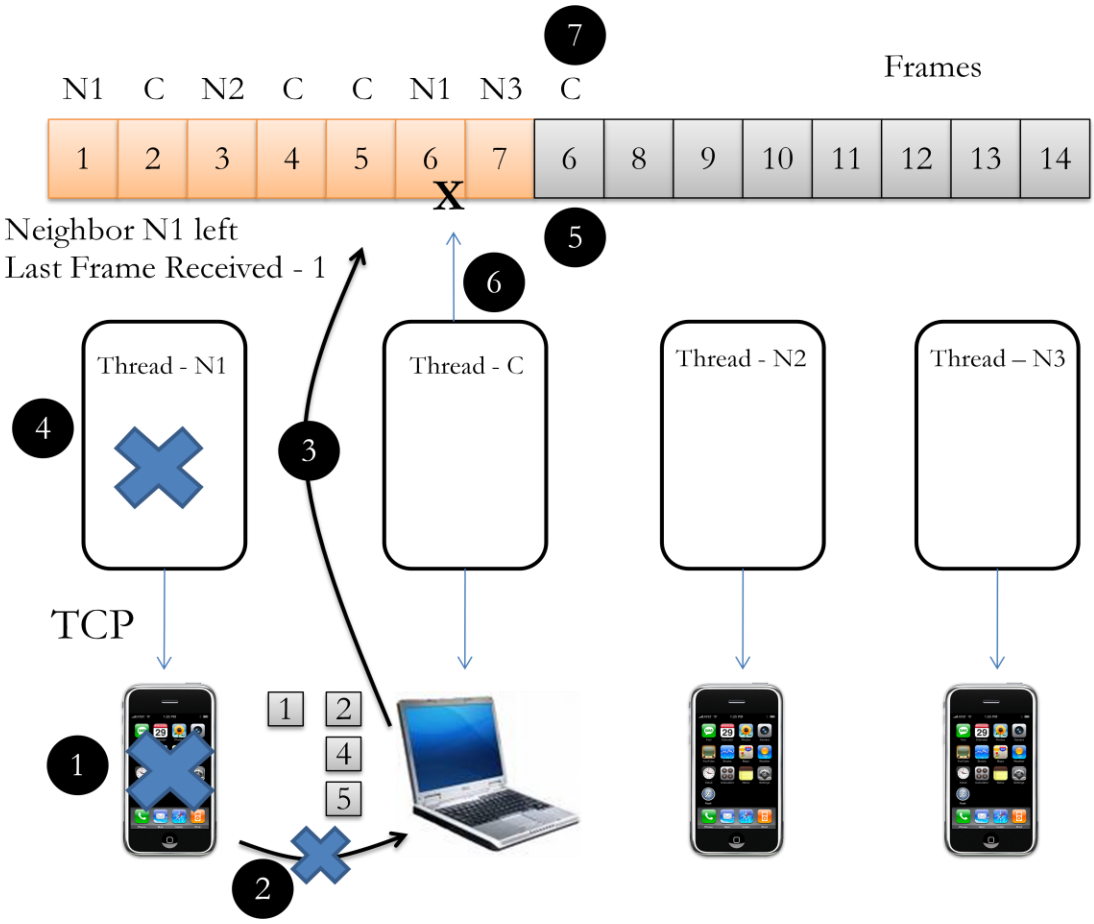

Figure 4.4: CStream adapting to neighbor leaving

Step 6: Suppose the Client thread now has spare bandwidth, it requests for 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.

Thus the system ensures that all the frames are delivered reliably and 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. We implement a policy where the player stops and waits for late frames to arrive as opposed to discarding late frames. The Buffer Manager which receives frames from the Server and through the Neighbor 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 first two seconds of the video to be buffered before starting to play. That is, we wait for *(2 * fr)* frames where *fr* is the frame rate. The choice of 2 to 4 seconds is common in many popular video players [oYT]. We found that around two seconds was optimal in our system both to decrease the startup delay and to give a good visual quality without too many rebuffer events.

Stop and Rebuffering: After the initial buffering, the player plays the frames continuously as long as they are in the buffer. The Player plays the video with the appropriate frame rate that was reported in the meta-data. 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. It waits for the frames to arrive.

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

To illustrate with an example, assume that the frame rate of a video is 15 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 it starts playing. Suppose the 45th frame is not available in the buffer when the player is supposed to play it. The Player 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) is received before starting to play again. Suppose another rebuffer event happens at the 105th frame, the player waits until all the 120 frames are received before playing again.

5 Evaluation

We built the complete CStream system comprising the Video Server, Client and the Neighbor components. CStream was written in C# .NET with code base of about 3000 lines. Source video files are stored in AVI format and CStream uses an open source AVI Video Library [oAVI] to extract the frames from the source video files. Figure 5.1 shows a screen shot of the CStream Video Player. Users can request a video by filename using 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. It also displays the aggregate throughput, the playout time and the number of rebuffer events.

Figure 5.1: CStream system user interface

5.1 Experimental Setup

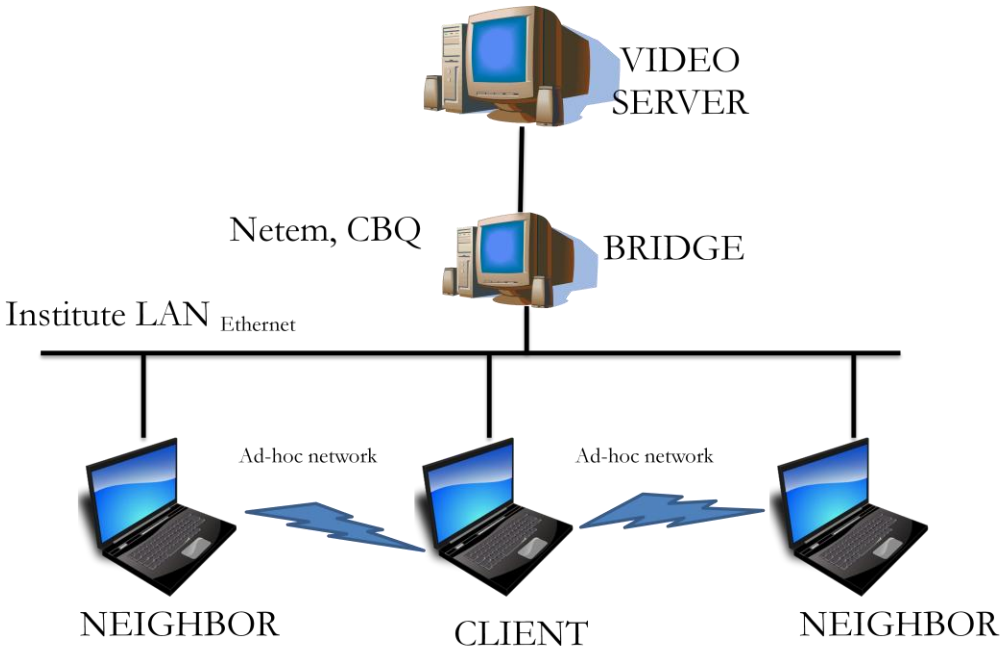

Figure 5.2: Experimental setup

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 using CBQ and the experiments were run for different bandwidth settings.

