A Content Delivery Model for Online Video

by

Liang Yuan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2009

Liang Yuan

Abstract

Online video accounts for a large and growing portion of all Internet traffic. In order to cut bandwidth costs, it is necessary to use the available bandwidth of users to offload video downloads. Assuming that users can only keep and distribute one video at any given time, it is necessary to determine the global user cache distribution with the goal of achieving maximum peer traffic.

The system model contains three different parties: viewers, idlers and servers. Viewers are those peers who are currently viewing a video. Idlers are those peers who are currently not viewing a video but are available to upload to others. Finally, servers can upload any video to any user and has infinite capacity.

Every video maintains a first-in-first-out viewer queue which contains all the viewers for that video. Each viewer downloads from the peer that arrived before it, with the earliest arriving peer downloading from the server. Thus, the server must upload to one peer whenever the viewer queue is not empty. The aim of the idlers is to act as a server for a particular video, thereby eliminating all server traffic for that video. By using the popularity of videos, the number of idlers and some assumptions on the viewer arrival process, the optimal global video distribution in the user caches can be determined.

Acknowledgements

I want to express my gratitude to my supervisor, Dr. Gordon Agnew, without whom this thesis would not have been possible. He was the one who brought me into the Cisco Graduate Internship Program. With his inspiration, guidance and encouragement, he helped make this short journey fun for me. I wish to also thank Professor Ian Munro and Professor Paul Ward for reviewing my thesis.

I would also like to thank the numerous people at Cisco: my manager, Milton Xu, my mentor, Dr. Steve Workman, and of course Suran De Silva for their kind assistance and helpful advice.

I am indebted to my friends who put up with my sometimes difficult self. They inspired me, cared for me and most importantly gave me reasons to persevere. Thank you, Andy Sun, Bo Hu, Brian Ji, Fangjin Yang, Funing Gao, Henry Hoang, Murched Ajami, Nancy Xu, Omar Karim, Pat Nicholson and Yan Huang.

Finally, I wish to thank the teachers that have been a part of my life from the very beginning: my elementary school teachers at Rödaberg, my junior high teachers (especially Mr. Miwa), my high school teachers (especially my programming teacher Mrs. Marshall) and my teachers at Waterloo (especially Dr. Larry Smith, Dr. Paul Ward, and Dr. MacPhie).

Dedication

I wish to thank my parents: my dad, Dr. Shi Zeng Yuan and my mom, Yi Qin Duan. They have always been there for me, taught me and loved me. I dedicate this thesis to them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The traffic makeup of the Internet is constantly changing. Today, a fundamental shift is taking place. For many years, P2P traffic used to dwarf HTTP traffic [1], largely driven by the relative ease at which music, movies and software could be downloaded. However, this trend is slowly changing. A recent study conducted by Ipoque has shown a proportional decrease in P2P traffic as compared to HTTP traffic [1] (see Table 1.1).

Table 1.1: Protocol usage in Germany [1].

Protocol class	2008/2009	2007
P2P	53%	69%
Web	26%	14%
Streaming	7%	8%
Other	1%	1%

Another study of North American broadband users even puts HTTP traffic above P2P traffic. The study by Ellacoya Networks [1] found that HTTP traffic accounted for 46% of all traffic where P2P traffic only accounted for 37% [2] (see Table 1.2).

Table 1.2: Proportion of protocol usage in North America. [2].

Type	%
HTTP	46%
P2P	37%
non-HTTP video streaming	3%
Other	14%

[1]In 2008, Ellacoya was acquired by Arbor Networks [7].

This change in traffic is largely due to file hosting sites, such as RapidShare and Megaupload taking over the use of P2P networks to deliver content, but also to the increasing media-richness of web pages [1] and in particular video streaming. Streaming video accounts for nearly 36% of all HTTP traffic [3]. YouTube (the most popular video website) accounts for nearly 20% of all HTTP traffic or almost 10% of all traffic on the Internet [3] (see Table 1.3).

Table 1.3: Distribution of HTTP traffic by bandwidth [3].

HTTP content type	%
Text and Images	45%
Streaming Video	36%
Streaming Audio	5%
Other	14%

The web has changed into a place where anyone can publish anything. A place where User Generated Content (UGC) is quickly becoming more important than traditional publisher content. In the meantime, P2P networks are still stuck on distributing content that is generally created by content providers: movies, TV shows, software and music. Table 1.4 shows that video traffic accounts for the majority (57%) of P2P traffic.

Table 1.4: BitTorrent content type distribution for a German University based on local traces [1]. Based on the data collected, video accounts for a large portion of BitTorrent traffic.

Category	Type	%
Video	Movie	30%
	TV	15%
	Anime	7%
	Porn	5%
Software	Games	24%
	Application	17 %
	Music	1 %

The various anti-piracy agencies around the world have long tried to shut down the illicit distribution of licensed content to very little success. However, new laws enacted by Sweden [8] has shown promise and will only push more and more P2P users onto the web. One reason why content providers have not embraced the P2P world is because these tools were not built to support revenue generation. There are no advertisements that can be shown at opportune times and no paid subscriptions.

However this is slowly changing as well. Knowing that it is difficult to fight an essentially free service in P2P networks, websites such as YouTube and Hulu have begun offering TV content and music videos free [9], subsidized only by advertisement. Content is provided in high quality and can be viewed without any waiting. By contrast, P2P content cannot be used until the full file has been completely downloaded.

If this continues, and video traffic from P2P continues to move in the direction of the web, and with the explosion of user generated content in its own right, YouTube faces serious scalability and bandwidth issues. By some estimates, YouTube will incur a $362 million dollar bandwidth bill in 2009 for serving at a peak rate of 30 million Mbps [4]. See Table 1.5 for more details.

Table 1.5: YouTube income and expenses [4].

Category	Type	$
Revenue	Advertisements	$240 million
Expenses	Bandwidth	$362 million
	Content licensing	$256 million
	Other	$93 million
	Total expenses	$711 million
Net Loss		$471 million

All of this points to a need for a P2P application that can offload video traffic on the web. This report serves to provide a model of such an application and some interesting results. Based on the analysis, a P2P system for YouTube is viable and can be highly beneficial in offloading traffic.

1.1 YouTube

At the forefront of the new Internet is the Youtube online video sharing service. It was started in February 2005 by three former PayPal employees [10]. It is by far the most popular of its kind and estimates in 2006 put the number of daily views at 100 million videos [11]. The number of videos on Youtube is also increasing at a rate of about 65000 a day, far more than traditional production schemes [11].

YouTube is perhaps the best example of the UGC revolution. Other sites in this category include Flickr (a photo sharing site), Facebook (a social networking site) and more recently Twitter (a micro-blogging platform).

Although a small fraction of requests to YouTube are for videos, the majority of data transferred is due to video traffic [12]. As video quality increases, this will become even more pronounced. Therefore, distributing video traffic is the most important aspect of the bandwidth requirements. One benefit of video traffic is that it does not change very often. The same video is viewed over and over again, unlike other items on the web page.

1.1.1 Operation

Every video uploaded to YouTube must be converted into Adobe's Flash Video (.flv) format. The benefit of this format is that anyone with the Flash Player 7 (or later versions) can view the videos without any additional downloads [12]. As of December 2008, the market penetration of Flash Player 7 exceeds 99% in most places [13].

The *external progressive download* [14] feature of the .flv format allows any part of a video to be viewed before the full content has been downloaded. Unlike traditional streaming methods, no dedicated, stateful, streaming servers are required. Any conventional HTTP web server is sufficient. Downloaded data is saved in the browser's cache and sent to the Flash Player (.swf) for display.

To view a video, a user must first navigate to a webpage containing the video. This amounts to a HTTP request to `www.youtube.com/watch?v=<videoID>`. Suppose one wanted to see one of the most popular videos (117 million views as of April 2009),"Evolution of Dance" at `http://www.youtube.com/watch?v=oHg5SJYRHA0`. The first request will return a HTML web page. After parsing this page, the web browser will make additional requests. Embedded content such as .css, .js, .jpg, .ico, and .swf files are served by `ytimg.com`. The .flv may be downloaded from YouTube directly, a cache like `v13.lscache6.googlevideo.com` or perhaps a Content Distribution Network like Limelight. The HTML web server performs a load balancing role to distribute traffic efficiently. This is all illustrated in Figure 1.1.

A P2P system should hook into this existing architecture with as few modifications as possible. Information about the P2P network can be embedded into the `.html` file. If a client supports P2P, instead of requesting the video by traditional means a P2P network can be used. This way, no other feature of the system is affected. For example, any sort of statistics tracking, advertisements, or user specific information can be served as usual.

As an example consider Figure 1.2. This system requires users to have two pieces of software: a proxy, and a P2P client. The proxy is transparent for all non-video requests. Video requests are routed to the P2P client. Any acquired data is sent back to the browser. On the server side, a single seed must be available to communicate with the P2P network. If traffic is light, the server may serve the client without any P2P benefit. However, with more simultaneous requests coming from multiple clients, the servers will benefit from clients interchanging data with each other.

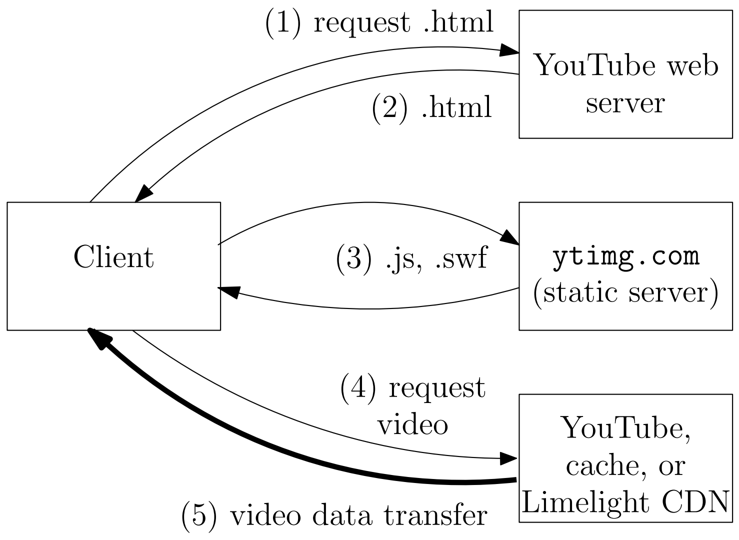

Figure 1.1: Requesting video content on YouTube. First, the .html web page is loaded. Then, the .swf Flash player and other small-sized content is loaded. Finally, data is transferred sequentially from a video web server. Usually, web pages are stored on a web server which is optimized for computations of dynamic content. ytimg.com is probably optimized for distributing small, often requested files. Videos are stored on a server optimized for data transfer of large static content.

1.1.2 Browser and P2P Client Integration

Some browsers support add-ons which provides the functionalities of a P2P client. An example is FoxTorrent [15] for the Firefox browser. The Opera browser has a P2P client built-in [16]. While this simplifies downloading of torrent files, there is still a disconnect between the web and the viewing of the content. The downloaded content is generally not viewable inside the browser. It is as if e-mails are downloaded through the browser to be viewed inside an external e-mail reading application.

LittleShoot

LittleShoot [17] is an example of a new browser with P2P technology built-in. It requires a background running LittleShoot client which handles the connections to the LittleShoot central server. The central server is used for finding content and peers. Actions performed from within the browser are sent through the LittleShoot client. One significant issue is in having to use the LittleShoot search engine in order to find P2P-assisted content. This content is not displayed in a web format and instead in an application format. This produces a disconnect between normal web content and P2P-assisted content.

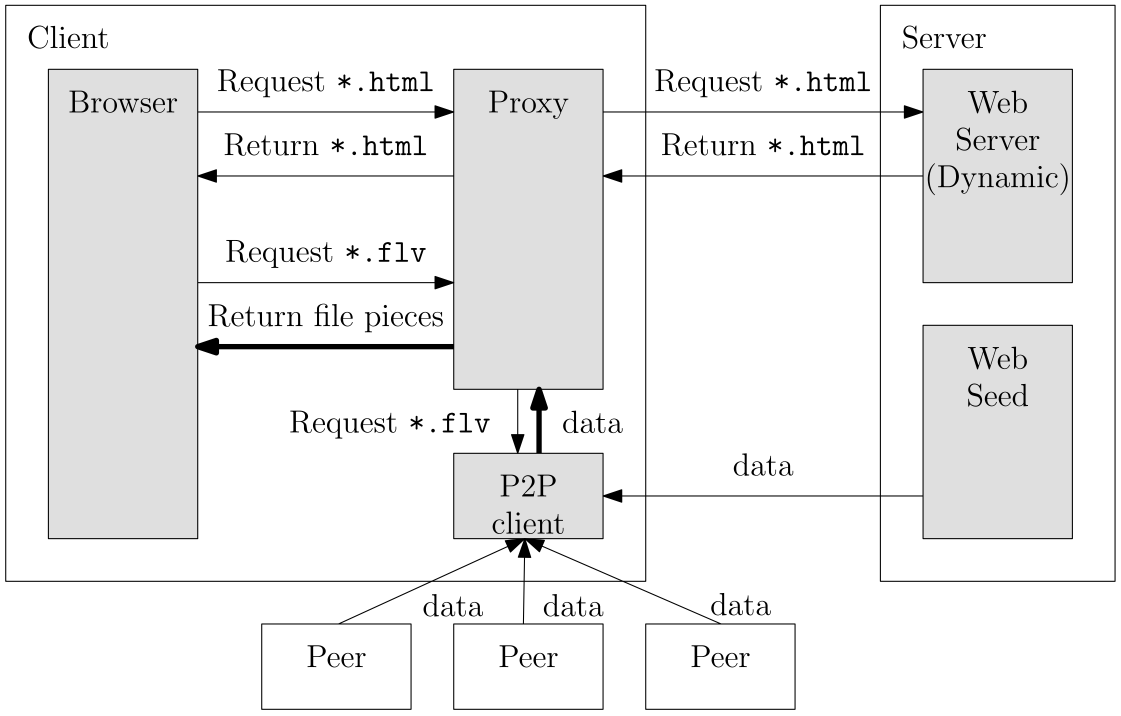

Figure 1.2: Requesting video content using a proxy and a P2P client. Requests to dynamic content or unknown content are forwarded by the proxy without manipulation. For servers that support web seeding, requests are routed to a P2P client which gathers data from the P2P network. Data is eventually sent back to the browser.

SlapVid

Slapvid (now obsolete) [18] was a P2P system used to view video inside a browser. A Java applet manages the delivery of content which is displayed within a Flash video player.

The first part of the video is downloaded from a central server. While this is taking place, the applet gathers nodes and initiates the P2P protocols. Subsequent pieces of the video are downloaded via the P2P network.

1.2 Youtube Characteristics

YouTube is essentially made up of three components: videos, users and the server. Videos are generally short and its popularity may vary widely. Users may watch videos at any time and in any order. The server has an infinite capacity for videos which can be accessible by all users.

1.2.1 Videos

Number of videos

The number of YouTube videos as compared to traditional video content is significant. A 2006 scrape of YouTube's website showed that almost 1.7 million videos[19] were available in the Entertainment category alone. Youtube uses 14 categories [20] to classify its videos. In May 2006, the number of new videos uploaded to YouTube totalled 50,000 a day, which jumped to 65,000 by June. At this rate, almost 25 million new videos would be available every year. It is not inconceivable that by 2009, YouTube would have over 100 million videos.

By contrast, according to the Internet Movie Database [5], there has only been about 890,000 movies and 745,000 TV episodes released all-time (up to April 2009)! See Table 1.6. Only a fraction of these videos are actually available from commercial video rental companies. Blockbuster[21], Zip.ca[22] and Netflix[23] advertises hosting at most 100,000 movies and TV shows.

Table 1.6: Number of traditional videos [5].

Type	Number
TV episodes	745,298
movies released theatrically	442,399
direct to video movies	73,738
made for TV movies	73,472
TV series	60,418

Popularity of videos

Using traces obtained by Cha et al[24], the distribution of popular videos is displayed in Figure 1.3. Large portions of the data exhibits a non-power-law nature. This is evidenced by the curve as opposed to a line in the log-log graph. Whereas power-law distributions are common for this type of phenomenon, it seems the distribution observed is that of a log-normal distribution.

Clearly, a small portion of videos receive the majority of views and a large portion of videos receive very few. The top 100 videos received more than 100,000 views each and the bottom 150,000 videos received less than 100 views each. This is important from a caching point of view as caching the popular files will produce very good results. From Figure 1.4, one can see that 80% of the views are generated by the top 10% of the videos. An additional 10% of videos are needed to generate another 10% of views. This is important to take into consideration for any model.

Figure 1.5 shows the complementary cumulative distribution function (CCDF) for the popularity of the videos. For this dataset, the cutoff for the top 10% of

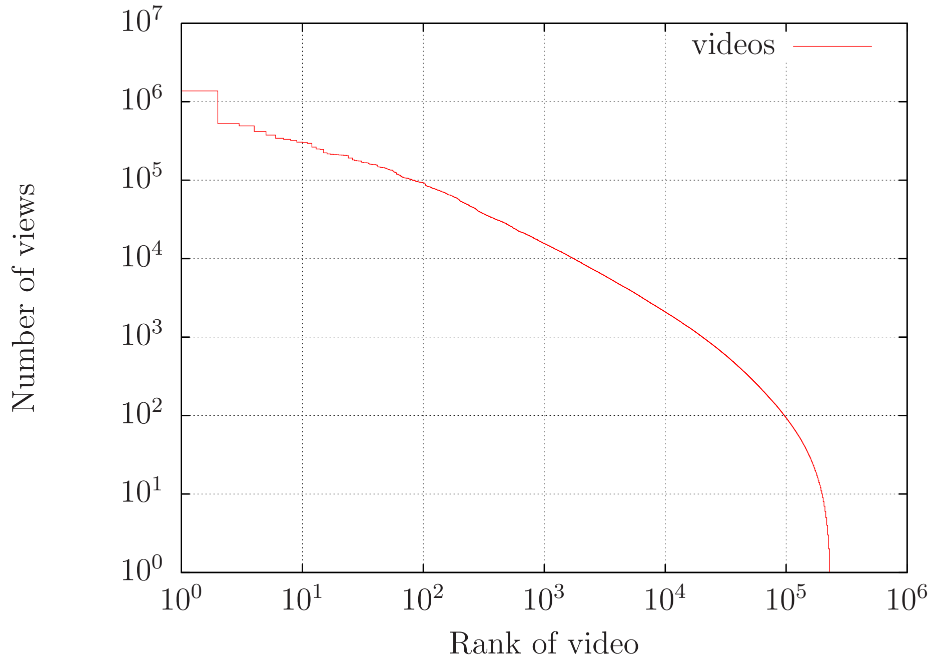

Figure 1.3: Number of views for videos in YouTube's Science and Technology category over 30 days (January 15, 2007 - February 14, 2007).