Figure 5.2 shows the experimental setup. 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. 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) to perform traffic shaping on traffic from Video Server. 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 machine and the Neighbors

are connected to each other through a wireless ad-hoc network. All the machines in our experimental setup are in our Institute LAN.

5.1.1 Bandwidth Control

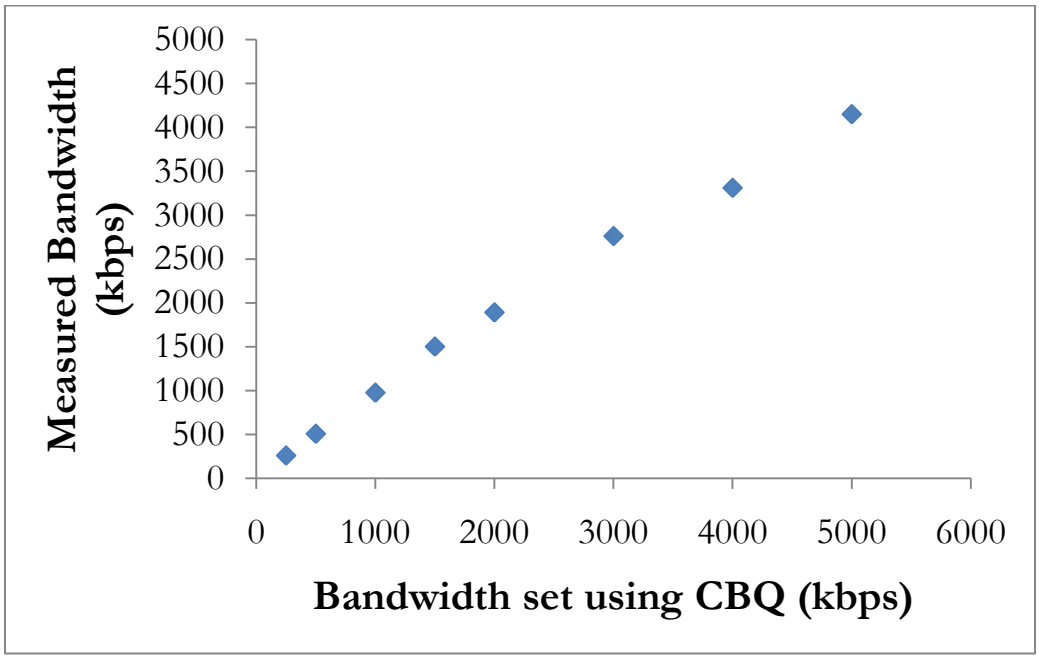

Figure 5.3: Bandwidth set using CBQ vs. measured bandwidth

We use the Linux bridge to connect Video Server to the Institute LAN and hence all the traffic from the Video Server goes via the bridge. Class based queuing (CBQ) discipline allow us to control bandwidth to specific destination IP addresses. CBQ on the bridge is used to control the bandwidth on a per flow basis from Video Server to each of the Neighbor and the Client. To ensure correct operation, bandwidth values set using CBQ is validated by measuring the throughput from the Video Server to the nodes (with iperf) over a 30 second time interval. Figure 5.3 show that the actual throughput measured closely matches the expected throughput.

5.1.2 Experimental Parameters

We evaluated the performance of the CStream system by varying the following parameters

- **Number of Neighbor nodes**: The number of Neighbors is varied from zero to two.
- **Bandwidth of the nodes to the Video Server**: The available bandwidth on the link from the Video Server to the Client and the Neighbors is varied using bandwidth control as described in the previous section. Six different bandwidth settings: 250 Kbps, 500 Kbps, 1 Mbps, 2 Mbps, 3 Mbps and 5 Mbps are used in the experiments
- **Video content**: Different video contents are used in the experiments.. A short and a long video is used to evaluate CStream. The details of the video are listed in Table 5.1.
- **Location of nodes**: The physical location of Neighbors with respect to the Client is changed to vary the wireless ad-hoc network bandwidth.

5.1.3 Performance Metrics

The performance of CStream is evaluated using the following metrics. These metrics are measured at the Client. We log the frame number, timestamp of each arrived frame, frame size and information about the link (Client or Neighbor) through which the frame was transferred. The metrics are then calculated offline using the log.

- **Aggregate Throughput (Kbps):** Aggregate throughput is the total application throughput at the Client. It is 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. The total time taken is the sum of 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.
- **Re-buffer events:** The number of rebuffer events is the number of times the Video Player stopped to rebuffer frames after it 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 cartoon_dog.avi	**Long Video** foreman.avi
Length	8 seconds	33 seconds
Size	10 MB	26 MB
Encoded bitrate	10 Mbps	6.3 Mbps
Frames per second	15	12
Average frame size	85 KB	68 KB
Total frames	120	400
Resolution	320x240	176x144

Table 5.1: Features of the video content

5.2 Results

We evaluate all the performance metrics for both the short video and long video. For all the experiments, we present the average over three runs of the experiment along with the standard deviation. Since the variance of the metrics was very less across the runs, average over three runs gave a good estimate of the CStream performance. Unless stated, equal bandwidth is set for the Client and the Neighbors for all the experiments. Also, unless otherwise stated, 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 was around 13 Mbps (significantly higher than the wired bandwidth).

5.2.1 Aggregate Throughput

First, the aggregate throughput, i.e. the throughput observed at the Client both in the presence and absence of Neighbors is measured. The aggregate throughput is calculated as the ratio of the total size of the video downloaded to the total time taken to download all the frames.

For each bandwidth setting (CBQ), we vary the number of Neighbors and compare the throughput. Figure 5.4 and Figure 5.5 show the throughput for each bandwidth setting with 0, 1 and 2 Neighbors for short video and long video respectively. The results show that bandwidth linearly increases with the increase in Neighbors. Table 5.2 and Table 5.3 show the average and standard deviation of throughput for the short video and long video, respectively.

For every bandwidth setting, the throughput approximately doubles in the presence of one Neighbor and it triples in the presence of two Neighbors with the Client and the Neighbor having the same bandwidth. As an example, when the bandwidth was set to 1 Mbps, with any number of Neighbors, the Client got a throughput of 938 Kbps for the short video. In the presence of one Neighbor with the same bandwidth, the throughput increased to 1854 Kbps. In the presence of two neighbors the throughput was 2806 Kbps. For both the short video and the long video the throughput numbers for all the settings remains similar. This shows that, irrespective of the video content, CStream effectively utilizes bandwidth. The increase in throughput improves the quality of streaming significantly as shown in the subsequent sections.

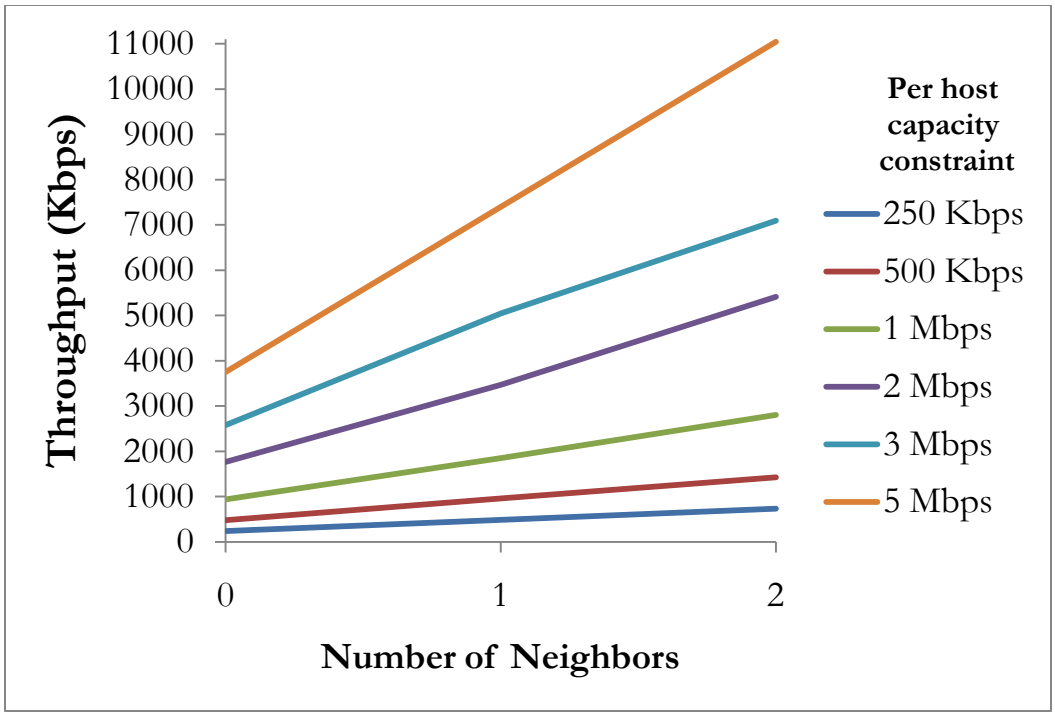

Figure 5.4: Average aggregate throughput for short video

Per host bandwidth constraints	Number of neighbors					
	0		1		2	
	Avg	StdDev	Avg	StdDev	Avg	StdDev
250 Kbps	239	0.3	486	1.6	734	0.8
500 Kbps	477	0.7	962	2.4	1430	8.4
1 Mbps	938	1.6	1854	22.7	2806	23
2 Mbps	1767	3	3472	21.3	5411	219.7
3 Mbps	2576	28.7	5047	30.7	7096	156.3
5 Mbps	3754	179.3	7396	219.1	11047	1413.9

Table 5.2: Average and standard deviation of aggregate throughput (in Kbps) for short video

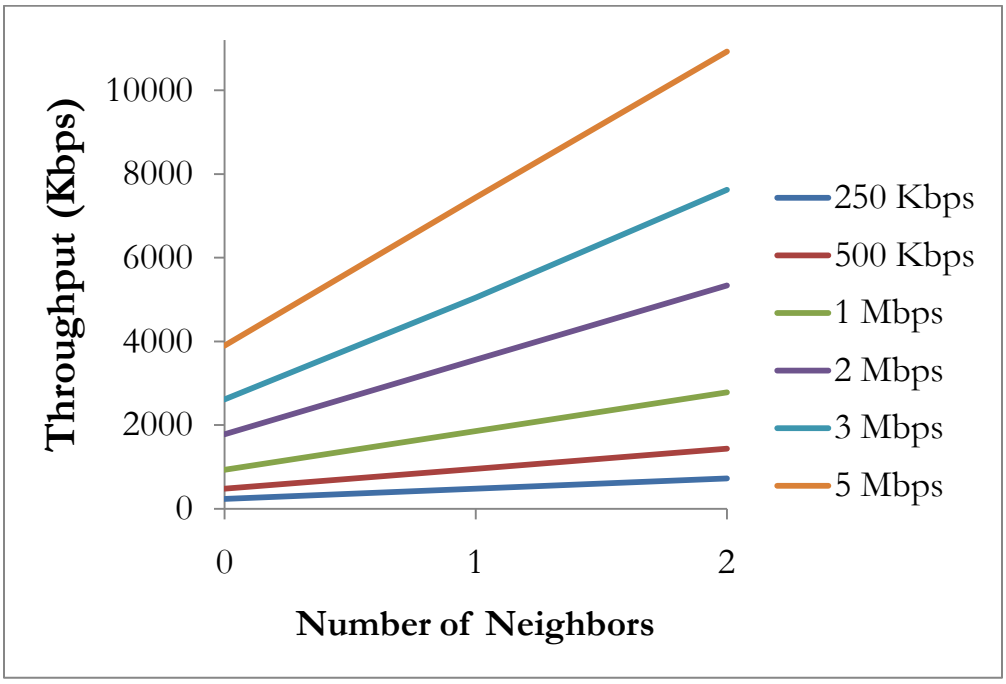

Figure 5.5: Average aggregate throughput for long video

Per host bandwidth constraints	Number of neighbors					
	0		1		2	
	Avg	StdDev	Avg	StdDev	Avg	StdDev
250 Kbps	238	0.1	483	4.6	726	6.2
500 Kbps	476	0.3	955	1.4	1434	3.2
1 Mbps	932	10.2	1854	21	2778	35.5
2 Mbps	1777	15.4	3558	70.6	5338	180.7
3 Mbps	2611	51	5045	217	7624	72.4
5 Mbps	3902	17.5	7437	142.7	10925	222.4

Table 5.3: Average and standard deviation of aggregate throughput (in Kbps) 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 measured. The playout time is the total time taken to play the video, including the startup delay, the time taken in rebuffer events and the playing time.

Figure 5.6 and Figure 5.7 shows the playout time for every bandwidth setting for the short and long video, respectively. Table 5.4 and Table 5.5 show the corresponding average values and the standard deviation. As can be seen, for a given bandwidth setting, the playout time decreases multiplicatively with the increase in the number of neighbors. For instance, for the 1 Mbps setting for the short video, without any neighbors, the playout time is around 90 seconds. With one neighbor, the playout time decreases to around 45 seconds, a 50% decrease in playout time. With two neighbors, the playout time decreases to around 31 seconds, a 66% decrease in playout time.