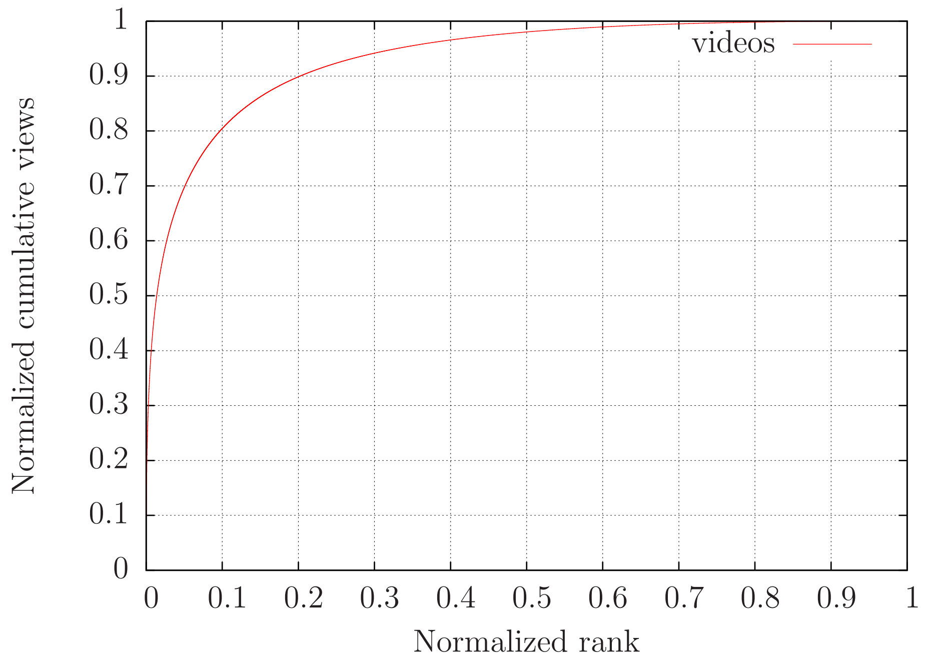

Figure 1.4: Integral of the number of views weighted by the total number of views for videos in YouTube's Science and Technology category over 30 days (January 15, 2007 - February 14, 2007).

videos is about 1000 views. Curve-fitting and focusing on these 10% of videos suggests a log-normal distribution with $\mu = 3.7$ and $\sigma = 2.26$:

$$P\{X \geq x\} = \frac{1}{2} - \frac{1}{2}\mathrm{erf}\left(\frac{\ln(x) - \mu}{\sigma\sqrt{2}}\right) \tag{1.1}$$

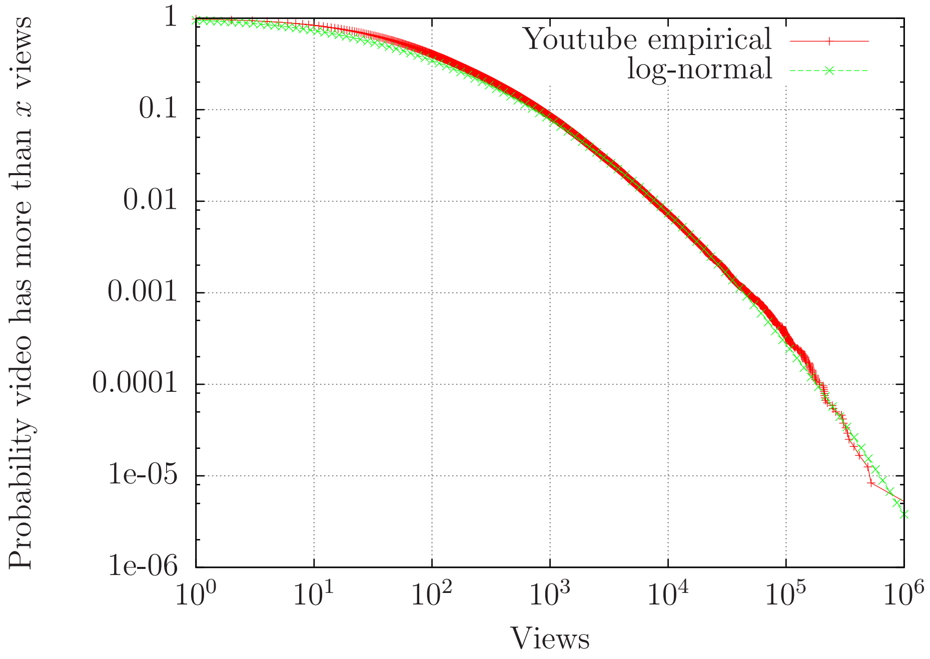

Figure 1.5: Complementary cumulative distribution function (CCDF) of the popularity of videos in YouTube's Science and Technology category over 30 days (January 15, 2007 - February 14, 2007).

Video duration, file size and bit rate

The majority of videos on YouTube have fixed bit rates. Therefore, the video duration, file size and bit rate can be described by the following formula:

$$\text{File size} = (\text{Video duration})(\text{bitrate}) \tag{1.2}$$

From Cheng et al's [25] YouTube traces, the average file size is 8.4 MB. The bit rate is concentrated around 330 Kb/s, with two other peaks at 285 Kb/s and 200 Kb/s [25]. The bit rate allows the majority of broadband users to watch the video without any delay during the duration of the video.

Bandwidth

Since videos can be accessed from anywhere and anytime, the server bandwidth can be considered to be infinite. However, local traces observed by Zink et al [26] showed that YouTube distributes videos at peak data rates of either around 700 Kb/s or 1200 Kb/s.

1.2.2 Users

YouTube does not provide significant information on its users. Some local network traces are available but show very limited information on all the users for any particular video. An example is the local trace conducted at the University of Massachusetts by Zink et al [26]. It is difficult to characterize all the arrivals for a particular video.

For these reasons a very simple user model is assumed. For every video, users arrive according to a random Poisson process. There is no correlation between the users of different videos. In reality, it may be important to characterize the likelihood for users to watch similar or related videos.

1.3 Scope

To simplify the analysis, users may cache at most one video. In addition, what each user has in the cache is not important. Instead, what is important is the probability of a video existing in the user's cache and the global cache at any particular instant.

1.4 Outline

Chapter 2 provides a background on content distribution networks and existing peer to peer systems. Chapter 3 introduces the model used in describing a peer-to-peer system for online video. Chapter 4 presents results and analysis. Finally, Chapter 5 concludes the study and Chapter 6 presents avenues for future work.

Chapter 2

Background

2.1 Content Distribution Networks

A Content Distribution Network (CDN) is a networked system whereby content, such as movies or photos are delivered to end users in an acceptable manner. The main goals of such a network are described below. How well a network tackles these challenges usually determine how popular the network becomes.

Scalable: The network should be *content scalable* in the sense that additional content can be added easily. The network should also be *user scalable* in that the load at the content provider should be inelastic as the number of users change [27].

Cost: The network should be cheap to setup. It should also decrease the content deployment costs for content providers and should be competitively priced for users.

Timely: The content should be delivered to users as quickly as possible. For a real-time delivery network, such as streaming video, data should be delivered before the users attempt to use it.

Usable: The act of announcing new or updated content should be easy for content providers. The act of searching and the delivery of content should be easy for users.

Easily deployable: Initial setup of the whole system to the stage where it is usable should be easy. It is much easier to deploy an application layer CDN, than a CDN which requires new routers or other changes to existing networks. The system may need to be backwards compatible with some existing system or use an existing network.

Robust: The CDN should be adaptive to changes in network conditions, network configurations, number of users, and so on. In other words, the system should be resilient.

Data integrity: The users must be confident that the downloaded content is the desired content. This includes both protection against malicious content and users, and non-malicious errors such as transmission errors due to a lossy network.

Security: This is a catch all phrase that is a combination of privacy, data integrity, authentication access control, anonymity, deniability and accountability. Depending on the CDN, some, all or none of these requirements may apply.

Fairness: This refers to the ability of the CDN to delivery content in a fair manner with respect to its users. No user should be starved of content.

Resource management: Content, storage, computational power, and bandwidth should be used efficiently in the network.

Highly available: Content should be easy to discover and should be available for downloading for as long as possible. Usually, availability refers to content as a whole; bits and pieces are generally not very useful.

Fault tolerant: The system should be accommodating when failures occur. This is particularly a problem of systems with centralized points of failures.

CDNs can be generally grouped into one-way content distribution and two-way content distribution. In one-way content distribution, there is a clear distinction between providers of content and consumers of content; content flows from the provider to the consumer. Most websites can be considered one-way content providers. CDNs such as Akamai [28], Limelight [29] and CDNetworks [30] are examples of one-way content distribution networks. These CDNs use a caching approach where popular content is located close to end users. In two-way content distribution, consumers also act as providers to other consumers. Peer-to-Peer (P2P) networks such as the very popular BitTorrent [31] and Octoshape [32] are two-way content distribution networks.

Content can either be consumed in real-time or when the download of the content has been completed in full. These two types of content delivery methods require different models and designs as they are bounded by different constraints. YouTube users watch videos as they are being downloaded. Thus, a P2P system would need to account for this.

2.1.1 Traditional CDNs

Most websites today operate on the traditional CDN model. A user makes a request for some content and through some cloud of logic, the CDN returns the content. Usually, this is done on the Application Service Provider (ASP) model, otherwise known as on-demand software or Software as a Service (SaaS) [33]. What this means is that companies pay CDNs for bandwidth and computational resources used for distributing content to its users. Traditional CDNs should not be confused with centralized CDNs, in fact most CDNs today are complex distributed systems. Various approaches are used by traditional CDNs to provide high performance. Two popular approaches are by using edge servers and load balancing techniques.

Edge servers are machines deployed by CDNs at the edges of a network where they are closest to users. Often, a CDN will replicate content at multiple edge servers separated geographically and logically. When a user makes a request, the closest edge server should respond. This reduces the amount of traffic going through the backbone of the network and through its interconnects [34]. Often, media assets are delivered with less delay and better throughput. The redundant nature of CDNs also allows CDNs to be resilient to network changes. Should a server go down, the CDN can automatically sense the change and redirect the user. Often times, CDNs offer services for multiple customers and can redistribute resources where needed. This benefits the end user in the form of availability of content.

Load balancing attempts to balance the incoming requests among a cluster of closely grouped servers. One way this is accomplished is by a front-facing application-layer switch, which delegates requests to any number of connected web servers. Another method is by Domain Name System (DNS) [35] round-robin which returns different network identifiers for different requests to the same service. Load balancing improves scalability, reliability and total capacity [34].

Traditional CDNs are generally content scalable but not user scalable. Content providers incur costs for every piece of content distributed. Deploying a new CDNis difficult, as it requires starting out with numerous servers at the same time. Content is usually transferred to users in a timely manner. CDNs are extremely easy to use for both providers and users. Traditional CDNs are quite robust to moderate network changes. However, because they do not scale with the number of users, flash crowds may cause problems. Data integrity is achieved by providing hashes of files on a trusted server. Content, regardless of popularity is highly available as long as the content provider deems necessary. Traditional CDNs are usually distributed and do not have single points of failure. However, should one point fail due to excessive traffic, it may cause the other mirrors to go down as well.

The majority of people access content and the Internet through web browsers. One reason why traditional CDNs are useful is because the popular web browsers do not have built-in mechanisms to connect to other browsers or other applications and so it would be very difficult to implement a peer-to-peer network within a browser.

2.1.2 Peer-to-Peer CDNs

Peer-to-Peer CDNs are systems where nodes (known as peers) download and upload content at the same time. The biggest benefit of P2P CDNs over traditional CDNs is the ability to scale as the number of users increases. This means that for content that become popular in a short period of time, P2P systems usually perform better. Conversely, data that is not popular sometimes become unavailable since nodes come and go as they please. Most peer-to-peer CDNs are standalone applications that do not work inside a browser. However, there are some peer-to-peer applications that make use of the Internet for locating content but rarely for distribution purposes.

One event often experienced by P2P networks is this notion of *flash crowds*. This is analogous to the more familiar event for websites known as the *Slashdot effect* [36] or the *Digg effect* [37, 38]. This event occurs when a very popular website such as Digg links to a not so popular website and the unexpected increase in traffic causes the site to be very slow or unaccessible. Similarly, in P2P systems, a flash crowd usually gathers when something extremely popular appears for the first time. This could be the latest release of the Ubuntu [39] operating system or perhaps bootleg media such as movies or music. When this occurs, the number of users participating in the download and upload of this content increases very quickly. Contrary to traditional CDNs, however, P2P systems experiencing flash crowds actually benefits from the upload bandwidth of each node. Although initially, the aggregate bandwidth achieved by the system may be low, the aggregate bandwidth continues to grow as peers finish downloading the content or parts of the content, as now these users begin uploading.

In any network requiring a specific topology, *churn* is an incredibly important parameter. Churn describes the joining and leaving of nodes. Usually, peer-to-peer networks experience high levels of churn that is also unpredictable. Networks with rigid topologies experience high overhead costs associated with reconfiguration when nodes are short-lived in the system. To this end, some peer-to-peer network topologies are random, in the sense that there is no specific order required. This has been shown to do well in content distribution network settings.

An in depth study of peer-to-peer CDNs is provided in Section 2.2 on page 15.

2.1.3 Other CDNs

Multicasting and CoopNet are examples of other content delivery systems.

Multicasting

Most CDNs rely on unicast data streams, where data is sent between pairs of nodes. Multicasting on the other hand allows data to be sent between members of a larger group simultaneously. Similar to peer-to-peer applications, there are numerous application layer multicasting schemes [40, 41, 42]. There are also network layer multicasting schemes [43].

Efficient multicasting usually requires a good network topology such as a shortest path tree [27]. However, this suffers from high overhead in constructing the topology. The system also suffers when nodes experience high churn, requiring the network to reconfigure itself.

CoopNet

CoopNet [44] is an interesting academic CDN. Its intended application is in allowing nodes to act as temporary caches for web sites. When a user visits a webpage, the user is redirected to past users that have previously downloaded the file. One limitation of CoopNet is requiring the web server to keep track of the past users and the content that they may have. Also, only full file downloads are facilitated which limits the size of files delivered.

2.2 Peer-to-Peer Systems

Peer-to-Peer (P2P) systems are an emerging area of research. Two excellent surveys on the topic are presented in [45] and [46]. This section presents a discussion of peer-to-peer systems and attempts to explain why a peer-to-peer scheme makes sense for content distribution applications.

2.2.1 What is P2P?

Although numerous definitions have been proposed and are pliable in different contexts, a good definition is provided by Androutsellis-Theotokis and Spinellis [45].[1] A less restrictive definition is employed by this manuscript: *A peer-to-peer system is a networked system of interconnected nodes which is capable of resource exchange on a node-to-node basis without the assistance of other nodes.* Note that the only restriction of peer-to-peer systems over other networked systems is that the resource exchange itself must be performed on a node to node basis without any external help. Setting up the network, updating the network, or even finding out where content is may be assisted by a central authority.

A major assumption of peer-to-peer systems is that nodes have excess, available resources. The five major P2P resources are: bandwidth, content, computational, storage and human.

Bandwidth: Each node has excess upload and, or, download network bandwidth. The excess upload capabilities of nodes are a basic requirement of P2P systems.

Content: Each node has content that other nodes may not have. This is another primary requirement of P2P systems in that nodes may have something that other nodes want.

Computational: Each node has excess computational resources such as CPU cycles. This is a secondary requirement since the computational requirements of a basic P2P system is low. This may be used to segment files into pieces or calculating hashes for data integrity.

[1]Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, Central Processing Unit (CPU) cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority. [45]

Storage: Each node has excess temporary and, or, long-term storage capabilities. Short term storage may be used to store network state or pre-computed values. Long-term storage can be used to store content which may be consumed at a later time or uploaded to others.

Human: Each node usually consists of client software that is operated by a human. A human may have out-of-band knowledge about the network. An example is knowing that another node (perhaps a friend) is geographically closer, so connecting to this node may provide a faster download. A human may also make decisions where it may be difficult for computers. Security applications may use mouse movements as a source of randomness.

Depending on the objectives of the P2P network, resource tradeoffs may be possible to achieve those goals. By contrast, a user in a traditional CDN usually only needs to use download bandwidth and temporary storage.

2.2.2 Classification of P2P Systems

Androutsellis-Theotokis et al in [45] classified P2P applications into five major categories: communication and collaboration, distributed computing, Internet service support, database systems and content distribution.

Communication and Collaboration: These applications allow users to communicate with each other. Examples include Internet Relay Chat (IRC) and Extensible Messaging and Presence Protocol (XMPP). Skype [47] allows people to make calls over the Internet via Voice over IP (VoIP).

Distributed Computing: These applications allow computations to take place over many nodes all at once. Nodes may participate in delegating jobs, doing computations and, or, combining results. Examples include protein folding (Folding@home [48]) and the Search for Extraterrestrial life (Seti@home) [49].

Internet Service Support: These applications provide infrastructure to support various Internet services. Examples include peer-to-peer multicast systems [42], security systems that protect against denial of service [50] or virus attacks, and Internet search [51].

Database Systems: These applications act as a distributed storage system which offers efficient search, modification and deletion. A good overview of peer to peer database systems is provided by Ryeng et al [52].

Content Distribution: These applications allow nodes to deliver content to nodes that desire the content. Content distribution is the focus of this manuscript and will be discussed in the remainder of this chapter.

2.2.3 P2P CDN

Peer-to-peer content distribution networks are by far the most popular applications of peer-to-peer technology. It is a system where the primary objectives are to facilitate *publishing*, *searching*, and *transfer* of content by members of its network [45]. All of these networks are application-level *overlay networks*. In other words, these networks overlay a logical network on top of the existing web. Thus, two nodes may be close in the P2P network and yet be far apart physically.

In [46], the network architecture is categorized as either *structured* or *unstructured* networks. Structured networks follow a specific set of rules when the network is being setup or updated. Unstructured networks on the other hand, do not follow such rules and usually exhibit a random graph. Structured networks usually place pointers to content in nodes at deterministic locations. This way, queries for content can be efficiently routed [45]. In unstructured networks, however, there is no relationship between content and node.

Peer-to-peer networks that utilize centralized servers are called *hybrid* networks (Figure 2.1a). Networks where ordinary nodes may have centralized roles are called *partially centralized* networks (Figure 2.1b). A network where all nodes are equal and does not have a centralized component is *purely decentralized* (Figure 2.1c). Unstructured networks usually employ a hybrid or partially decentralized architecture to allow for efficient queries. Structured networks are usually used in purely decentralized environments.

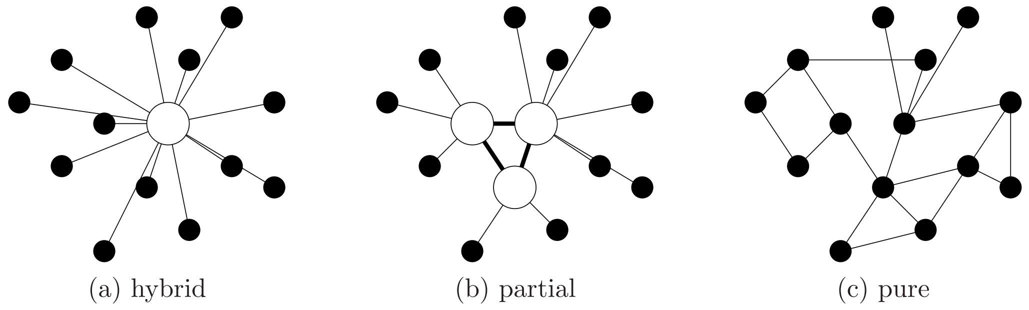

Figure 2.1: P2P CDN architectures. The black dots represent peers. The white circle in (a) represents a centralized server. The white circle in (b) represents superpeers.