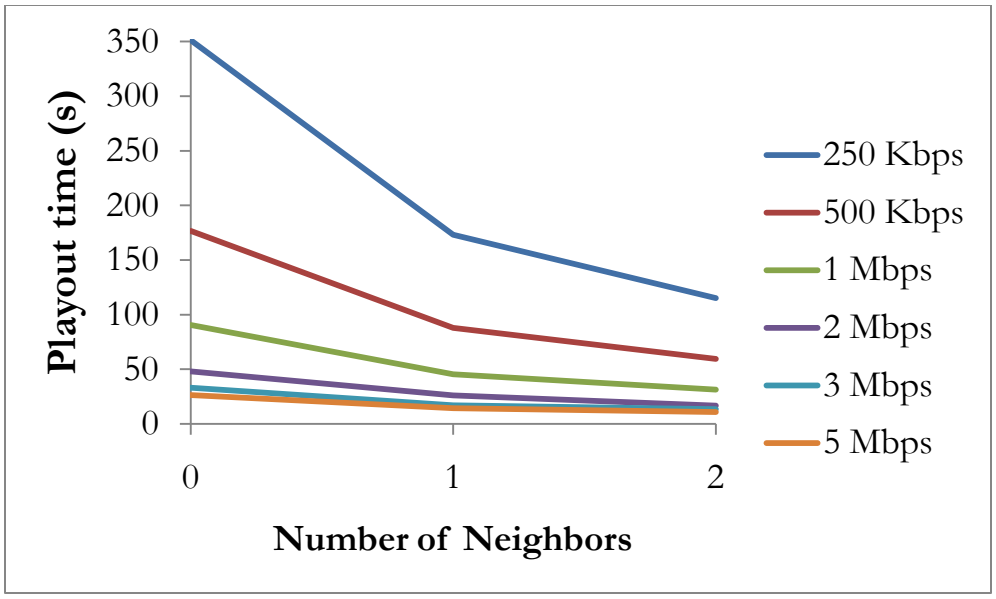

Figure 5.6: Average playout time for short video

Per host bandwidth constraints	Number of neighbors					
	0		1		2	
	Avg	StdDev	Avg	StdDev	Avg	StdDev
250 Kbps	351.31	0.44	173.22	0.56	115.05	0.24
500 Kbps	176.68	0.24	87.93	0.39	59.44	0.37
1 Mbps	90.42	0.16	45.34	0.95	31.39	0.16
2 Mbps	48.02	0.08	25.99	0.14	16.68	1.19
3 Mbps	33.09	0.17	16.96	0.21	13.52	0.60
5 Mbps	26.34	1.18	14.47	0.42	10.9	0.74

Table 5.4: Avg and stddev of playout time (in seconds) for short video

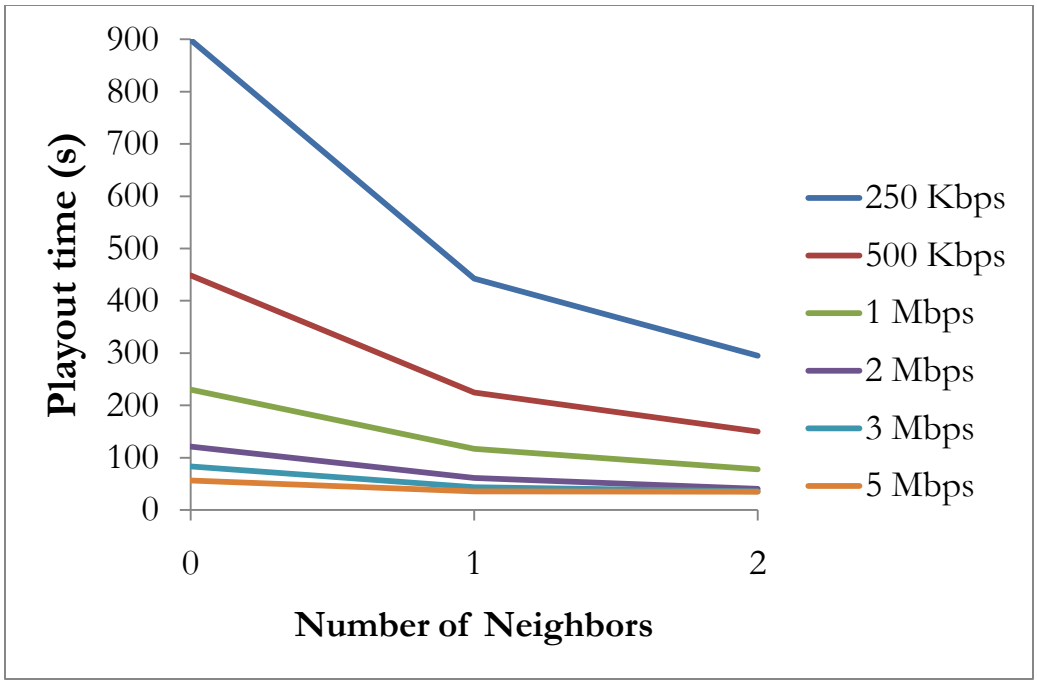

Figure 5.7: Average playout time for long video

Per host bandwidth constraints	Number of neighbors					
	0		1		2	
	Avg	StdDev	Avg	StdDev	Avg	StdDev
250 Kbps	898.9	0.41	442.6	4.14	294.7	2.58
500 Kbps	448.3	0.28	224.7	0.50	150.1	0.61
1 Mbps	230.2	2.53	116.6	1.34	77.4	1.14
2 Mbps	121.4	0.94	61.1	1.73	40.6	1.33
3 Mbps	83.4	1.61	43.4	2.02	35.6	0.21
5 Mbps	56	0.07	35.3	0.01	34.8	0.02

Table 5.5: Average and standard deviation of playout time (in seconds) for long video

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 before playing the first frame.

Figure 5.8 and Figure 5.9 show the startup delay with the increase in number of neighbors for each bandwidth setting for the short video and long video, respectively. Table 5.6 and Table 5.7 are corresponding tables that summarizes the average and the standard deviation. Similar to the playout time graph, there is approximately multiplicative improvement in the startup delay. As an example, the startup delay at 1 Mbps with 0 neighbors for short video is around 22 seconds. The startup delay reduces to around 11.5 seconds for one Neighbor, a 2x improvement. With two Neighbors, the startup delay is around 7.5, a 3x improvement compared to the scenario with no Neighbors.

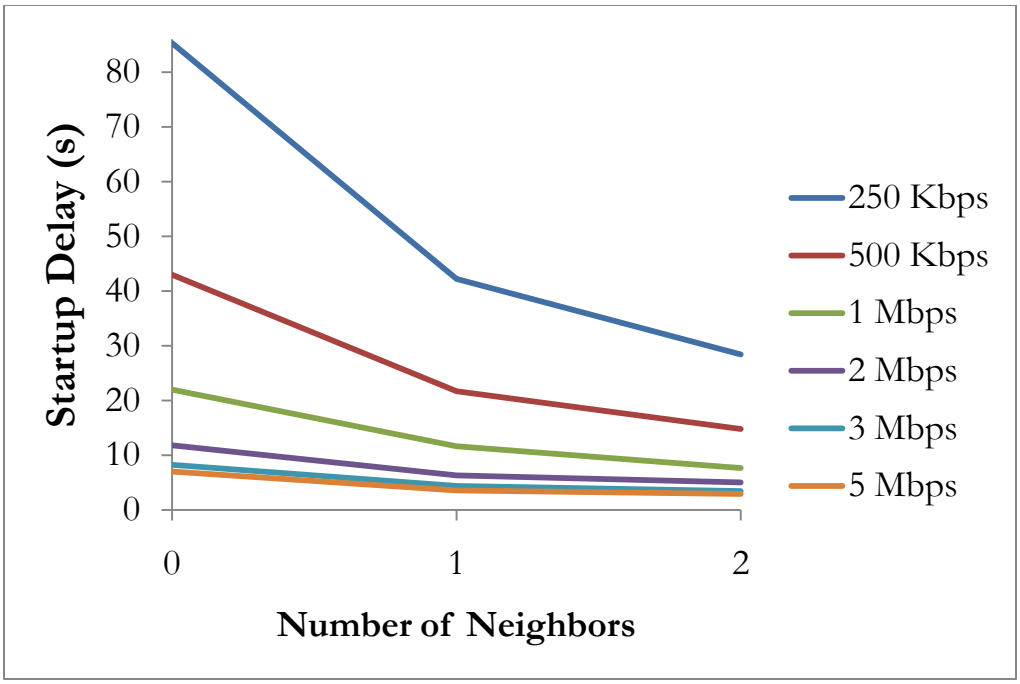

Figure 5.8: Average startup delay for short video

Per host bandwidth constraints	Number of neighbors					
	0		1		2	
	Avg	StdDev	Avg	StdDev	Avg	StdDev
250 Kbps	85.31	0.46	42.2	0.31	28.42	0.27
500 Kbps	43	0.22	21.69	0.16	14.8	0.14
1 Mbps	21.97	0.15	11.64	0.43	7.63	0.09
2 Mbps	11.83	0.11	6.33	0.15	5	1.21
3 Mbps	8.22	0.03	4.41	0.14	3.43	0.11
5 Mbps	7.03	1.3	3.55	0.11	2.92	0.74

Table 5.6: Average and standard deviation of startup delay (in seconds) for short video

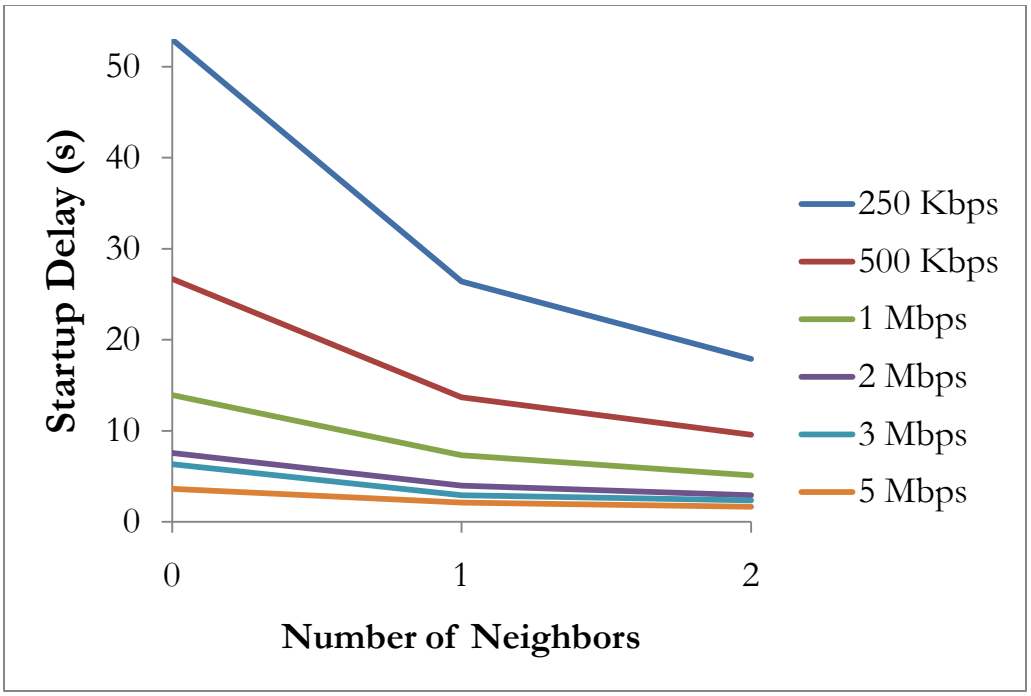

Figure 5.9: Average startup delay for long video

Per host bandwidth constraints	Number of neighbors					
	0		1		2	
	Avg	StdDev	Avg	StdDev	Avg	StdDev
250 Kbps	52.97	0.13	26.4	0.39	17.9	0.37
500 Kbps	26.69	0.02	13.68	0.06	9.57	0.01
1 Mbps	13.94	0.12	7.32	0.12	5.12	0.19
2 Mbps	7.58	0.25	4	0.05	2.95	0.16
3 Mbps	6.33	1.81	2.94	0.03	2.37	0.21
5 Mbps	3.65	0.15	2.12	0.01	1.65	0.02

Table 5.7: Average and standard deviation of startup delay (in seconds) for long video

5.2.4 Rebuffer Events

The final performance measure to assess the quality of the video is the number of rebuffer events. The number of rebuffer events is the number of times the Video Player stopped to rebuffer frames after it started playing. In our implementation, the CStream Video Player stops when it does not have the next frame in the sequence to play. After it stops, it waits until the next two seconds of frames is buffered except when the total remaining frames to be buffered is less than two seconds worth of frames.

Table 5.8 shows the average number of rebuffer events over three experimental runs for every bandwidth and Neighbor setting for the short video. Table 5.9 shows the same measure 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 bandwidth constraints	**Number of neighbors**		
	0	**1**	**2**
250 Kbps	3	3	3
500 Kbps	3	3	3
1 Mbps	3	3	2
2 Mbps	3	2	1
3 Mbps	2	1.3	1
5 Mbps	2	1	0

Table 5.8: Average of number of rebuffer events for short video

Per host bandwidth constraints	Number of neighbors		
	0	**1**	**2**
250 Kbps	15	14.3	14
500 Kbps	15	13	12
1 Mbps	13	11	9
2 Mbps	11	6.6	2
3 Mbps	9	2.6	0
5 Mbps	6	0	0

Table 5.9: Average of number of rebuffer events for long video

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 Neighbor during streaming is compared to their corresponding bandwidth settings. An ideal frame assignment scheme should distribute the frames 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 66.66% of the video (frames) should be sent through the Client and 33.33% through the Neighbor to minimize the total download time.