2.2.4 Definitions

Some important P2P terms are described below. More terms are described in the Glossary.

Peer: An endpoint in a P2P system which usually consists of a client application connected to the network. A peer is usually capable of upload, download and routing network information. Synonyms include: nodes, users, leaves and servents.

Leech: A peer which only participates for personal gain and does not contribute to the network. A synonym of leech is freerider.

Seed: A peer which is only contributing to the network.

Superpeer: A peer with high bandwidth, disk space and processing power, that have additional centralized responsibilities. Usually superpeers store network state which allows search queries to be performed more efficiently. Synonyms include supernodes, ultrapeers and hubs.

Neighbour: Peers usually keep a list of all the nodes it knows to be in the system. These nodes make up the neighbours of that peer.

Churn: Churn refers to the dynamics in which nodes join and leave the network [53]. A high rate of churn means that nodes stay in the system for only a short period of time.

Flash Crowd: A flash crowd usually appears when a popular piece of content is released. During a short period of time, the network needs to handle many more nodes and an increase in bandwidth for the desired content.

DHT: A Distributed Hash Table is used to place pointers to content at deterministic locations in a network. The aim is to efficiently route queries.

Pollution attack: Malicious nodes may perform a pollution attack by publishing lots of seemingly authentic, but garbage content into a network with the hope of making it difficult for users to determine the real content.

Poisoning attack: A poison attack is used to make downloaded content unusable by modifying various aspects of the system. For example, by modifying the central search index of a network, users may be redirected to malicious nodes.

Sybil attack: A Sybil attack can be deployed by having multiple nodes work together to disrupt a network [54]. For example, malicious nodes can surround a victim node and deny that node access to the network. It is not necessary to perform the attack using multiple entities. Appearing to be multiple identities is enough.

2.3 P2P CDN Applications

The following discussion presents a brief survey of some of the most popular P2P CDN schemes of the past and present. The applications will be analyzed in terms of system architecture, the actions of nodes, and how content is distributed. The purpose of this chapter is to arrive at a set of basic requirements for a good P2P CDN.

2.3.1 Napster

Napster [55] was one of the first peer-to-peer file sharing systems, and probably the most notorious. It was created by Shawn Fanning while he was a student at Northeastern University in June, 1999 [56].

The original Napster architecture was a hybrid decentralized network using a central server to handle all nodes and queries. Users wanting to participate in Napster had to download a Napster client application. To join the system, the software would connect to the Napster servers which had known, static addresses. Then, the software would tell the central server what files a user is sharing and their contact information [57].

A user may search the central server to find out who had a desired file. Then, the user can select a single node from the central server's response and connect to the node directly. Data transfer takes place peer-to-peer between the two nodes independently of other nodes and the central directory. This is illustrated in Figure 2.2.

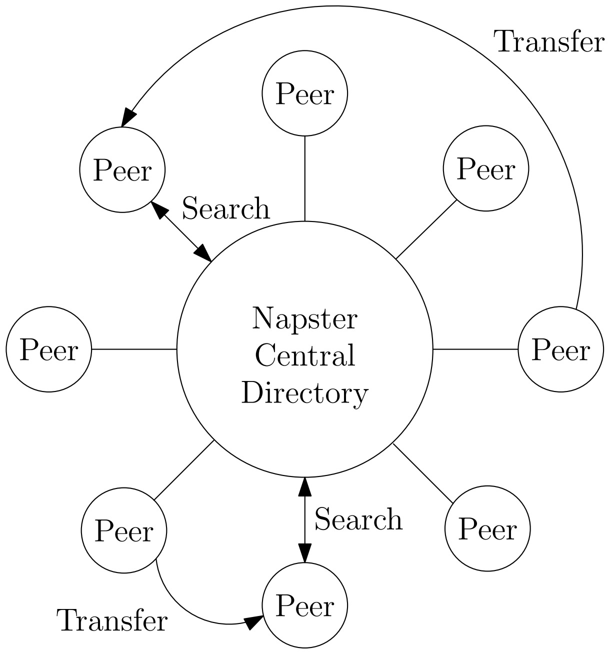

Figure 2.2: Napster Architecture. The central directory keeps track of all the nodes and their offered content. Search requests are sent to the central server. Transfer takes place peer-to-peer.

Using Napster, a user had no way of verifying the authenticity of a file without actually downloading it and many files labelled with the same file name could have content ranging from the desired, to garbage, to malicious content. As such, Napster suffered from pollution attacks.

One of the major flaws of Napster was that the central server was absolutely required in the operation of the system and represented a single point of failure. When the Recording Industry Association of America (RIAA) wanted to shut down Napster due to copyright infringement of music content, all it had to do was shut down its central servers. By late 2001, Napster was no more.

2.3.2 Gnutella

Gnutella [58], pronounced *newtella*, is a peer-to-peer protocol developed by Justin Frankel and Tom Pepper in early 2000 while working at Nullsoft [59] and subsequently AOL [60]. The original Gnutella (version 0.4) in contrast to Napster, was purely decentralized, meaning that all nodes could perform the same actions and have the same responsibilities. There was no centralized server whatsoever.

The purely decentralized nature of Gnutella presents a problem for nodes that want to join the network. Because nodes come and go, where would a new node connect to? This process of finding at least one node to connect to is known as *bootstrapping* [61]. Bootstrapping can be performed by acquiring a list of nodes previously known to be in the system. This could be nodes that are known to stay in the system for a long time, a list of nodes previously connected to, or a list of nodes that others publish in a public area such as an online forum [62]. Once a node has connected to a single node in the system, the node can query the network to find the addresses of other nodes. Nodes may leave the system at any time. The remaining nodes should detect the loss and update their neighbour lists accordingly.

Unlike Napster, users do not publish content to any other node. It only keeps a list of files it is willing to share. Search is accomplished by broadcasting a query to all neighbouring nodes. Each of these nodes continue to broadcast the query to all of its neighbours. This process continues until a Time to Live (TTL) limit attached to the query, goes to zero. The TTL is a value set by the initiating node which each node must decrement when the query is received. This process of searching is known as a *flood*.

Each node queried is required to respond if it has the requested content. The responses come back in the same path that the query took in the reverse direction. Once the responses come back to the original node, the user application makes a direct connection to the node with the desired content. Data transfer takes place peer-to-peer. An illustration of a file download is shown in Figure 2.3.

Because of its decentralized configuration, it would be much more difficult for the RIAA or the government for that matter to take down the network. However, Gnutella suffers from other shortcomings. A node that is not actively downloading still needs to participate in search request propagation and the search mechanism itself is very inefficient. In addition, due to the use of TTL, a search will stop before querying the entire population of nodes which may prevent finding certain rare files. To help alleviate these problems, version 0.6 of Gnutella introduced *ultrapeers*.

Ultrapeers are peers that have more resources than the average peer. Slower peers, known as *leaves* in Gnutella, perform all of their actions through one or many ultrapeers. Leaves let their ultrapeers know what they are sharing and all queries are routed through the ultrapeers. This way, only ultrapeers participate in the flooding mechanism [63]. Ultrapeers also keep a cache of nodes in the system and search results.

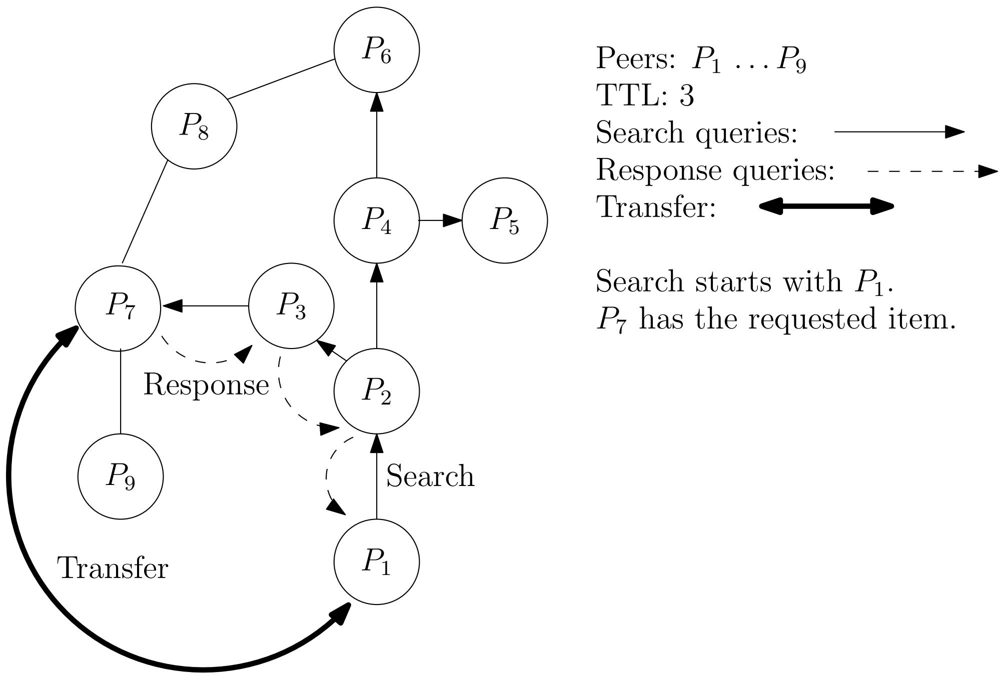

Figure 2.3: The Gnutella network uses a purely decentralized approach. In this example, the search begins at P_1 which sends out a search request to all of its neighbours. The time-to-live is set to 3. P_2 decrements the TTL to 2 and forwards the search request to all of its neighbours. P_2 also searches its own file list and cannot find the file. This process continues until TTL is 0. The only node responding is P_7 and the response follows this same path as the requests. The data transfer takes place peer-to-peer between P_1 and P_7.

Another improvement of version 0.6 was that search queries contained the IP address and port of the node that initiated the request (or the ultrapeer that is representing the node). Using this information, query responses can take a shortcut on the way back so that responses go directly to the original node (or one of its ultrapeer).

As with Napster, there is no way to verify the authenticity of a file without downloading first. Also, like Napster, there is no incentive for users to offer any files at all. However, unlike Napster, Gnutella has much better fault tolerance since there is not a single point of failure. If a particular node goes down, the network does not suffer too much.

Examples of software applications that operate on the Gnutella network includes: BearShare [64], Gnucleus [65], Limewire [66] and Morpheus [67].

2.3.3 FastTrack

The FastTrack [68] protocol and its software implementation, KaZaA [69] was created by Niklas Zennström, Janus Friis, and a group of programmers headed by Jann

Tallinn in March, 2001 [68]. FastTrack is a hybrid decentralized, unstructured P2P system very similar to Gnutella with ultrapeers. It uses a two-level hierarchy of peers where the core is made up of a mesh of superpeers and the edges are normal peers. These superpeers together act similarly to Napster's central directory; they keep track of what files peers have and where the peers are. Normal peers connect to the network through the superpeers. Superpeers can also query for, and download content like a normal peer. This is illustrated in Figure 2.4.

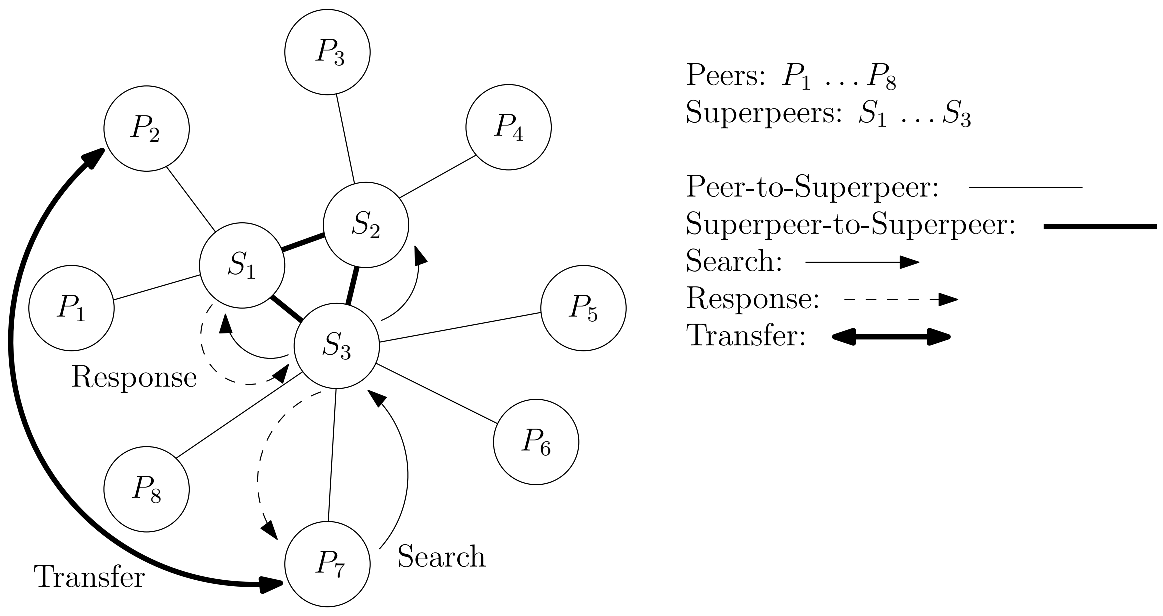

Figure 2.4: FastTrack Architecture. Queries are sent to a superpeer which communicates with other superpeers to arrive at a response. Transfer takes place peer-to-peer.

Using KaZaA or some other FastTrack application, a user can join the network through a list of known supernodes. This list is created out-of-band; it could come with the software, the software can keep track of previously connected superpeers, or by some other method. Since this is an unstructured system, normal nodes leaving the system does not affect the system much. When superpeers leave, its peers must find other superpeers.

To publish content, nodes must inform the superpeers what files they are willing to share. To perform a search, a peer must contact a superpeer. The superpeer can then use a broadcast protocol to query the other superpeers. Once a response comes back, data transfer takes place peer-to-peer.

FastTrack uses the UUHash [70] hash function to hash files. A hash provides three benefits. First, a hash can be used to uniquely identify a file. Second, users can download pieces of a file from multiple users which have files with the same hash. This may shorten the download times. Finally, it can be used to verify a download by computing the hash of the downloaded files. Unfortunately, UUHash was a bad choice for a hash function because it was not collision resistant. In other words, it was very easy to find two different files with the same hash so FastTrack suffers from pollution attacks where fake or malicious files are mixed together with desirable files.

Other than KaZaA, iMesh [71] and Grokster [71] also operated using the FastTrack network, although all three used incompatible versions of the protocol [68]. Grokster was shut down in late 2005 [72].

eDonkey

The eDonkey network is a hybrid decentralized, unstructured, peer-to-peer file sharing network. Whereas previous peer-to-peer networks focused on downloading small files from single users, eDonkey was designed for downloading large files from a lot of users at the same time. The network used central servers to facilitate the sharing of files among the peers, much like Napster. The eDonkey2000 client was created by the MetaMachine Corporation in September, 2000 [73].

Each shared file on the eDonkey network was broken into 9500kB sized pieces which allowed users to download pieces from many users simultaneously [73]. A MD4 hash was computed for each piece to allow for piece-wise verification. Downloaded pieces can also be uploaded to other users.

A master, or root hash could then be computed by calculating the hash of the concatenation of all the piece hashes. This was used as the unique identifier for the file. Other secondary identifiers includes: file name, file size, extension, bitrate, codec and so on. Although MD4 is no longer a secure cryptographic hash function, it is still much better than the UUHash function in terms of being collision resistant. For most practical purposes, creating collisions using MD4 is still very difficult.

Although it is difficult to create two files with the same identifier, it is quite easy to name fake files with real-looking secondary identifiers. Many fake files can be generated to deploy a pollution attack. To alleviate this problem, various websites listed the root hash of files and by the help of its members, good identifiers could be determined. Users can search the website for a file and receive an unique identifier which can be used to search the actual eDonkey network for download. As long as the website and its users are truthful, users can download files with very high guarantees that the file is legitimate.

One of the major problems of the eDonkey network was the use of central servers. Two new schemes were devised on top of the eDonkey network which employed DHT technology. This allows a network to continue operating even when central servers fail. MetaMachine created Overnet by implementing the Kademlia algorithm. Furthermore, the eMule Project developed a Kademlia-like network called Kad which was added in version 0.4. A discussion of the Kademlia DHT is provided in §2.3.6.

By mid-2005, the eDonkey network had about two to three million users, 500 to two billion files and about 100 to 200 central servers [73]. Razorback2 was one of the most popular servers which hosted about a million users but was shut down in February, 2006 [73]. In September, 2006, MetaMachine discontinued the eDonkey2000 user client due to RIAA pressure. However, by this time, eMule and Shareaza [74] had already accounted for over 90% of the client software used and so the network is still very popular today [73].

Some example of eDonkey user applications include eDonkey2000 and eMule. MetaMachine originally wrote the first server software, which are not in use anymore. Lugdunum's eserver currently supports most eDonkey type networks [73].

2.3.4 eMule

eMule was created by someone with the alias Merkur (also known as s) in May of 2002 as an alternative to the eDonkey2000 client [75]. Using the same identification scheme as eDonkey, the unique MD4 root hash and secondary attributes were saved on both the eDonkey servers and the Kad network. Either of these networks could be searched and a list of node addresses is returned. eMule can then request pieces of the file from multiple nodes. Every node has a limited number of upload slots and implements a queueing system so that downloaders must wait until an upload slot becomes available [75] before being able to download. This is usually based on a first-come first-serve basis.

Low ID

eMule makes a distinction between users that can accept incoming connections and those that cannot. The users that cannot be reached by other nodes are given a *Low ID* from the servers. These users cannot be notified when another node is ready to upload to them, so they need to use periodic polling. This produces additional strain on the network. In addition, other nodes cannot easily download from a Low ID user and will need help from servers or other Kad clients.

A Credit System

eMule is one of the first peer-to-peer applications to consider a credit system which provides incentives for nodes in the system to upload to other nodes. Prior to this, a user can offer no files or bandwidth to a network and still be able to download. By using a credit system, users who have uploaded and have contributed to the network are given higher upload slot preference than users who are *freeriding*. The credit system is peer-to-peer, so that if a node A is uploading to a node B, then, node A will be given higher preference on node B's upload queue.

The credit score of a node is calculated as follows. If the total uploaded is less than 1 MB, then the ratio is fixed at 1. If the client uploads data but has not begun downloading, the credit score will be 10. Otherwise, the credit score is the lowest of Ratio1 and Ratio2. The credit is a value between 1 and 10.

$$Ratio1 = \frac{2(UploadedTotal)}{DownloadedTotal} \tag{2.1}$$

$$Ratio2 = \sqrt{UploadedTotal + 2} \tag{2.2}$$

Trusted peers

eMule also takes into consideration trusted peers. One example of this is friends who can be given a reserved upload slot. However, only one reserved slot is allowed to prevent nodes from uploading only to friends.

2.3.5 BitTorrent

Architecture

BitTorrent [31] is another popular peer-to-peer file sharing system, typically used to share large, popular files. By some estimates, BitTorrent accounts for about a third of all Internet traffic [76]. It was created in 2001 by Bram Cohen and is now maintained by BitTorrent Inc [31].

BitTorrent is a hybrid decentralized, unstructured file distribution network. A typical BitTorrent network consists of three components: a web server that publishes .torrent files, trackers that keeps track of nodes in a swarm and individual nodes that actually participate in the data transfer. The original BitTorrent architecture is shown in Figure 2.5. Nodes are grouped by the content they are sharing into what are called *swarms* which is a key difference between BitTorrent and other P2P networks. A user who is downloading two different files is a member of two different swarms.

All nodes periodically poll one or more central trackers to find out the location of other peers in the swarm. This way, a new node such as P_1 in Figure 2.5 can connect to existing nodes in the swarm. Among other things, the tracker's location is provided in a .torrent file which may be obtained from a website such as The Pirate Bay [77]. A sample .torrent file is given in Appendix A. Once the .torrent file has been downloaded, a BitTorrent client such as uTorrent [78] or Azureus [79] handles all BitTorrent protocol related tasks and the data transfer.

Each piece of content, such as a file, is broken into many smaller equal-sized pieces. However, unlike eDonkey, piece size is not fixed since it was determined that there is a tradeoff between efficiency and overhead when choosing the size. Peers are able to download and upload these pieces from multiple peers simultaneously and data transfer takes place peer-to-peer. *Seeds* are those peers that have completed the download and are offering their upload bandwidth to other peers.

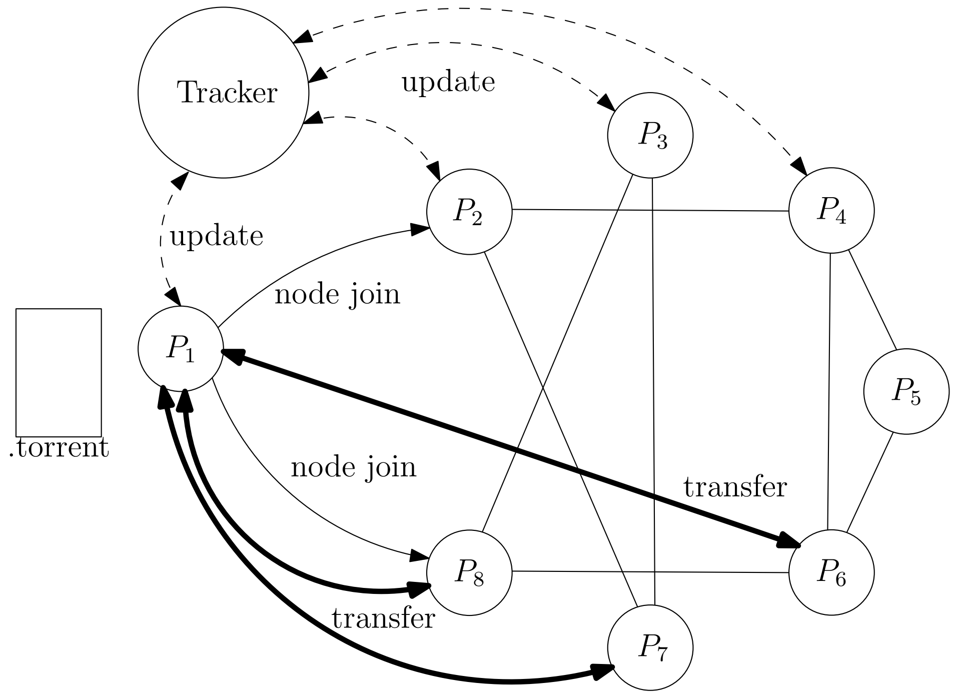

Figure 2.5: Original BitTorrent Architecture. Peers sharing the same file are combined into a swarm. To find other peers in the swarm, each peer must periodically update with a central tracker. The location of the tracker is contained within a .torrent file which a user may obtain from a website. Data transfer takes place peer-to-peer and pieces of a file may be obtained from different peers simultaneously.

A snippet of a BitTorrent transfer is presented in Figures 2.6 and 2.7. In the first round (Figure 2.6), peers P_1 and P_2 each download a different piece from seed S_1. In the second round (Figure 2.7), peers P_1 and P_2 are able to share pieces d_1 and d_2 with each other. However, because they both download the same piece d_3 from the seed, they cannot share pieces with each other in the next round.

Publishing Files

To share a file, the initial seeder must create a .torrent file. In addition to the tracker information, a .torrent file also contains metadata about the file. To prevent corruption of data pieces, a SHA1 hash is calculated for each piece and stored in the .torrent file. This way, a node can download a piece from any untrusted peer and verify the validity of the piece. The initial seeder usually publishes the .torrent file on a trusted website such as The Pirate Bay [77] and registers it with a tracker.

Unfortunately, this model suffers from the same fault tolerance issues as with other centralized networks such as Napster. In 2005, to combat this issue, BitTorrent introduced *trackerless* torrents based on the Kademlia protocol [80]. Using Kademlia, every node acts like a tracker and the network can operate even with-

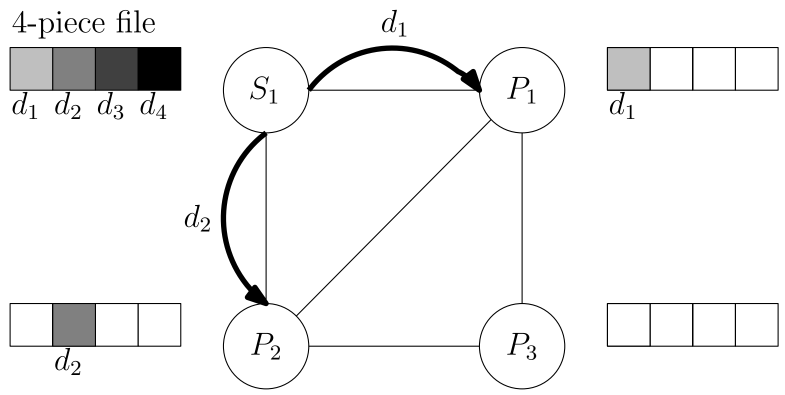

Figure 2.6: Example BitTorrent data transfer: Round 1. A particular file or group of files is broken into pieces (in this example four pieces). Here, a seed S_1, is distributing content to peers P_1 and P_2. In particular, P_1 receives piece d_1 and P_2 receives piece d_2.

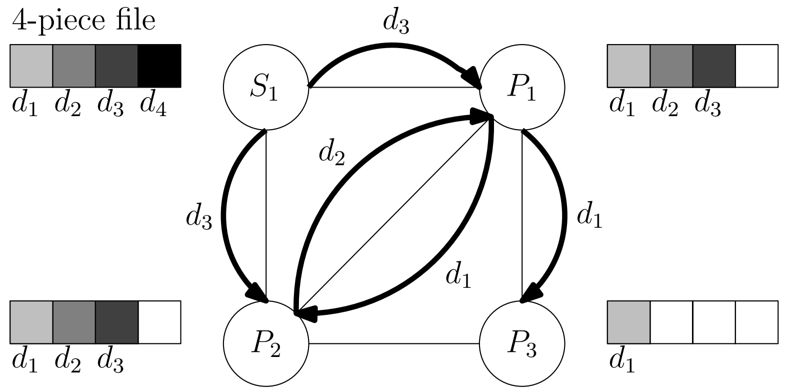

Figure 2.7: Example BitTorrent data transfer: Round 2. Peers P_1 and P_2 both downloads d_3 from the seed. In addition, the two peers transfer their unique pieces to each other simultaneously. P_1 also uploads to P_3.

out a central tracker. More details about Kademlia and its use in BitTorrent are given in §2.3.6. In addition to the DHT method, Peer Exchange (PEX) has been implemented to allow peers to share peer lists.

Obtaining Files

To download a file, a user must first download the .torrent file. Then, the BitTorrent client connects to the tracker or trackers found in the .torrent file. The tracker returns a random subset of all nodes participating in the swarm. The client then connects to these peers directly to begin data transfer.

All peers broadcast the pieces they have to their neighbours so that peers can determine which pieces are available for download. A peer can choose to download any piece that it does not already have. Once all the pieces have been obtained, the original file is reconstructed by concatenating the individual pieces.

Incentive to Upload

To shorten downloads, BitTorrent aims to maximize the aggregate bandwidth used in the network. This is accomplished by utilizing the maximum upload bandwidth (or download bandwidth, depending on how you look at it) of each individual node. However, a node only wants to maximize its own download bandwidth.

To provide an incentive for nodes to upload, BitTorrent implements a *tit-for-tat* strategy in its clients. This is a rounds-based strategy which is often used in the *Iterated Prisoner's Dilemma* game theory problem.[2] It is devised so that uploading provides better chances for a faster download.

For a node participating in an exchange with another node, it will always upload in the first round. On subsequent rounds, it will mimic the action taken by the other node in the previous round. Thus, if two nodes upload to each other, they will continue to upload to each other. If one node stops uploading, the other will too. Periodically, a peer will offer to upload to a random peer. This is called *optimistic unchoking* and allows search for more cooperating peers and give a chance to previously non-cooperating peers [82].

Piece Distribution

BitTorrent is a bulk-content CDN, which means files are used after every piece has been downloaded. From an availability point of view, it is necessary that every unique piece is available. Otherwise, a node cannot complete a download.

Downloading rare pieces offer the best opportunity to upload to others since most nodes will want the piece. This in turn allows nodes to utilize its download bandwidth. This is known as the *rarest-piece-first* strategy.

Globally, rarest-piece-first reduces the number of rare pieces in the system so that the distribution of pieces in the system becomes even. If rare pieces were part of the system, it would severely damage the performance of the transfers, since a lot of nodes will want to utilize the upload bandwidth of a few. Should certain pieces totally disappear, the transfer would not be able to finish. Heterogeneity in the system also makes the system robust in that peers can enter or leave as they

[2]The Prisoner's Dilemma problem was first created by Merrill Flood and Melvin Dresher working at RAND Corporation in 1950. It is usually presented as follows: Two suspects are arrested by the police. The police have insufficient evidence for a conviction, and, having separated both prisoners, visit each of them to offer the same deal. If one testifies (defects) for the prosecution against the other and the other remains silent, the betrayer goes free and the silent accomplice receives the full 10-year sentence. If both remain silent, both prisoners are sentenced to only six months in jail for a minor charge. If each betrays the other, each receives a five-year sentence. Each prisoner must choose to betray the other or to remain silent. Each one is assured that the other would not know about the betrayal before the end of the investigation. How should the prisoners act? [81] The iterated version of this problem has both suspects being arrested over and over again and remembering what the other suspect did last time they were in jail.

please with minimal effect on the status of rare pieces because hopefully other peers would have them.

In many situations, the original seeder of a file will at some point leave the system. It is necessary that this seeder must upload a copy of the original file before leaving. Once this is done, all the other peers must be able to share what they have, and eventually distribute the data to everyone else in the system.

Freeriding

A universal problem with all peer-to-peer systems is the freeriding or leeching problem. This problem occurs when a node only wants to download. Tit-for-tat discourages freeriders during downloads, however, as soon as the download is finished, a user may quickly disconnect from the network. An attempt has been made by private trackers such as Demonoid [83] to keep track of upload and download ratios on a per user basis. This way, users who have low upload ratios can be punished.

Another strategy employed by seeders is to withhold pieces until peers have a good enough upload ratio. Recently, there has been attempts to show that freeriders can achieve a good download rate with new clients such as BitThief [84] and BitTyrant [85].

Other applications

One novel enhancement of BitTorrent has been the attempts to bridge the gap between the browser-based Internet and the standalone P2P client applications. *Web seeding* allows content providers to source content from the Internet that can be accessed through a traditional swarm. This allows websites to offload bandwidth to downloaders to handle more traffic than otherwise possible. However, content is downloaded by the BitTorrent client application and does not yet pose a serious threat to traditional CDNs which work within the browser.

Another technique called *broadcatching* allows client applications to capture RSS feeds which can then be used to initiate a torrent download automatically. Miro [86] is an example of a client supporting broadcatching.

Besides traditional P2P client applications, BitTorrent is being increasingly used in different ways. TorrentRelay [87] is a service that will download a torrent for a user and then allow the user to download from it directly. It seems likely that TorrentRelay will become an easy target for anti-piracy groups. Also, various hardware devices such as routers have been designed to support BitTorrent natively.

Privacy

BitTorrent does not offer any anonymity. A tracker maintains a list of all Internet addresses that is participating in a swarm. Anonymity comes at a great price since it usually requires sacrificing performance.

To prevent eavesdroppers and malicious attackers from interfering with a BitTorrent transfer, Protocol Header Encryption (PHE), Message Stream Encryption (MSE) and Protocol Encryption (PE) can be used by BitTorrent clients. This allows users to download content that may otherwise have varying consequences. Recently, since BitTorrent traffic makes up a significant portion of Internet traffic, Internet Service Providers (ISPs) such as Comcast has attempted to limit the bandwidth used. Even though encryption can make it difficult to determine the exact content being shared, it is still possible to use traffic analysis to discover BitTorrent traffic.

2.3.6 Kademlia

Kademlia is a peer-to-peer distributed hash table. It was created in 2001 by Maymounkov and Mazières [80] and is currently used in the popular BitTorrent and eDonkey/eMule P2P networks. Distributed hash tables provide a decentralized system to efficiently store and locate ⟨key, value⟩ pairs among the member nodes. Usually, the *key* represents a unique identifier for some content, and the *value* represents a pointer, or a list of pointers to the nodes that have the content. This allows P2P networks to be fully decentralized and is a way to ensure high availability, robustness and improved fault tolerance.

System Architecture

A Kademlia network consists of nodes each with (for all practical purposes) unique 160-bit IDs. IDs are generated randomly by the nodes themselves. Logically, the IDs represent leaves in a binary tree with the position determined by the shortest unique prefix of its ID. Consider the example given in Figure 2.8 for the node with unique prefix 0011 borrowed from Maymounkov's paper [80].

To locate a particular node in $O(logn)$, where n is the number of nodes, it is necessary that every node knows of at least one node in each of its subtrees. This way, searching for a particular node is similar to a binary search. Keep in mind that each node will have a different partition of the binary tree.

Distance

In Kademlia, keys are also 160 bits in length. Therefore, keys and node IDs occupy the same identifier space. In fact, ⟨key, value⟩ pairs are stored at nodes with node IDs closest to the key. Therefore, searching for a particular key amounts to searching for a node with the same identifier as the key. To measure distance between two identifiers, Kademlia uses the exclusive-or (xor) bit-wise operation. For two identifiers x and y, their distance $d(x, y)$ interpreted as an integer is given by:

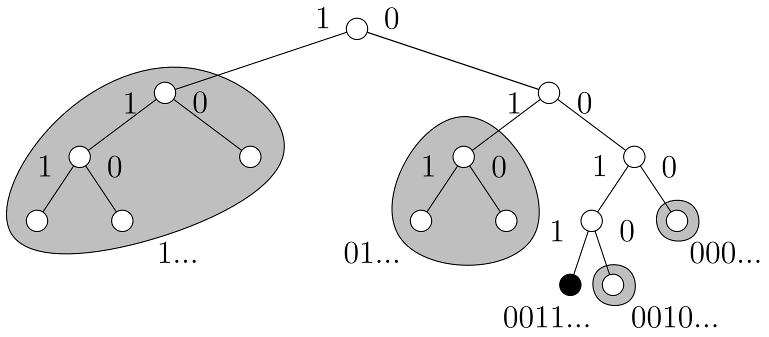

Figure 2.8: Kademlia system architecture as seen from the node with prefix 0011.... The binary tree is partitioned into subtrees that do not contain the node. All nodes with prefix 0010 do not contain the node with prefix 0011 since the fourth bit differs. Also, all nodes with prefix 000 do not contain the node since the third bit differs, and so on. Thus the subtrees for a node with unique prefix 0011 in this example are those with prefix 1, 01, 000, and 0010.

$$d(x, y) = x \oplus y \tag{2.3}$$

Kademlia interprets keys as bigendian values [88]. Therefore, for a 20 byte key $B_1B_2 \dots B_{20}$, B_1 represents the most significant byte. If two keys are close, then the distance in bit format should start with a long string of 0s. In other words, if two keys have a long common prefix, then they are close.

Node State

To route search queries, each node must keep track of all the *contacts* or peers it knows about. For each $0 \leq i < 160$, a node keeps a list, L_i of $\langle$IP address, UDP port, Node ID$\rangle$ triplets for nodes between 2^i and 2^{i+1} in distance from itself [80]. For small values of i, the possible nodes in L_i is very small since the address space 2^i to 2^{i+1} is small. However, for large values of i, for instance, $i = 159$, the list can be very large. For this reason, lists are usually limited to about 20 nodes. These lists are also referred to as k-buckets where k is a system-wide parameter. k is chosen such that it is unlikely that all k nodes will disappear from the system within an hour. K is usually set to 20.

Figure 2.9 illustrates what the k-buckets for node P_0 would look like. The distance between node P_0 and the other nodes are all quite far.

To determine what nodes to keep in the k-buckets, Kademlia uses a least-recently seen eviction policy. Whenever a node receives a message from another node it updates the k-bucket for that node. If the node is already in the list, it is moved to the tail of the list. If the k-bucket is full, the node pings the head of the list (least-recently seen node). If the least-recently seen node does not reply in a timely manner, it is removed from the list and the new node is inserted at the end

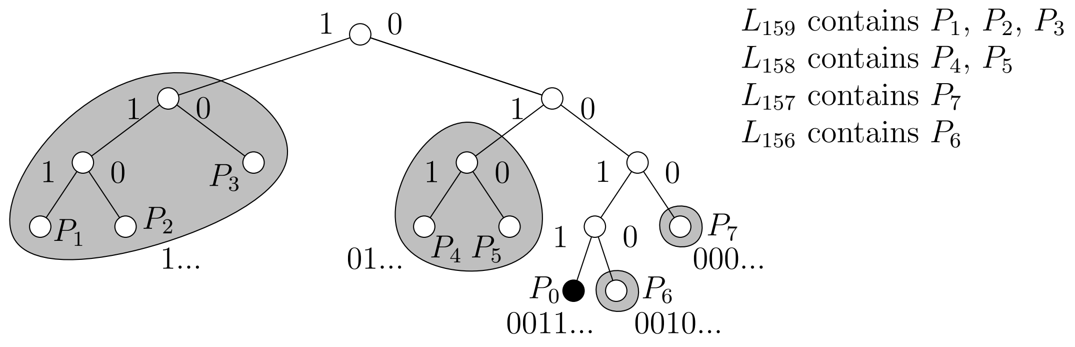

Figure 2.9: A node keeps track of all the nodes it knows about in k-buckets. Here, the k-buckets for node P_0 is shown.

of the list. On the other hand, if it does reply, it is moved to the tail of the list and the new node is discarded. This is illustrated in Figure 2.10.