We measure the total size of the frames downloaded by each node and compare their ratio to the bandwidth settings. The frame sizes in both the short and the long video are different. Figure 5.10 shows four different experiments with the short video. In the first experiment, there was one Neighbor and the

bandwidth of both the Neighbor and the Client is set to two Mbps. The graph shows the contribution of each node during streaming. As expected, both the nodes contributed around 50% each. In experiment 2, we set unequal bandwidth for the Client and the Neighbor. The Client bandwidth is set to 2 Mbps and the Neighbor bandwidth to 1 Mbps. Again, in this scenario, the contribution of each node almost matches the ratio of their bandwidths. In experiment 3 and experiment 4, we had two Neighbors with equal and unequal bandwidth respectively. In the both the experiments, the ratio of the video frames downloaded by each node matched the ratio of their bandwidths. Figure 5.11 shows the result of a similar set of experiments for the long video.

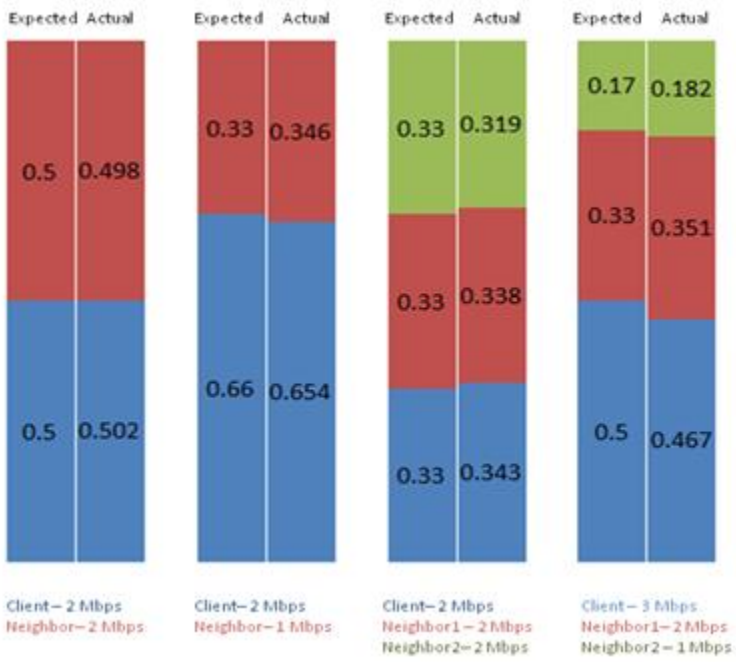

Figure 5.10: Frame distribution vs. ratio of bandwidth for short video

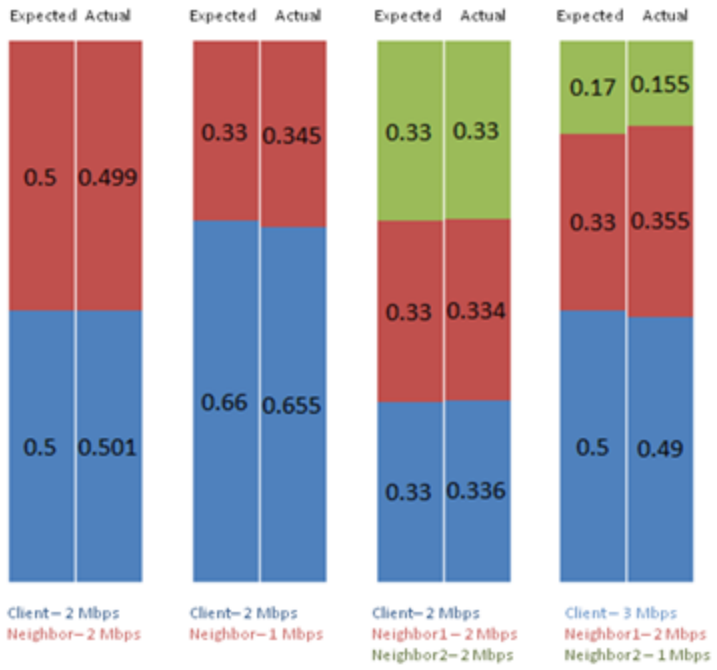

Figure 5.11: Frame distribution vs. ratio of bandwidth for long video

5.2.6 Impact of Wireless

Finally, we study the impact of wireless bandwidth on video streaming. In all the previous experiments, the wireless bandwidth was more than the wired bandwidth and hence the overall throughput was limited by the wired bandwidth. We study the impact of wireless by changing the location of the Neighbors with respect to the Client. In wireless, the throughput depends upon the signal strength between the nodes which decreases as the distance between the nodes increase. We varied the location of the Neighbor such that it had excellent, good and bad signal strength, visually based on the bars in the "*Connect to a network*" Windows dialog.

In our experiments, there was one Neighbor in addition to the Client both having a bandwidth of 2 Mbps. Long video is used for evaluating the Neighbor leaving and joining scenarios. Figure 5.12 shows the aggregate throughput

obtained in each of the scenario. In the excellent signal strength scenario, the wireless bandwidth was around 13 Mbps. In the good signal strength scenario, the wireless bandwidth was around 1 Mbps and it was around 500 Kbps in the poor signal strength scenario. As seen, the aggregate throughput dropped as the distance between the nodes increased. Specifically, it can be noted that the bandwidth contributed by the Neighbor in the good and poor scenario was constrained by the wireless bandwidth. In the excellent scenario the aggregate throughput was around 3500 Kbps, while in the good scenario it was around 2400 Kbps and in the poor scenario it was around 2000 Kbps

The bandwidth contributed by the Neighbor in CStream is the minimum of the wired and wireless bandwidth.

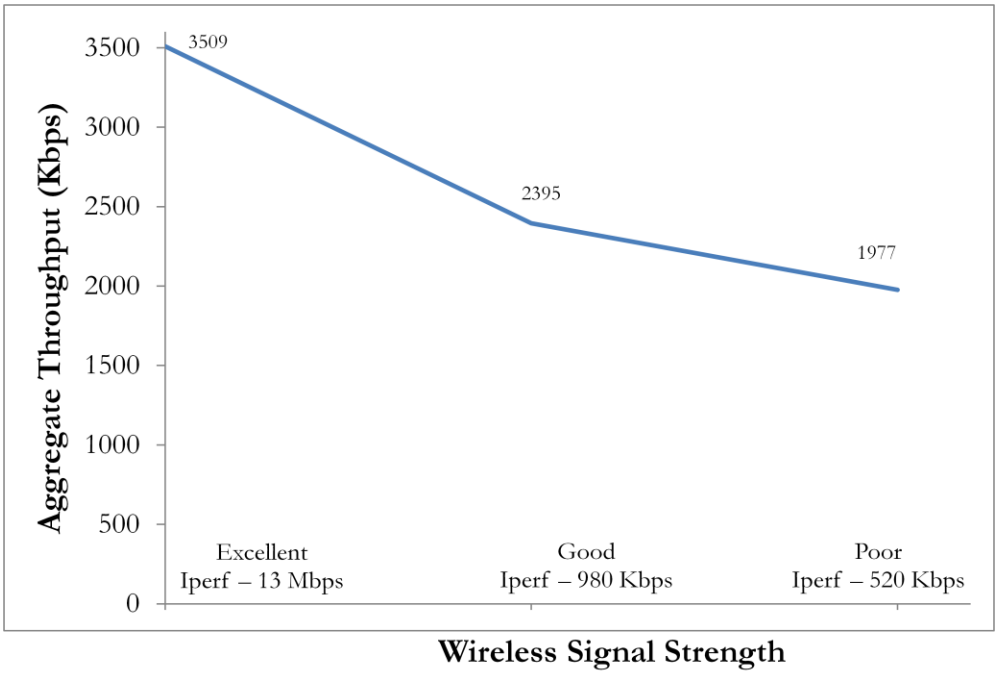

Figure 5.12: Impact of wireless signal strength on aggregate throughput for long video. In the experiment, there was one client and one neighbor both with bandwidth 2 Mbps