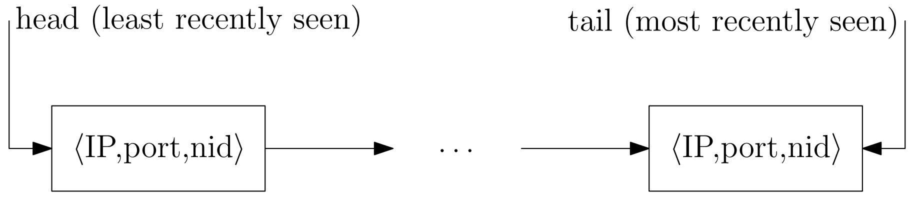

Figure 2.10: Each k-bucket is arranged so that the least recently seen node is at the head and the most recently seen node is at the tail.

Protocol Primitives

The Kademlia protocol requires four primitive Remote Procedure Call (RPC) operations: *PING*, *STORE*, *FIND_NODE*, *FIND_VALUE*.

PING: Probes a node to see if it is online.

STORE: Sends a ⟨data, value⟩ pair to a node for later retrieval.

FIND_NODE: Sends a node ID. The node should return at most k ⟨IPaddress, UDPport, Node ID⟩ triplets for the Node IDs closest to the request ID.

FIND_VALUE: Sends a key. If the corresponding value is stored, the node returns the value. Otherwise, *FIND_NODE* is performed.

Node Lookup

A node lookup is performed whenever a Kademlia node wishes to find the k closest nodes to a given node ID. Node lookups are performed in parallel where α denotes the amount of parallelism. It is known that 3 is a good value [89]. The lookup is performed in the following steps:

1. Pick α closest nodes from the set of nodes it knows about. Store closest node in *closestNode* and store the α closest nodes in *shortlist* [88].

2. For α nodes in *shortlist* that have not yet been queried, call FIND_NODE in parallel.

3. On response, update *shortlist* with any new nodes found. If a closer node is found, update *closestNode*. Nodes that do not respond in a timely fashion are discarded.

4. If *closestNode* was not updated, call FIND_NODE to the k closest nodes that have not been queried. If responses have been received from k closest nodes, stop the lookup. If *closestNode* was updated, go to step 2.

Operations

Operations in Kademlia use node lookups in conjunction with the primitive RPCs. To store a $\langle$key, value$\rangle$ pair, a node lookup is performed followed by the STORE RPC to all k closest nodes. To find a $\langle$key, value$\rangle$ pair, a node lookup is performed using FIND_VALUE instead of FIND_NODE.

Kademlia in BitTorrent

Keys in BitTorrent are generated by calculating the hash of the .torrent file meta-data. This is usually known as the *info-hash* [90]. The data stored is a list of all the peers participating in the swarm. $\langle$key, value$\rangle$ pairs are stored at the 8 nodes closest to info-hash [90]. 8 nodes is considered sufficient to minimize the probability that all 8 nodes will leave the system within the announce interval.

To join a swarm, a peer must perform a lookup based on the info-hash of the torrent. The 8 nodes closest to the info-hash should add the announcing peer to their peer list. As well, the list of known peers in the swarm is returned to the announcing peer.

Other Considerations

Queries in Kademlia contain the sender's contact information. This way, nodes can keep track of new and lost nodes in the system. However, if no queries have been performed in the system for some time, Kademlia requires nodes to make dummy queries to refresh the network. Similarly, when a node joins the network through a single other node of the network, it can perform a node lookup for itself which will refresh the system.

2.3.7 Other

Examples of other unstructured CDNs include Freenet. Some examples of structured CDNs include CAN, Chord, Pastry, Tapestry, etc.

2.4 Application to Online Video

A peer-to-peer system for online video would encounter some of the same requirements of other content distribution networks. These requirements are discussed below.

2.4.1 Scalable

If a user contributes as much upload bandwidth as download bandwidth, then as long as the original uploader uploads a single copy, any number of users can download it. It is infinitely scalable.

Using a fixed number of centralized servers to keep track of content or users such as in Napster, eDonkey and BitTorrent is unscalable. As the number of users increase, the amount of centralized resources must also increase. However, although BitTorrent trackers are used by peers, the trackers do not keep record what pieces each peer has but merely that a peer is participating in a swarm. This differs from Napster and eDonkey which keeps track of what each user is sharing. This reduces the resource requirement of the centralized server. In YouTube's case, the central servers are massive and it may be worthwhile to keep a lot more information centrally.

Gnutella has no centralized component, but its search mechanism is clearly unscalable as flooding the network is extremely inefficient. By using partially decentralized architecture such as in new versions of Gnutella and FastTrack, the search mechanism is made more efficient but is still unscalable since the core of the network still uses flooding for search queries. In YouTube's case, search can be efficiently conducted centrally.

2.4.2 Cost

Costs will dramatically fall if a peer-to-peer system is implemented for online video. This is the primary purpose of the rest of this study. This will allow YouTube to improve the quality of videos, reduce advertisements and improve the overall user experience.

2.4.3 Timeliness

Typically, downloading from P2P networks is slower than downloading from a traditional CDN. This is because downloading from P2P networks usually require a ramp-up time for gathering other nodes and rare pieces (as in BitTorrent).

In YouTube's case, some content is wildly popular whereas most content is not. Thus, peer-to-peer activity can be very efficient in distributing popular content. The unpopular content can be served by centralized means. As long as the peer-to-peer architecture can fall back on YouTube's servers to maintain downloading at the video bit rate, users will not feel any delay in watching videos.

2.4.4 Usable

The current P2P systems rely on client applications to communicate with other peers and deliver content. However, the majority of users access the Internet through a browser. Using an additional client application presents an extra hurdle for both content providers and their users.

Thus, one really important aspect of deploying a peer-to-peer system for online video is easy installation and seamless integration with existing browsers. This could be accomplished by a method similar to using Tor [91] in a browser such as Firefox[92] [93].[3] The P2P application could run a proxy server which communicates over the P2P CDN network. The browser could then route requests through the proxy server over to the P2P network. The proxy could be an add-on to the browser itself or it could be a background application.

2.4.5 Easily Deployable

Users may participate in a P2P network simply by downloading the client application. As this is a large cost-saving endeavour for YouTube, should YouTube want to deploy this feature, many engineers can be hired with the expected cost savings to create the system.

2.4.6 Robust

Networks that use centralized components are not as robust as fully decentralized systems. However, centralized servers provide more efficient means of searching and maintaining network state. Thus, it comes down to a tradeoff between the required robustness and efficiency.

In particular, a combination of an unstructured system and a structured system seems to work well as in the case of BitTorrent and eDonkey. In BitTorrent, the

[3]Tor is a privacy preserving network that..

unstructured system uses a central tracker and central websites to facilitate efficient search and storage of network state. The additional Kademlia network, provides a structured means of search and finding nodes. Should the central tracker fail, the members of a swarm can continue to operate over the Kademlia network.

2.4.7 Data integrity

There are two requirements of data integrity. First, data received from any peer should be the same as the original data of the content provider. Second, the metadata (file name, file size, author) of a file should be reliable.

Older P2P networks provided no guarantees on the quality of the data being downloaded. BitTorrent, on the other hand, uses SHA1 to prove that any piece received is an exact copy of the original piece. Using web sites where trusted users can comment on specific files, a search for metadata should return reliable content. eDonkey and BitTorrent use such approaches.

Unless network coding is used, data integrity should be very easy to accomplish with a centralized server which hands out piece hashes.

2.4.8 Security

BitTorrent offers encryption functionality to try to prevent eavesdroppers or malicious attackers from interfering with a transfer. For the majority of applications, privacy is not a significant issue since users are accustomed to leaving behind a trail of information when using the web. Privacy also requires additional resources and foolproof methods are difficult. For instance, to prevent traffic analysis, something like Tor would have to be used which would not only degrade the Tor network but provide an extremely slow download.

2.4.9 Fairness

Fairness is achieved inn existing peer-to-peer systems on a node-to-node basis. eMule has a credit system that keeps track of the amount uploaded by neighbouring peers. BitTorrent uses tit-for-tat. It is possible to cheat these existing systems. Fairness can be better achieved by a centralized system which YouTube can deploy.

2.4.10 Highly Available

Popular content is always highly available on P2P networks when a lot of users are simultaneously downloading the same thing. Conversely, downloading rare content is difficult. Searching for rare content is slow in purely decentralized and structured networks, peers usually do not stay in the system forever and downloading from a few peers takes a longer time. By having a large, existing centralized system, YouTube can ensure that any peer-to-peer activity can always fall back on the centralized system. This ensures that content is highly available.

Chapter 3

System Model

3.1 Components

A web-based video-on-demand (VOD) peer-to-peer (P2P) system has three components:

1. *servers* with infinite capacity
2. *viewers*; those peers who are watching as they are downloading videos
3. *idlers*; peers who are not watching videos but are still in the system.

3.1.1 Servers

Existing VOD systems on the web like YouTube have essentially enough capacity to handle any and all requests. There is also very little delay from when a user decides to watch a video until when the video starts streaming. Therefore, a request for service can be considered to be immediate.

3.1.2 Viewers

A peer who wishes to watch a video, downloads the video as the viewing is taking place. The bitrate of a video is the rate at which data is being consumed. This means a video with a constant bitrate of 300 Kbps will require a user to download 300 kilobits per second so that the video can be viewed without stuttering. If a user downloads at the bitrate, then the user will finish watching at the same time as when the user finishes downloading. If users downloads faster than the bitrate, the download will be complete before the video has finished playing.

3.1.3 Idlers

A peer who has finished watching and downloading a video but is still in the system is defined as an idler. These peers are able to upload videos that they have viewed in the past. For example, a peer may be choosing the next video to go to, or has paused the video at some point and will be on the web page longer than the duration of the video. Idlers are very similar to seeds in a BitTorrent-like P2P system. The difference is that idlers are assumed to not care about what video files they keep in their temporary caches since any video is available on-demand.

3.2 Collaboration

Collaboration can take place between two peers if one peer has content that another peer needs. Most existing peer-to-peer systems look at sharing of the same file between peers who need it. In BitTorrent, peers connect in a swarm to download the same torrent.

Single video collaboration (SVC) takes place when peers only upload the video that they are currently watching and downloading. This is shown in Figure 3.1.

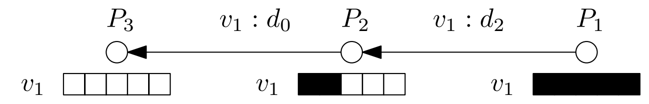

Figure 3.1: Single video collaboration. A peer P_2 is downloading data piece d_2 of video v_1 from peer P_1 and uploading d_0 of the same video to P_3.

On the other hand, *multiple video collaboration* (MVC) allows users to upload videos different from the ones they are currently downloading. For example, one peer may have one video and another peer has another video and they are able to share these videos between each other. No peer wants to upload without the proper incentives. Uploaders should receive something in return from the downloaders such that the net bandwidth (uploaded versus downloaded) for any particular peer should be 0 in the long run. See Figure 3.2.

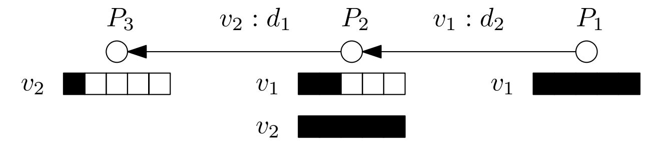

Figure 3.2: Multiple video collaboration. A peer P_2 is downloading data piece d_2 of video v_1 from peer P_1. At the same time, this peer is uploading d_1 of another video v_2 to peer P_3.

So far, SVC has been more popular because it allows users to participate in bartering and use tit-for-tat incentive schemes which has been shown to be very efficient. MVC requires some sort of credit or incentive system so that a user uploading a video, v_1, is given credits to download another video, v_2. This additional overhead has not been broadly implemented.

3.3 Service Models

The aim for any collaborating P2P system is to decrease the bandwidth, computation and storage on servers. In other words, its aim is to maximize peer traffic by having peers upload to other peers.

Single video collaboration allow peers that have pieces of a particular video to upload to other peers. Since a participating peer helps reduce server costs, these peers can be given priority over other peers for services. This may include higher quality videos and a reduction in advertisements. In large enterprises, collaboration may be mandatory when viewing videos to keep costs down for the company.

An earlier arriving peer can upload to peers arriving after. If peers leave the system as soon as they finish downloading, the earliest arriving viewer still in the system would have to download from the server. Figure 3.3 illustrates this situation.

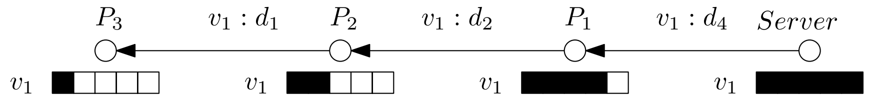

Figure 3.3: SVC with immediate peer departure. Server must upload to earliest arriving viewer.

Idlers are those peers that have finished downloading but are still in the system. These idlers can substitute as servers for the earliest arriving viewer. This would require peers to store videos that have already been watched in a cache. The number of idlers is dependent on the popularity of the video. If a video is popular, the number of viewers watching that video is high which means the number of idlers for that video is probably also high. See Figure 3.4.

If the number of idlers is high for a particular video, then many of these idlers will not be called upon to upload since only one is required to satisfy the download of the earliest arriving viewer. Thus, it would be beneficial if these idlers can upload a video that has no available idlers. See Figure 3.5.

One important observation to make is that the last arriving peer to a video does not have to upload to any other peer. Thus, this peer can be considered a viewer for the current watching video and an idler for any other videos that peer may have. In this situation, this peer may be downloading one video and uploading another video. However, it is assumed that this peer will remove its idler role as soon as another peer arrives for the currently watching video.

It is assumed that peers can only download one video at a time from either the server or from one peer. Also, it is assumed that peers can only upload one video at

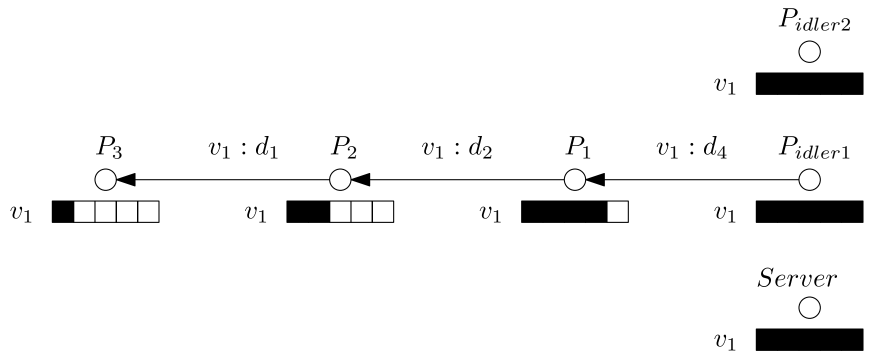

Figure 3.4: SVC with idlers offering previously watched video. Server is not required if an idler exists.

a time to one user. This restriction is to simplify the analysis and will be discussed as a secondary property of the system.

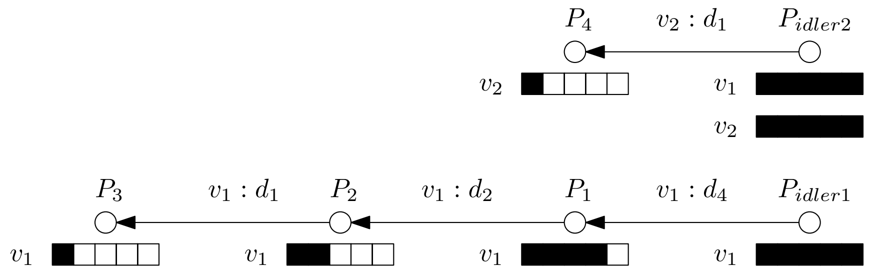

Figure 3.5: MVC with idlers offering previously watched video. Server is not required if an idler exists for any video not just the previously watched video.

3.4 Single Video Collaboration

In single video collaboration, each video can be studied separately from all other videos. To consider the server usage without any P2P assistance, the requests to a video can be modelled as a M/G/∞ queue. Requests for videos arrive according to a Poisson process with rate λ. The arrival process has been observed to be Poisson-like during short intervals through local network traces collected at a large university [26]. There are fluctuations depending on the time of day. However, for the peak periods of each day when P2P would be most helpful, the traffic is Poisson-like. The server uploads data according to a general distribution function G. The service times are independent and independent of the arrival process. The number of servers is infinite, so arrivals are serviced immediately.

Let $\rho = \lambda/\mu$ denote the mean number of busy servers. This is also the mean number of users viewing a particular video. The necessary queueing theory background is available in Appendix C on page 78.

3.4.1 Viewer Queue

All peers wanting to watch a particular video can be modelled as arrivals to a M/G/∞ queue. Because most users view and therefore download the video from beginning to the end, it is difficult for a particular user to upload to those that arrived before them. Therefore, it is assumed that users can only upload to those that arrive after. In particular, a strict rule is in place which forces users to upload to only the user that arrived immediately after. This is a modelling artifact and in reality, users may upload an useful piece to any needy peer.

It is assumed that peers download at the bitrate of the video. That is, peers download just fast enough to view the video without stuttering. This also means that peers must download the videos in sequential order from beginning to the end. Peers are also assumed to have enough upload bandwidth to upload at the bitrate and so any user arriving to a non-empty queue can download from the user that arrived immediately before it. The bitrate is assumed constant and 300 Kbps unless otherwise noted.

The earliest arriving peer in the queue must download from the server, and the rest of the peers can download from other peers. When the earliest arriving peer leaves the queue, the next earliest arriving peer having lost its stream downloads from the server. Therefore, with $n > 1$ users in the queue, the proportion of the traffic that is served by peers is $\frac{n-1}{n}$. Obviously, with $n = 0$, no peer traffic exists.

In reality, peers may share the video with a large number of peers at the same time. Some peers may not be able to upload at the bitrate and some peers may upload more than the bitrate. The same situation exists for downloading.

If no P2P is employed, the mean traffic sent per unit time by the server for a video with parameter ρ is,

$$\begin{aligned} T_{\text{server, no p2p}}(\rho) &= b\sum_{n\geq 0} np_n \\ &= b\rho \end{aligned} \tag{3.1}$$

where b is the bitrate of the video. In other words, the traffic sent per unit time on average is the mean number of users multiplied by the data rate required. $\rho = \lambda/\mu$ where λ represents the arrival rate and $1/\mu$ represents the amount of time the user is in service. Since in this model users download at the bitrate, $1/\mu$ is in fact the length of the video. This is illustrated in Figure 3.6.

3.4.2 No Idlers

If P2P is employed without any help from idlers, then the system can be modelled as a single M/G/∞ queue. Peers arrive to the queue at rate λ and leave at rate $n\mu$ where n is the current number of users in the queue.

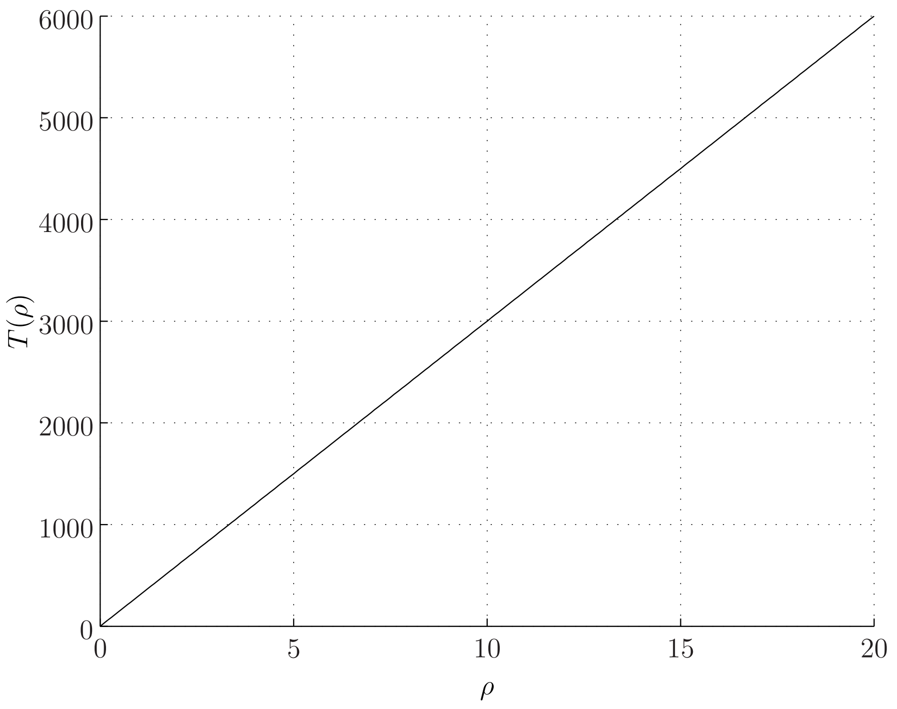

Figure 3.6: $T_{\text{server, no p2p}}$ and $b = 300$ Kbps.

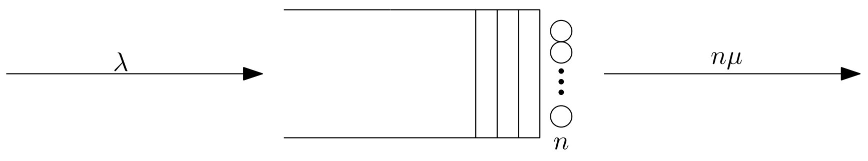

Figure 3.7: M/G/∞ queue modelling arrivals of viewers.

For any given number of users more than 0, the server only needs to upload one copy of the video to the earliest arriving peer. Therefore, in this situation, the server sends,

$$\begin{aligned} T_{\text{server, no idlers}}(\rho) &= b\sum_{n\geq 1} p_n \\ &= b(1-p_0) \\ &= b(1-e^{-\rho}) \end{aligned} \tag{3.2}$$

Thus, the traffic distributed by peers is,

$$\begin{aligned} T_{\text{peers, no idlers}}(\rho) &= T_{\text{server, no p2p}}(\rho) - T_{\text{server, no idlers}}(\rho) \\ &= b(\rho - (1-e^{-\rho})) \end{aligned} \tag{3.3}$$

The traffic for this model are shown below in Figure 3.8.

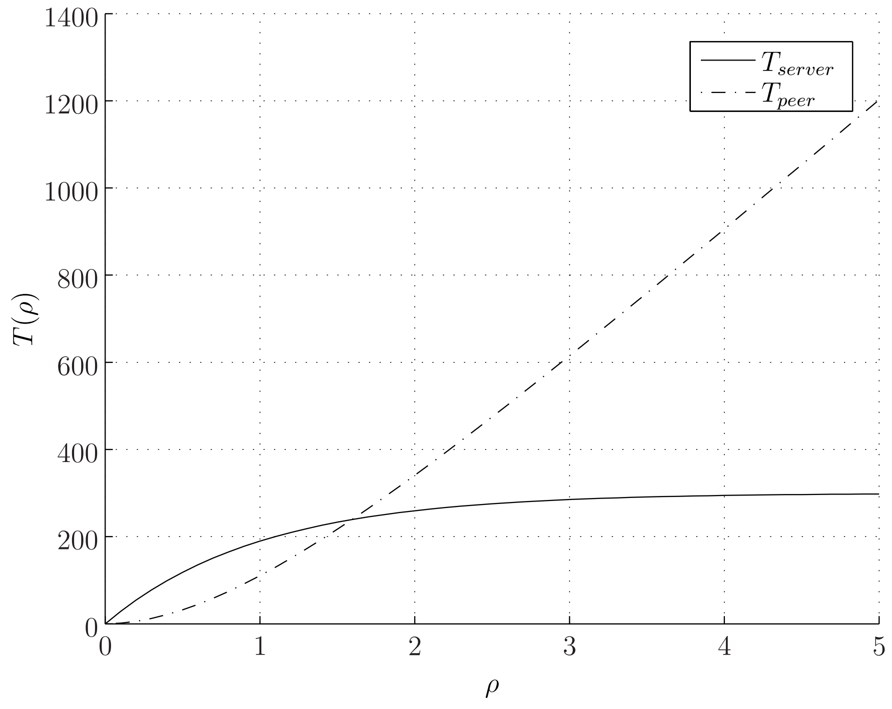

Figure 3.8: $T_{\text{server, no idlers}}$ and $T_{\text{peers, no idlers}}$ with $b = 300$ Kbps. As ρ approaches ∞, T_{server} approaches b.

The proportion of total traffic distributed by peers is,

$$\begin{aligned} P_{\text{peers, no idlers}}(\rho) &= \frac{T_{\text{peers, no idlers}}}{T_{\text{server, no p2p}}} \\ &= \frac{b(\rho - (1 - e^{-\rho}))}{b\rho} \\ &= 1 - \frac{1 - e^{-\rho}}{\rho} \end{aligned} \tag{3.4}$$

This is shown below in Figure 3.9. When ρ is small, very little server traffic can be saved with this model. However, when ρ is large, a big proportion of server traffic can be saved. This all depends on the popularity of a particular video.

3.4.3 P2P with LRU Cache of Size 1

If idlers stay in the system for a period of time after viewing a video, then these idlers can contribute their upload bandwidth. Here, it is assumed that each peer maintains a copy of the last watched video and can offer to upload this video to peers. This can be considered as a cache of size 1 with a Least Recently Used (LRU) caching policy that keeps the most recently watched video. In this situation, if a peer has a particular video, that peer can ensure that no server utilization is required for that video since that peer can act as the server.

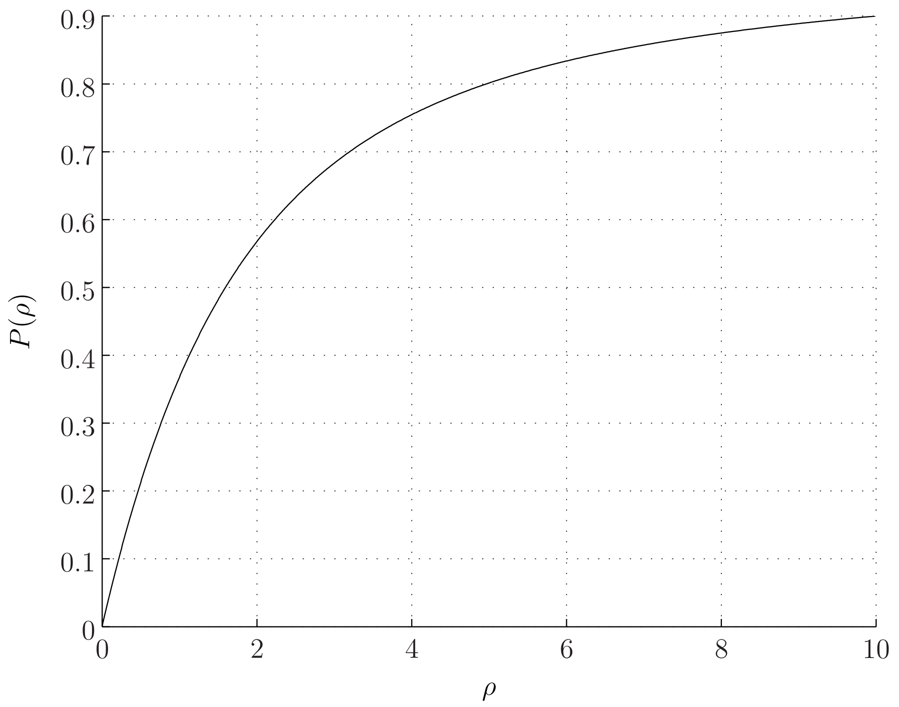

Figure 3.9: $P_{\text{peers, no idlers}}(\rho)$ with $b = 300$ Kbps. As ρ increases, P_{peers} approaches 1.

To model this, consider a system of two M/G/∞ queues in series. The first M/G/∞ queue, Q_{viewer}, is for all the users that are currently downloading and viewing a particular video. A second M/G/∞ queue, Q_{idler}, is used to model all those users who have finished downloading that video and are now idling in the system. It is assumed that every user in the first queue, upon finishing the download goes to the second queue. Users leave the system after idling in the second queue.

Let the arrival rate into the first queue be Poisson with rate λ. Because every user is assumed to eventually go to the second queue, the arrival rate of the second queue is also Poisson with rate λ. Let the mean service time of the first queue be $1/\mu_{viewer}$ and the mean time spent in the second queue be $1/\mu_{idler}$. Let the average number of users in each queue be denoted as $\rho_{viewer} = \lambda/\mu_{viewer}$ and $\rho_{idler} = \lambda/\mu_{idler}$ for Q_{viewer} and Q_{idler}, respectively. Also, let the population probabilities be $p_{viewer,i}$ and $p_{idler,i}$ for Q_{viewer} and Q_{idler}, respectively. Note that in the first queue, service time refers to the time spent watching videos. In the second queue, server time refers to the time spent not watching videos but still in the system right after watching a video. See Figure 3.10.

Obviously, the more popular a video is, the more viewers and idlers it has. Therefore, the number of idlers for a particular video should be proportional to the number of viewers. Let $0 < \eta$ be a parameter which represents the proportion of

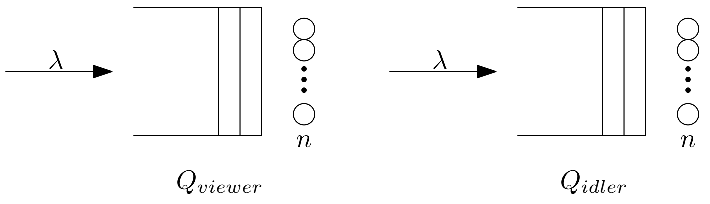

Figure 3.10: Q_{viewer} and Q_{idler} model. Although the arrival rates is the same for both queues in steady-state, the service times are different. If users view videos more than they idle, then Q_{viewer} will usually be longer.

viewers that are expected to idle. Then,

$$\frac{1}{\mu_{idler}} = \eta \frac{1}{\mu_{viewer}} \tag{3.5}$$

$$\rho_{idler} = \eta \rho_{viewer} \tag{3.6}$$

In steady state, assuming the two queues are independent, the probability that no idlers in Q_{idler} can substitute as the server is the probability that Q_{idler} is empty:

$$\begin{aligned} P\{Q_{idler} \text{ empty}\} &= p_{idler,0} \\ &= e^{-\rho_{idler}} \end{aligned} \tag{3.7}$$

Thus, instead of uploading 1 video when there are users, the server only uploads when Q_{idler} is empty and the expected server upload is $e^{-\rho_{idler}}$. Therefore, the traffic per unit time distributed by the server is,

$$\begin{aligned} T_{\text{server, with idlers and LRU cache}}(\rho) &= b \sum_{n \geq 1} p_n e^{-\rho_{idler}} \\ &= b(1 - e^{-\rho_{viewer}}) e^{-\rho_{idler}} \end{aligned} \tag{3.8}$$

This is illustrated in Figure 3.11. For small values of η, it is as if no idlers are in the system. For $\eta = 1$, the average number of users in Q_{idler} is the same as Q_{viewer}. If users are enticed to stay in the system for a long time, which is represented by a large η, then this is an extremely attractive P2P scheme since very little server traffic will be needed.

The traffic per unit time distributed by peers is,

$$T_{\text{peers, with idlers and LRU cache}}(\rho) = b(\rho_{viewer} - (1 - e^{-\rho_{viewer}}) e^{-\rho_{idler}}) \tag{3.9}$$

This is shown in Figure 3.12. As η increases, T_{peers} approaches $b\rho_{viewer}$. This means, as more users stay in the system, peer traffic dominates and server traffic is minimized.

The proportion of total traffic distributed by peers is,

$$\begin{aligned} P_{\text{peers, with idlers and LRU cache}}(\rho) &= \frac{b(\rho_{viewer} - (1 - e^{-\rho_{viewer}}))e^{-\rho_{idler}}}{b\rho_{viewer}} \\ &= 1 - \frac{(1 - e^{-\rho_{viewer}})e^{-\rho_{idler}}}{\rho_{viewer}} \end{aligned} \tag{3.10}$$

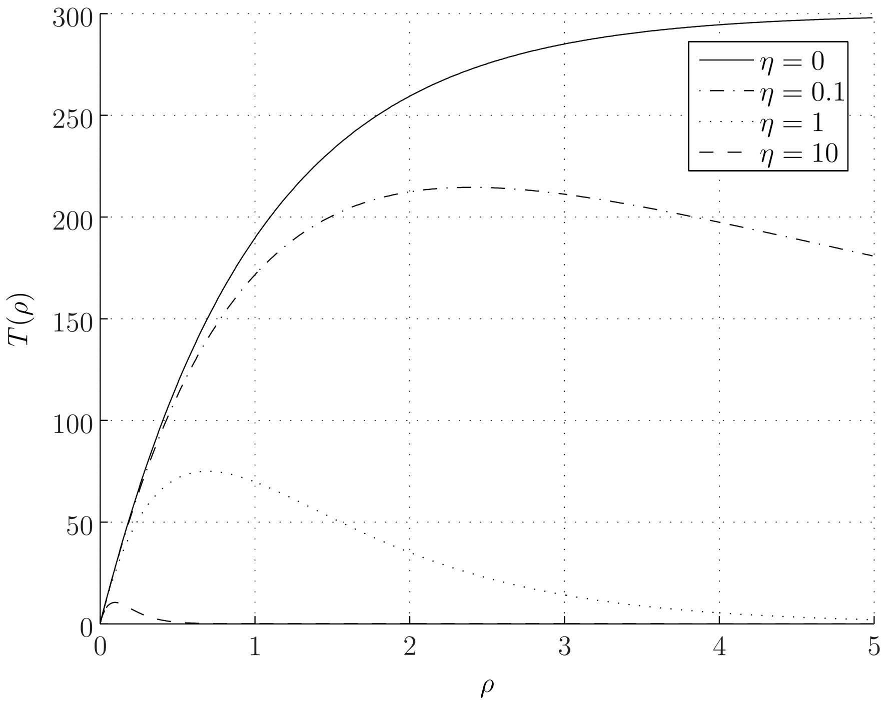

Figure 3.11: $T_{\text{server, with idlers and LRU cache}}(\rho)$ with $b = 300$ Kbps for different values of η. As η increases, T_{server} decreases for all values of ρ.

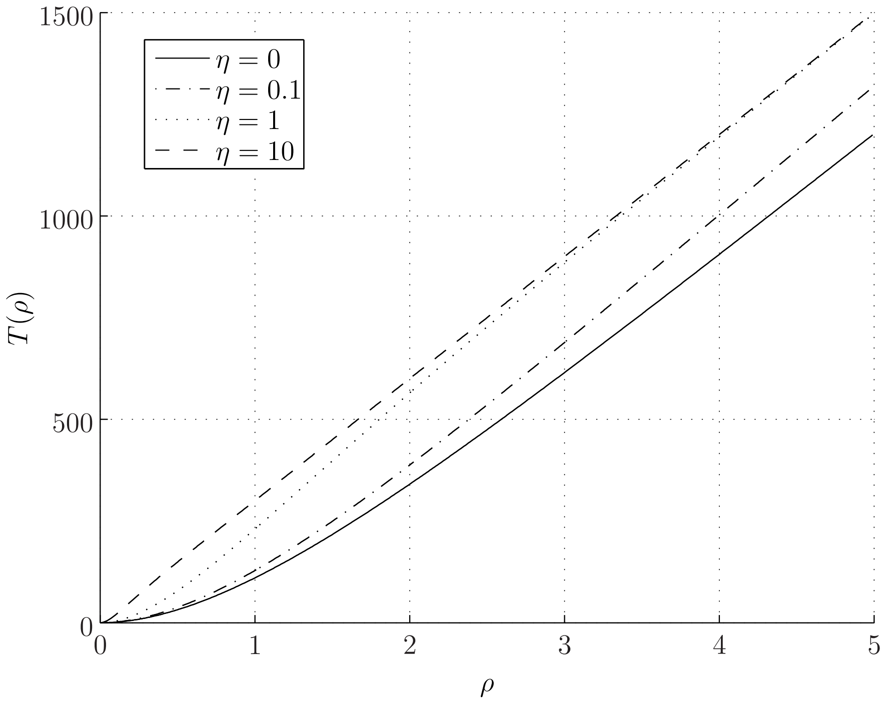

Figure 3.12: $T_{\text{peer, with idlers and LRU cache}}(\rho)$ with $b = 300$ Kbps for different values of η. As η increases, T_{peer} approaches a straight line.

This is illustrated in Figure 3.13. Of course, as peers stay in the system longer, the proportion of peer traffic approaches 1.

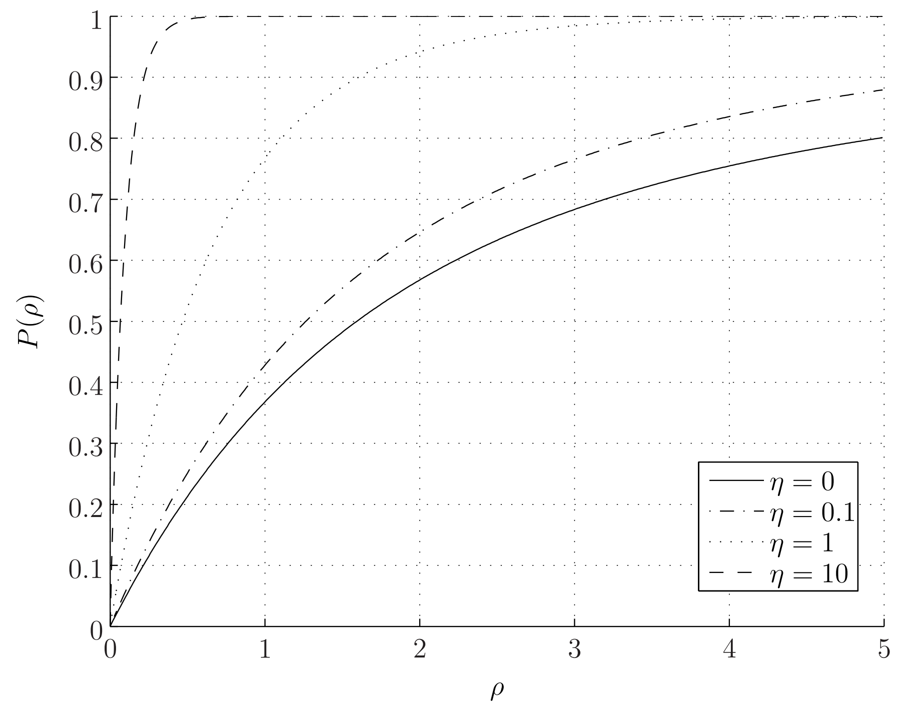

Figure 3.13: $P_{\text{peers, with idlers and LRU cache}}(\rho)$ with $b = 300$ Kbps for different values of η.