5.2.7 Neighbors Joining

Additionally, we studied the impact in the overall throughput when neighbors join and leave in the middle of a streaming session. We measured the instantaneous throughput over time observed at the Client when the neighborhood changed. The instantaneous throughput in our experiments was calculated as the ratio of the total size of the last 20 frames to the time taken to download them. We choose 20 frames to smooth the fluctuations in throughput.

Figure 5.13 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 a 1787 Kbps with no Neighbors to 3576 Kbps) after the first Neighbor joined and almost tripled (to 5747 Kbps) after the second Neighbor joined.

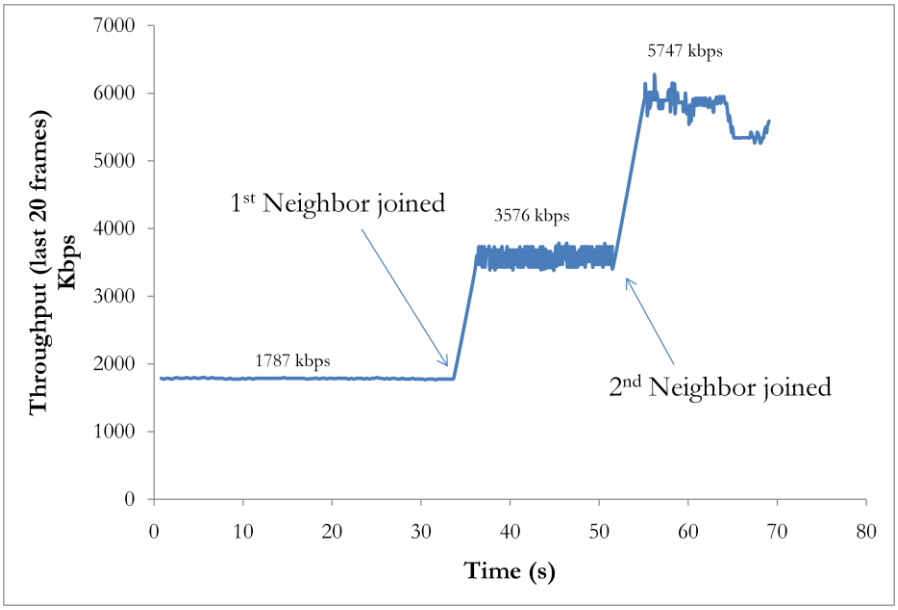

Figure 5.13: Change in instantaneous throughput when neighbors join

5.2.8 Neighbors Leaving

Figure 5.14 shows a similar experiment 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 5653 Kbps to 3773 Kbps when the first Neighbor left and to 1781 Kbps when the second Neighbor left. Both the experiments show that CStream dynamically handles changing neighborhood and effectively uses the available bandwidth.

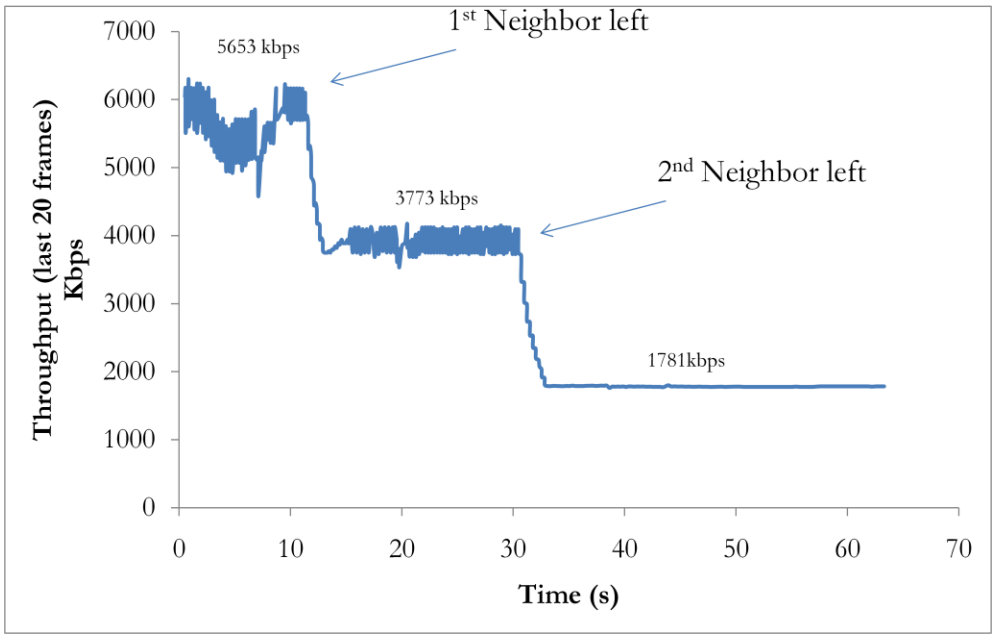

Figure 5.14: Change in instantaneous throughput when neighbors leave

6 Future Work

This chapter presents the possible extensions to CStream. Section 6.1 discusses a better frame distribution scheme that will improve the delivery of frames. Section 6.2 discusses plans for real world deployment and evaluation. Then, Section 6.3 and 6.4 details how CStream can be used with other video types and also streaming audio as a part of the video. Finally, Section 6.5 discusses security and incentives for CStream.

6.1 Better Frame Distribution

The current implementation of CStream adapts to the changes in the bandwidth, but when the bandwidth of the links is unequal, it leads to out of order delivery of frames. For example, if the Client is ten times faster than the Neighbor, and the first frame is assigned to the Neighbor, then according to current scheme it is likely that frames 2 to 11 will streamed via the Client link. Frame 1 will likely be received by the Client only after it has streamed frames 2 to 11. This out of order delivery of frames may impact the performance of video streaming and may result in increased rebuffer events. In such a case where the Client and Neighbor links have different bandwidth, then the frame distribution algorithm could assign the frames more intelligently. A solution is to keep track of the frame requests by the Client and Neighbor threads and assign frames based on their frequency compared to others. The bandwidth of the links could be estimated and frames assigned based on their ratios. To make sure frames arrive in order, the frame assignment scheme could assign frames to threads in such a way that by the time they reach the Client, they are almost in order. For example, in the above scenario,

if the Video Server estimates that the bandwidth of the Neighbor is ten times slower than that of Client, it would look ahead and assign frame 11 to the Neighbor instead of the frame at the head of the queue (frame 1). In that way, frame 11 will arrive at the Client at the same time it had finished streaming frames 1 to 10 through the Client link.

6.2 Real-World Deployment

A natural next step in evaluation is to measure the performance of CStream in a real-world setting. A real-world deployment, say in an apartment complex, will help study ISP diversity, their bandwidth optimizations and its effect on CStream performance. It will also help evaluate our assumption on the density of nodes and node idle time.

In addition, we evaluated CStream performance on PCs but an immediate possible extension is to use CStream to aggregate 3G bandwidth for smart phones and see how it improves the video streaming performance. To accomplish this, the CStream code would need to be ported to a smart phone or a 3G modem could be used on the PCs. We can also evaluate the performance of CStream in scenarios that involve both 3G and Ethernet links (a mix of laptops and smart phones).

6.3 Other Video Formats

The source videos were stored in AVI format in our CStream implementation. CStream can be extended to support other video types like MPEG which are layer encoded. For instance, MPEG videos have three kinds of frames: I, B and P frames. I frames define the base layer of the video and the B

and P frames are enhancement layers. CStream could stream these videos in such a way that it sends base layer through the Client link and enhancement layers through the Neighbors.

Additionally, CStream currently does not do video scaling when the combined bandwidth of the participating neighbors and the Client node is not sufficient to support the video. CStream could be extended to include different methods of video scaling when there is not enough bandwidth to stream the video.

6.4 Audio stream

The initial CStream system built sends only video frames and excluded the audio stream. A possible extension is to consider audio as part of the stream. One simple implementation would be to stream the audio entirely through the Client link and give higher priority to audio stream compared to video stream.

6.5 Incentives and Security

CStream assumes that all the Neighbors are willing to collaborate and are trusted, so there is no focus on incentives or security. One possible area of future work is to examine the security issues the system needs to address. This includes the Video Server encrypting and signing video frames and the Client verifying the content to prevent man-in-the-middle attacks. And maintaining a trusted set of neighbors and blacklisting 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. A simple solution for incentives is to implement a TFT (tit for tat) based scheme as in BitTorrents [LRLZ06]. In a TFT based

scheme, a node gains credit when it uploads data to other peers and spends the credit to download data from other peers. A more complex solution is to design a micropayment based scheme similar to COMBINE [APRT07] to incentivize nodes to help. The payment scheme COMBINE uses includes a signed node of credit, termed an IOU (abbreviated from the phrase "I owe you") indicating the amount of payment made.

7 Conclusion

The popularity of video streaming systems has tremendously increased over the past few years. Despite its popularity, video streaming still remains a challenge in many scenarios. With limited broadband bandwidth at homes and 3G bandwidth in smart phones, the quality of video streaming suffers. To overcome this problem, we note two important network characteristics typical of most homes. First, there is a high density of Internet connections available in every neighborhood with many idle users. Although the available bandwidth at one Client is limited, the total unused bandwidth available in a neighborhood is high and can be aggregated together. Second, most of today's wireless devices have multiple interfaces that enable them to connect to nearby devices in an ad-hoc network at the same time they are connected to the Internet. CStream leverages the above two facts to aggregate Internet bandwidth for better video streaming.

This thesis presents CStream prototype, 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 its quality. We designed and built an entire CStream system including the Video Server, Client and the Neighbors. CStream by design dynamically adapts to changes in the neighborhood.

CStream was evaluated on a small test bed of computers. Aggregate throughput achieved with the CStream system- was measured varying the number of Neighbors participating in video streaming. Video quality was 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 system increases linearly with the increase in the number of neighbors participating in the streaming. Playout time to stream and play the entire video also decreases multiplicatively as the number of neighbors increased. Similarly, the startup delay to play the first frame also decreased with the increase in number of contributing neighbors. A sharp decrease in the rebuffer events was observed as more Neighbors participated in video streaming. We ran experiments to verify the contribution of the Neighbors to video streaming and the results show that the ratio of the frames contributed by all nodes (both Client and Neighbor) is proportional to the available bandwidth. We finally studied the impact of the location of the nodes, placing the Neighbors in different positions relative to the Client such that the signal strength between the Client and the Neighbor was excellent, good and bad. We observed that bandwidth contributed by the Neighbor was constrained by the limited wireless throughput for good and bad signal strengths. So bandwidth contributed by Neighbor is then limited by the minimum of the wired bandwidth and wireless throughput to the Client. CStream experimental results demonstrate how collaborative streaming improves the aggregate throughput and the quality of video streaming.

The contributions of this thesis include the following:

- Proposed a novel system for video streaming that connects neighboring nodes in an ad-hoc network to aggregate Internet bandwidth.
- Designed a system that streams video through multiple Internet connections and that dynamically adapts to the changing neighborhood (nodes joining and leaving).

- Implemented a frame distribution scheme that distributes video frames across multiple connections and makes full utilization of the available bandwidth.
- Built the entire CStream system including the Client, Neighbor and the Video Server. It also includes a Video Player through which users can request and play videos.
- Performed detailed performance evaluation of the entire system over a range of video and network configurations, showing a linear improvement in throughput and video quality as the number of co-operating users increases.

Bibliography

[80211N] IEEE 802.11n-2009—Amendment 5: Enhancements for Higher Throughput. IEEE-Standards Association. 29 October 2009. doi:10.1109/IEEESTD.2009.5307322

[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. In *Elsevier Journal of Computer Networks*, vol. 47, no. 4, pp. 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, In *IEEE Transactions on Mobile Computing*, vol. 5, no. 4, pp. 388-403, April 2006.

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

[JJKPS08] S. Jakubczak, M. Jennings, M. Kaminsky, K. Papagiannaki and S. Seshan. Link-alike: Using Wireless to Share Networking Resources in a Neighborhood. In *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R) (invited paper)*, vol. 12, no. 4, October 2008.

[KLBK08] S. Kandula, K. Lin, T. Badirkhanli and D. Katabi. FatVAP: Aggregating AP Backhaul Bandwidth. In *Proceedings of Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, April 2008.

[LRLZ06] J. Liu, S. Rao, B. Li and H. Zhang. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. In *Proceedings of the IEEE*, vol. 96, no. 1, pp. 11-24, January 2008.

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

[oEE09] The Exabyte Era. http://www.cisco.com/web/IN/about/network/the_exabyte_era.html. *Retrieved on* 10 Sep. 2009.

[oORB] Orb. http://www.orb.com.

[oWiki3G] 3G. http://en.wikipedia.org/wiki/3G.

[oYT] YouTube. http://www.youtube.com.

[PMDC03] R. Prasad, M. Murray, C. Dovrolis and K. Claffy. Bandwidth Estimation: Metrics, Measurement, Techniques and Tools. In *IEEE Network*, vol. 17, pp. 27-35, November 2003.

[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.