Chapter 4

Results

4.1 SVC Numerical Examples

For a particular video with users, the server must upload to one viewer unless the video is available from a seed in the idler queue. If the idler queue is considered global. That is, there is a single queue for all of the idling peers. Then the global optimization problem is to ensure that the idling peers as a whole can offload as much of the server requirements as possible.

So far, only the simple LRU cache of size 1 has been looked at. In what follows, this will be shown to be a sub-optimal strategy. For the purposes of demonstration, suppose there are two videos, v_1 and v_2 that are of the same length, $1/\mu = 1$. However, let v_2 be ten times as popular as v_1. Therefore, v_1 is has a viewer queue with parameter $\rho_1 = \rho$ and v_2 has a viewer queue with parameter $\rho_2 = 10\rho$. Assume $\rho = 1$ and $b = 1$.

4.1.1 No P2P

Thus, by Equation 3.1, the total server utilization without P2P is:

$$\begin{aligned} T_{v_1,\text{server, no p2p}} &= b\rho \\ &= 1 \end{aligned} \tag{4.1}$$

$$\begin{aligned} T_{v_2,\text{server, no p2p}} &= b10\rho \\ &= 10 \end{aligned} \tag{4.2}$$

4.1.2 P2P but No Idlers

If peers are allowed to help but no idlers are used, then by Equation 3.3,

$$\begin{aligned} T_{v1,\text{peers, no idlers}} &= b(\rho - (1 - e^{-\rho})) \\ &= 1(1 - (1 - e^{-1})) \\ &= 0.368 \end{aligned} \tag{4.3}$$

$$\begin{aligned} T_{v2,\text{peers, no idlers}} &= b(10\rho - (1 - e^{-10\rho})) \\ &= 1(10 - (1 - e^{-10})) \\ &= 9.0 \end{aligned} \tag{4.4}$$

The total peer utilization of this two video system with no idlers is 9.368 out of a possible 11. Alternatively, the server load has been decreased from 11 to 1.632.

4.1.3 P2P with LRU Cache of Size 1

If users do not leave the system right away and instead idles for one-tenth of the viewing time, then they can offer the previously watched video for upload. Therefore, if the viewing parameters for the two videos are $\rho_{1,viewer} = 1$ and $\rho_{2,viewer} = 10$, then the idling parameters for the two videos are $\rho_{1,idler} = 0.1$ and $\rho_{2,idler} = 1$. By Equation 3.9,

$$\begin{aligned} T_{v1,\text{peers, with idlers and LRU cache}} &= b(\rho_{1,viewer} - (1 - e^{-\rho_{1,viewer}})e^{-0.1\rho_{1,idler}}) \\ &= 1(1 - (1 - e^{-1})e^{-0.1}) \\ &= 0.428 \end{aligned} \tag{4.5}$$

$$\begin{aligned} T_{v2,\text{peers, with idlers and LRU cache}} &= b(\rho_{2,viewer} - (1 - e^{-\rho_{2,viewer}})e^{-0.1\rho_{2,idler}}) \\ &= 1(10 - (1 - e^{-10})e^{-1}) \\ &= 9.632 \end{aligned} \tag{4.6}$$

Now, the server utilization has decreased from 1.632 in the no-idler case, to 0.94 if viewers idle 10% of the time.

4.1.4 P2P with Arbitrary Cache of Size 1

Now, assume idling users can have any video in its cache. In other words, assume users have watched both videos at some point in the past and can keep either video in its cache. Therefore, the expected size of the global idling queue is Poisson distributed with parameter $0.1\rho + \rho = 1.1\rho$. Suppose, idlers with v_1 in its cache are distributed with parameter $x\rho$ whereas seeds with v_2 are distributed with parameter $(1.1 - x)\rho$. Then, the peer utilizations are as follows:

$$\begin{aligned} T_{v1,\text{peers, with general cache}} &= b(\rho - (1 - e^{-\rho})e^{-x\rho}) \\ &= 1(1 - (1 - e^{-1})e^{-x}) \\ &= 1 - 0.632e^{-x} \end{aligned} \tag{4.7}$$

$$\begin{aligned} T_{v2,\text{peers, with general cache}} &= b(10\rho - (1 - e^{-10\rho})e^{-(1.1-x)\rho}) \\ &= 1(10 - (1 - e^{-10})e^{-(1.1-x)}) \\ &= 10 - e^{-(1.1-x)} \end{aligned} \tag{4.8}$$

For maximum utilization, it is necessary to maximize the sum of the previous two values: $f(x) = 1 - 0.632e^{-x} + 10 - e^{-1.1-x}$. Rudimentary optimization shows that $x = 0.32$ is a maxima from which the two equations can be solved:

$$T_{v1,\text{peers, with optimal cache}} = 0.541 \tag{4.9}$$

$$T_{v2,\text{peers, with optimal cache}} = 9.542 \tag{4.10}$$

The total server utilization is 0.92 out of a possible 11. In this situation, it can be seen that the unpopular video should be cached slightly more than possible from a LRU scheme to be optimal. The make up of T_{peers} can be seen in Figure 4.1. Please keep in mind that the y-axis has been truncated to focus on the important areas.

4.2 Cache Parameters

In reality, a website holds millions if not billions of videos. It is assumed that users can keep any video that they have watched in their local cache. The combination of the local cache of all the users make up the global cache of videos at any time. Idlers are able to seed these videos to any peer that requests the files. The goal of the global cache is to minimize the total server bandwidth used for all the videos. Therefore, this is a cache distribution problem which answers the question of what videos peers should keep.

Based on the previous analysis, the cache distribution is dependent on a few factors:

1. distribution of video popularity
2. number of idling peers
3. replacement strategy
4. cache size

The analysis is restricted to caches of size 1 and is left as an open problem to optimize for larger cache sizes. Therefore, at any point in time, an idler can only hold onto and seed a single video.

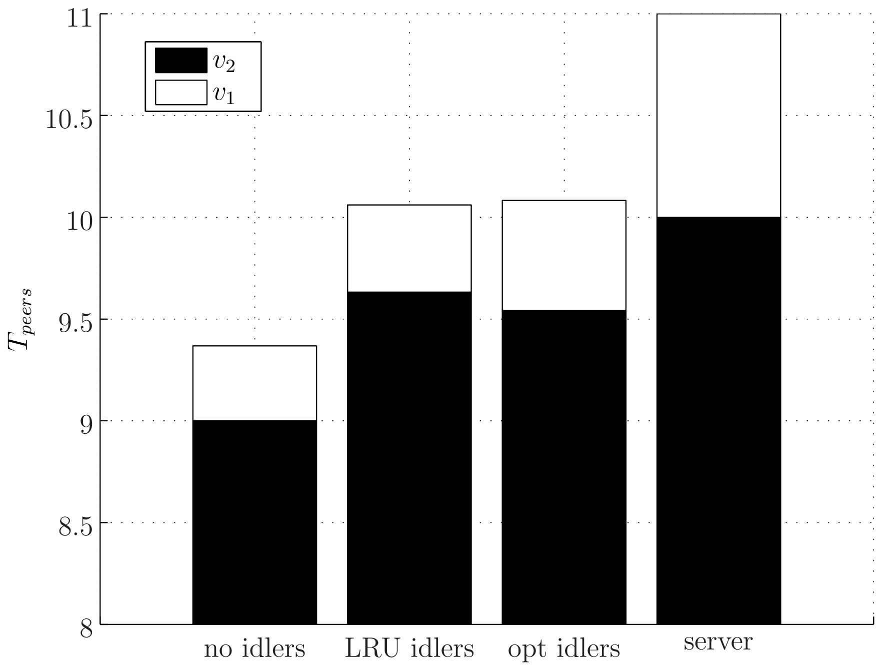

Figure 4.1: Comparing the peer traffic for v_1 and v_2 for different P2P models.

4.2.1 Video Popularity

Video popularity refers to the mean number of viewers for a particular video. On YouTube, video popularity follows a log-normal distribution. There are a few popular videos with many viewers and many unpopular videos with few viewers. A log-normal distribution for $\rho > 0$ with parameters μ and σ has the following probability density function (PDF),

$$f(\rho; \mu, \sigma) = \frac{1}{\rho\sigma\sqrt{2\pi}} e^{\frac{-(\ln\rho-\mu)^2}{2\sigma^2}} \tag{4.11}$$

where μ and σ are the mean and standard deviation of the variable's natural logarithm [94].

The cumulative distribution function (CDF) is as follows,

$$F(\rho; \mu, \sigma) = \frac{1}{2} + \frac{1}{2}\mathrm{erf}\left[\frac{\ln(\rho) - \mu}{\sigma\sqrt{2}}\right] \tag{4.12}$$

If X is a random variable denoting the number of users watching a particular video, the expected value and variance are,

$$\mathrm{E}(X) = e^{\mu+\sigma^2/2} \tag{4.13}$$

$$\mathrm{Var}(X) = (e^{\sigma^2} - 1)e^{2\mu+\sigma^2} \tag{4.14}$$

Given the mean and variance, the parameters μ and σ can also be calculated,

$$\mu = \ln(\mathrm{E}(X)) - \frac{1}{2}\ln\left(1 + \frac{\mathrm{Var}(X)}{(\mathrm{E}(X))^2}\right) \tag{4.15}$$

$$\sigma = \sqrt{\ln\left(\frac{\mathrm{Var}(X)}{(\mathrm{E}(X))^2} + 1\right)} \tag{4.16}$$

Depending on the distribution parameters, the log-normal distribution can take many different shapes. Using data provided online[1] from prior work [25], the two parameters for a large (700 thousand) set of videos was calculated to be $\mu = -3.726$ and $\sigma = 2.237$. Although inaccurate, henceforth, this data set will be assumed to be a crude representation of YouTube as a whole. The trace data and capture methodology are described in Appendix B.

Therefore, the mean and variance are,

$$\mathrm{E}(X) = e^{\mu+\sigma^2/2} = 0.2945 \tag{4.17}$$

$$\mathrm{Var}(X) = (e^{\sigma^2} - 1)e^{2\mu+\sigma^2} = 12.86 \tag{4.18}$$

To see the effect on the distribution when the mean and variance are different, consider means and variances ±50% from these values. See Table 4.1.

Table 4.1: Mean and variance of trace data.

Label	Mean	Variance	μ	σ
a	E(X)	Var(X)	-3.726	2.237
b	0.5E(X)	0.5Var(X)	-4.763	2.387
c	0.5E(X)	1.5Var(X)	-5.312	2.606
d	1.5E(X)	0.5Var(X)	-2.580	1.878
e	1.5E(X)	1.5Var(X)	-3.119	2.146

The trace data and the best-fit lines are shown in Figure 4.2. Figure 4.3 represents the PDF of the trace data and Figure 4.4 represents the CDF of the trace data.

To see which videos account for the most traffic, $f(\rho)\rho$ is shown in Figure 4.5. The majority of traffic is concentrated at $\rho < 5$. The video group with the most traffic are those videos with $\rho = 0.0241$ which represents the peak in Figure 4.5.

Total YouTube traffic

Assume the bitrate (b) of all the videos are the same. Also, assume there are $i = 1, 2, \ldots, N$ different videos with parameters ρ_i. Then the mean server traffic is,

$$T_{\text{YouTube server, no p2p}} = b\sum_{i=1}^{N} \rho_i \tag{4.19}$$

[1] `http://netsg.cs.sfu.ca/youtubedata.html`

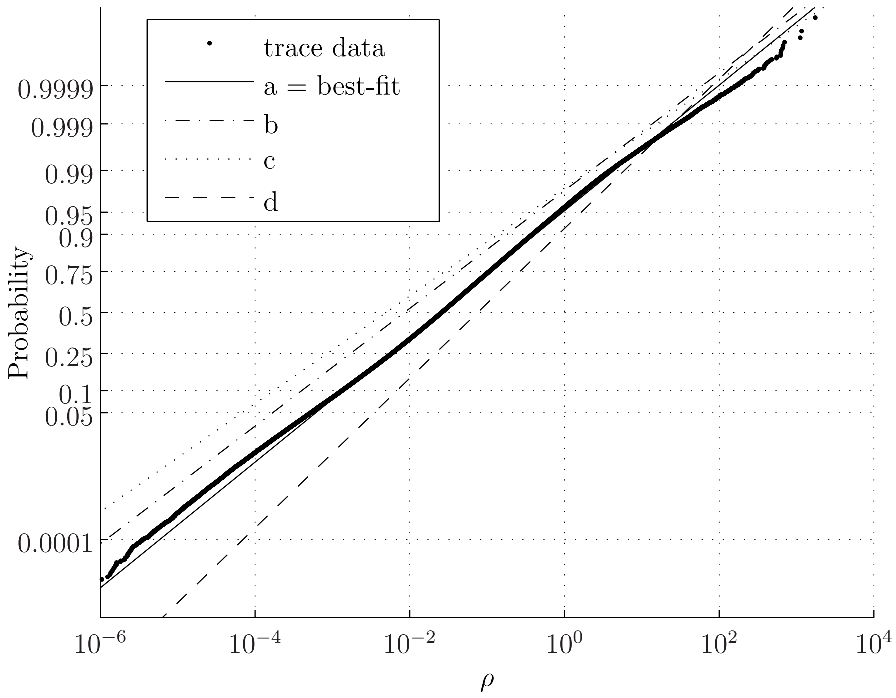

Figure 4.2: Probability plot showing trace data and best-fit.

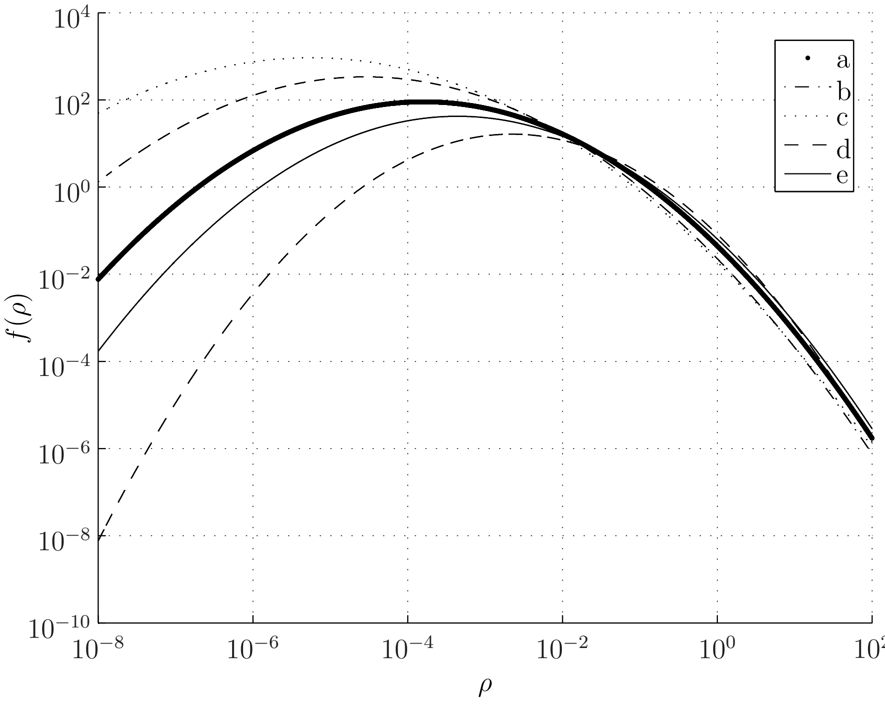

Figure 4.3: Log-normal probability density function for different means and variances.

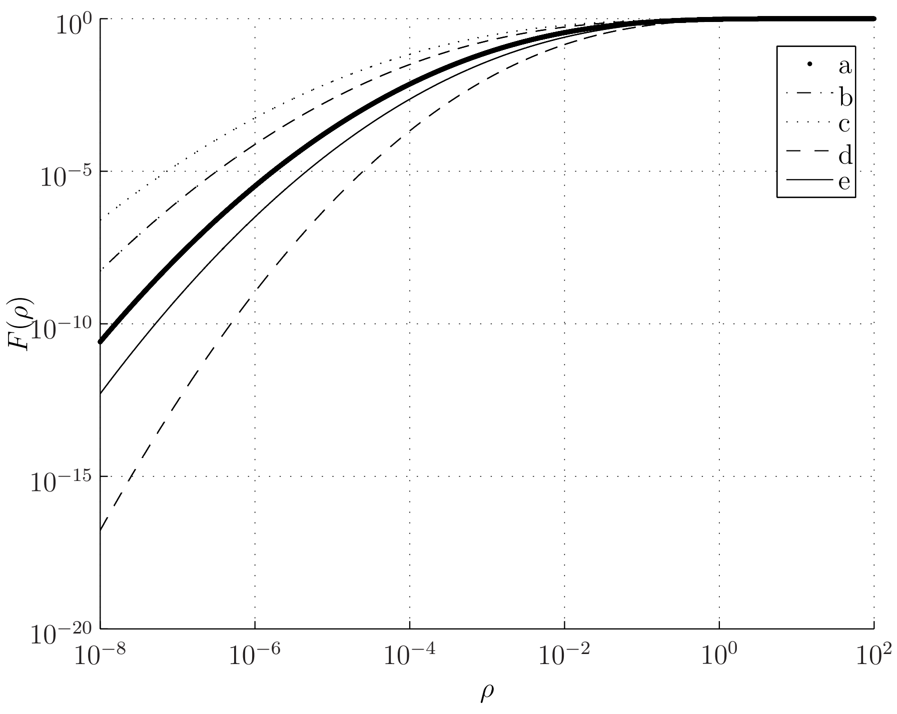

Figure 4.4: Log-normal cumulative distribution function for different means and variances.

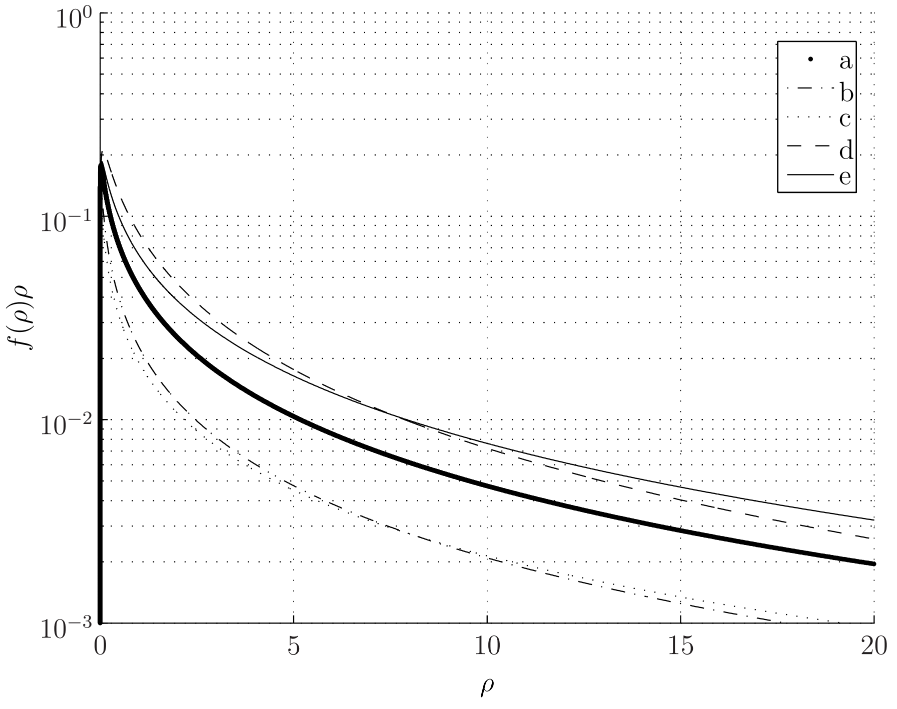

Figure 4.5: $f(\rho)\rho$ for different means and variances.

If $\sum_{\rho=1}^{N} \rho_i$ can be approximated by $E[\rho]N$, where $E[\rho]$ is the mean value of ρ_i, then,

$$\begin{aligned} T_{\text{YouTube server, no p2p}} &= bNE[\rho] \\ &= bN \int_{\rho>0}^{\infty} f(\rho)\rho \qquad (4.20) \\ &= bNe^{\mu+\sigma^2/2} \qquad (4.21) \end{aligned}$$

where $E[\rho] = e^{\mu+\sigma^2/2}$ for a log-normal distribution.

For the trace obtained, $N = 743308$ and $E[\rho] = 0.294483$. Therefore, using the log-normal model, the mean number of peers online at the same time is: 218891. Without making any assumptions on the distribution of videos and simply summing the ρ of all the videos yields: 231035. If arrivals to each video is Poisson, then globally the arrivals to YouTube is Poisson with mean 218891 if video popularity is log-normal distributed.

4.2.2 Number of Idlers

The prior analysis assumed the number of idling peers is a function of the total number of peers in the system. The more peers there are watching videos, the more peers there are idling. With well constructed incentives, it may be possible to hold peers in the system longer. Obviously, the more seeds there are, the better the cache performance.

A secondary effect not modelled is due to the last arriving peer to a viewer queue. This peer does not have to upload to anyone until another peer arrives to that viewer queue. This last arriving peer can then use its upload capacity as if it were an idler. This technique would work better for less popular videos which is less likely to receive more than 1 peer at a time.

Let there be $i = 1, 2, \ldots, N$ different videos with parameters ρ_i. Assuming the view counts for videos are independent, then the total expected number of users in the system is $\sum_{i=1}^{N} \rho_i$. Let $0 < \eta$ be a parameter which represents the proportion of viewers that are idling in steady state. Then, the total number of idling users is,

$$E[Q_{cache}] = \eta \sum_{i=1}^{N} \rho_i \qquad (4.22)$$

4.2.3 Replacement Strategy

A *cache hit* for video v occurs when at least one peer in the set of idling peers, has v and is not currently uploading some other video. Let the number of peers in the cache be defined by a Poisson distribution with parameter $E[Q_{cache}]$. Let the occurrence of videos $v_i, i = 1, 2, \ldots, N$ in the cache be defined by a probability distribution h. Thus, for a slot, S_j, in the cache, there exists a probability q_i, such that $h(S_j = v_i) = q_i$. In other words, let the probability that v_i is in a particular slot be q_i. Let the set of all slots be defined by $S = \{S_1, S_2, \ldots, S_j, \ldots\}$, where $|S|$ represents the number of slots available.

For a cache size limited to a single video, each peer can upload to at most one other peer at a time. There is no contention for upload bandwidth. Thus, by Bayes' Theorem, the probability that a particular video v_i does not exist in the system is,

$$\begin{aligned}
P\{v_i \text{ not in cache}\} &= \sum_{k=0}^{\infty} P\{v_i \text{ not in } S| \, |S| = k\} P\{|S| = k\} \\
&= \sum_{k=0}^{\infty} \left[\prod_{j=1}^{k} P\{v_i \text{ not in } S_j\}\right] \left[\frac{E[Q_{cache}]^k}{k!} e^{-E[Q_{cache}]}\right] \\
&= \sum_{k=0}^{\infty} \left[\prod_{j=1}^{k} (1 - q_i)\right] \left[\frac{E[Q_{cache}]^k}{k!} e^{-E[Q_{cache}]}\right] \\
&= \sum_{k=0}^{\infty} (1 - q_i)^k \left[\frac{E[Q_{cache}]^k}{k!} e^{-E[Q_{cache}]}\right] \quad \text{, by independence} \\
&= e^{-E[Q_{cache}]} \sum_{k=0}^{\infty} \frac{((1 - q_i)E[Q_{cache}])^k}{k!} \\
&= \frac{e^{-E[Q_{cache}]}}{e^{-(1-q_i)E[Q_{cache}]}} \sum_{k=0}^{\infty} \frac{((1 - q_i)E[Q_{cache}])^k}{k!} e^{-(1-q_i)E[Q_{cache}]} \\
&= \frac{e^{-E[Q_{cache}]}}{e^{-(1-q_i)E[Q_{cache}]}} \quad \text{, sum of a Poisson pdf} \\
&= e^{-q_i E[Q_{cache}]} \qquad (4.23)
\end{aligned}$$

For example, using the two videos from Chapter 4.1 on page 49 with $E[Q_{cache}] = 1.1$ and v_2 ten times as popular as v_1 it is possible to calculate the cache miss probability. If the peers utilize a single cache using LRU, then the cache distribution probabilities are: $q_1 = \frac{1}{11}$ and $q_2 = \frac{10}{11}$. Therefore,

$$P\{v_1 \text{ not in cache}\} = e^{-\frac{1}{11}1.1} = e^{-0.1} \qquad (4.24)$$

$$P\{v_2 \text{ not in cache}\} = e^{-\frac{10}{11}1.1} = e^{-1} \qquad (4.25)$$

which are the same probabilities as before.

The probability of a cache hit is

$$P\{v_i \text{ is in cache}\} = 1 - e^{-q_i E[Q_{cache}]} \qquad (4.26)$$

4.2.4 Cache Size

So far the analysis has focused on single video sized caches. In reality, a cache of size C can be explored. This makes the analysis a lot more difficult, because then there is contention for videos. Given that idlers each have C videos, which video will the idler upload? Perhaps an idler can upload two videos at the same time. Therefore, for modelling purposes, each peer can only upload one video at a time. Modelling bigger caches may require careful simulation.

4.2.5 Global Cache Model

In this system model, each video has a viewing queue made up of all those peers that are currently viewing a video. Let v_i's viewer queue be denoted by $Q_{v_i,viewer}$. In addition, there is one large queue, Q_{cache}, where all idling peers go to. This model is illustrated below in Figure 4.6.

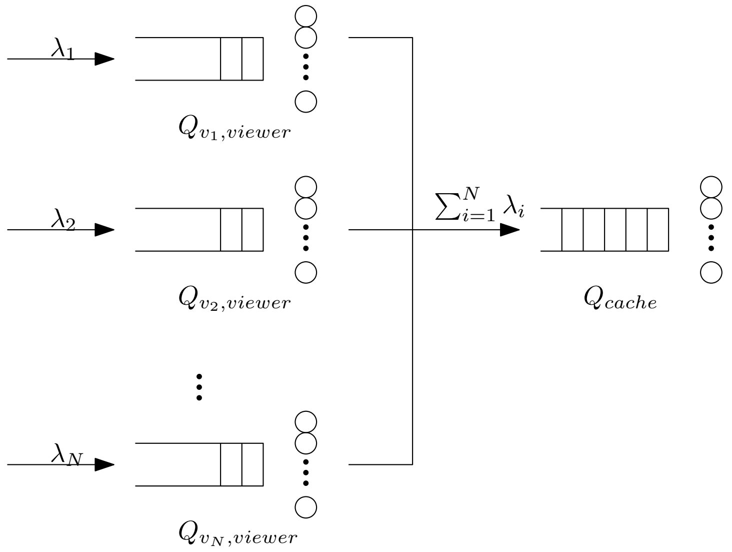

Figure 4.6: $Q_{v_i,viewer}$ and Q_{cache} model. All viewers go from viewer queues to the cache queue. Any user from the cache queue can become server for any video.

The probability that Q_{cache} can substitute as the server for video v_i is the probability that Q_{cache} contains v_i. Reproduced from the previous analysis,

$$P\{v_i \text{ not in cache}\} = e^{-q_i E[Q_{cache}]} \tag{4.27}$$

where

$$E[Q_{cache}] = \eta \sum_{i=1}^{N} \rho_i \tag{4.28}$$

Therefore, for a particular video v_i, the mean traffic distributed per unit time by the server is,

$$\begin{aligned} T_{v_i \text{ server, with p2p and general cache}} &= bP\{|Q_{v_i,viewer}| > 0\}P\{v_i \notin Q_{cache}\} \\ &= b(1 - e^{-\rho_i})e^{-q_i E[Q_{cache}]} \end{aligned} \tag{4.29}$$

where b represents the bitrate, ρ_i represents the mean number of viewers, and q_i represents the probability of a cache slot having v_i.

For all videos in the system, the traffic distributed per unit time is the sum of the traffic distributed by each individual video. Thus,

$$T_{\text{all servers, with p2p and general cache}} = b\sum_{i=1}^{N}(1-e^{-\rho_i})e^{-q_i E[Q_{cache}]} \tag{4.30}$$

4.2.6 Limits

All videos are popular

If all of the videos are popular, then ρ_i is large and $(1-e^{-\rho_i}) \rightarrow 1$. Whether a video has $\rho = 10$ or $\rho = 100$ does not really matter. In this situation, to make the best use of the cache, it is obvious that each video should receive equal representation in the cache so that $q_i = \frac{1}{N}$. This can be implemented as a rarest-first cache replacement strategy. Thus, Equation 4.30 can be simplified as follows,

$$\begin{aligned} T_{server,popular} &= b\sum_{i=1}^{N}(1-e^{-\rho_i})e^{-q_i E[Q_{cache}]} \\ &= bNe^{-\frac{1}{N}E[Q_{cache}]} \end{aligned} \tag{4.31}$$

If $E[Q_{cache}] \gg N$, then $e^{-\frac{1}{N}E[Q_{cache}]} \rightarrow 0$ so that $T_{server,popular} \rightarrow 0$. In other words, if all videos are popular and users idle for a long time, then the server requirements approach 0.

If $E[Q_{cache}] \ll N$, then $e^{-\frac{1}{N}E[Q_{cache}]} \rightarrow 1$ and $T_{server,popular} \rightarrow bN$. Therefore, if videos are popular but users do not idle, then the server must upload at the bitrate for each video all the time. This is because popular videos will usually have at least one viewer at all times.

All videos are unpopular

If all of the videos are unpopular, then ρ_i is small and $(1-e^{-\rho_i}) \rightarrow \rho_i$. Thus, Equation 4.30 can be simplified as follows,

$$\begin{aligned} T_{server,unpopular} &= b\sum_{i=1}^{N}(1-e^{-\rho_i})e^{-q_i E[Q_{cache}]} \\ &= b\sum_{i=1}^{N}\rho_i e^{-q_i E[Q_{cache}]} \end{aligned} \tag{4.32}$$

If $q_i E[Q_{cache}] \gg 1$, then $e^{-q_i E[Q_{cache}]} \rightarrow 0$ so that $T_{server,unpopular} \rightarrow 0$. Therefore, as long as the idling queue is large, it does not matter what caching strategy is used or what video popularity is involved.

However, if there are not many idlers, then $q_i E[Q_{cache}] \ll 1$ and $e^{-q_i E[Q_{cache}]} \rightarrow 1$. Now, $T_{server,unpopular} \rightarrow b \sum_{i=1}^{N} \rho_i$.

4.3 Global Cache Optimization

In a real setting, there are both popular and unpopular videos and choosing q_i will be dependent on both $E[Q_{cache}]$ and ρ_i. The goal is to minimize the following equation:

$$\begin{aligned}
T_{\text{all servers, with p2p and general cache}} &= b \sum_{i=1}^{N} (1 - e^{-\rho_i}) e^{-q_i E[Q_{cache}]} \\
\frac{T_{\text{all servers, with p2p and general cache}}}{b} &= \sum_{i=1}^{N} (1 - e^{-\rho_i}) e^{-q_i E[Q_{cache}]} \\
f(q_1, q_2, \ldots, q_N) &= \sum_{i=1}^{N} k_i e^{-q_i E[Q_{cache}]} \qquad (4.33)
\end{aligned}$$

where $k_i = (1 - e^{-\rho_i})$ and $E[Q_{cache}]$. For a particular $\{k_i, E[Q_{cache}]\}$ there exists an optimum set of q_is.

In general, for videos v_i ranked by ρ_i such that $\rho_1 \geq \rho_2 \geq \ldots \geq \rho_N$ where ρ_1 is the most popular, q_i has the following properties,

1. $\sum_{i=1}^{N} q_i = 1$
2. $q_1 \geq q_2 \geq \ldots \geq q_N$, since more popular videos will require more server utilization and should be represented as such in the cache.
3. After some cutoff, c, $q_{c+1} = q_{c+2} = \ldots = q_N = 0$, since it may not be optimal to cache some unpopular videos at all. This is dependent on the size of the cache, $E[Q_{cache}]$.

4.3.1 Solving for q_i

For some cutoff c, let,

$$q_1 = 1 - \sum_{i=2}^{c} q_i \qquad (4.34)$$

Then, Equation 4.33 becomes,

$$\begin{aligned}
f(q_1, q_2, \ldots, q_N) &= \sum_{i=1}^{N} k_i e^{-q_i E[Q_{cache}]} \\
f(q_2, \ldots, q_N) &= k_1 e^{-(1-\sum_{i=2}^{c} q_i)E[Q_{cache}]} + \sum_{i=2}^{N} k_i e^{-q_i E[Q_{cache}]} \\
&= k_1 e^{-(1-\sum_{i=2}^{c} q_i)E[Q_{cache}]} + \sum_{i=2}^{c} k_i e^{-q_i E[Q_{cache}]} + \sum_{i=c+1}^{N} k_i e^{-q_i E[Q_{cache}]} \\
&= k_1 e^{-(1-\sum_{i=2}^{c} q_i)E[Q_{cache}]} + \sum_{i=2}^{c} k_i e^{-q_i E[Q_{cache}]} + \sum_{i=c+1}^{N} k_i \qquad (4.35)
\end{aligned}$$

where $q_{c+1} = q_{c+2} = \cdots = q_N = 0$.

Theorem 1 *Let $x*$ be an interior point of a domain D in R^n and assume that f is twice continuously differentiable on D. It is necessary for a local extrema of f at $x*$ that [95]*

$$\nabla f(x*) = 0 \qquad (4.36)$$

which implies that $\forall x_i$,

$$\frac{\partial f(x*)}{\partial x_i} = 0 \qquad (4.37)$$

Using Theorem 1 and Equation 4.35, $\forall q_{i,\ 2 \le i \le c}$,

$$\begin{aligned}
\frac{\partial f}{\partial q_i} &= E[Q_{cache}] k_1 e^{-(1-\sum_{i=2}^{c} q_i)E[Q_{cache}]} - E[Q_{cache}] k_i e^{-q_i E[Q_{cache}]} \\
0 &= k_1 e^{-(1-\sum_{i=2}^{c} q_i)E[Q_{cache}]} - k_i e^{-q_i E[Q_{cache}]} \\
k_i e^{-q_i E[Q_{cache}]} &= k_1 e^{-(1-\sum_{i=2}^{c} q_i)E[Q_{cache}]} \\
k_i e^{-q_i E[Q_{cache}]} &= k_1 e^{-q_1 E[Q_{cache}]} \quad \text{, since } q_1 = 1 - \sum_{i=2}^{c} q_i \\
\frac{e^{-q_i E[Q_{cache}]}}{e^{-q_1 E[Q_{cache}]}} &= \frac{k_1}{k_i} \\
-q_i E[Q_{cache}] + q_1 E[Q_{cache}] &= \ln\left(\frac{k_1}{k_i}\right) \\
q_1 - q_i &= \frac{1}{E[Q_{cache}]} \ln\left(\frac{k_1}{k_i}\right) \qquad (4.38)
\end{aligned}$$

Thus, using Equation 4.38 and $\sum_{i=1}^{c} q_i = 1$ for $i = 2, 3, \ldots, c$ yields c equations and c unknowns. The choice of the cutoff c depends on the distribution and values of k_i. For some c, the following is true,

$$\begin{bmatrix} 1 & -1 & 0 & & \cdots & 0 \\ 1 & 0 & -1 & 0 & \cdots & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & -1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \vdots \\ q_c \end{bmatrix} = \begin{bmatrix} \frac{1}{E[Q_{cache}]} \ln\left(\frac{k_1}{k_2}\right) \\ \frac{1}{E[Q_{cache}]} \ln\left(\frac{k_1}{k_3}\right) \\ \frac{1}{E[Q_{cache}]} \ln\left(\frac{k_1}{k_4}\right) \\ \vdots \\ \frac{1}{E[Q_{cache}]} \ln\left(\frac{k_1}{k_c}\right) \\ 1 \end{bmatrix} \tag{4.39}$$

4.3.2 q_1

q_1 can be solved by summing up the left hand side and equating it to the sum of the right hand side.

$$\begin{aligned} cq_1 &= \left(\frac{1}{E[Q_{cache}]} \sum_{i=2}^{c} \ln\left(\frac{k_1}{k_i}\right)\right) + 1 \\ &= 1 + \frac{1}{E[Q_{cache}]} \left((c-1)\ln(k_1) - \sum_{i=2}^{c} \ln(k_i)\right) \\ &= 1 + \frac{(c-1)\ln(k_1)}{E[Q_{cache}]} - \frac{1}{E[Q_{cache}]} \sum_{i=2}^{c} \ln(k_i) \\ q_1 &= \underbrace{\frac{1}{c} - \frac{1}{cE[Q_{cache}]} \sum_{i=2}^{c} \ln(k_i)}_{\text{positive}} + \underbrace{\frac{(c-1)\ln(k_1)}{cE[Q_{cache}]}}_{\text{negative}} \end{aligned} \tag{4.40}$$

Since $k_i \leq 1$, $\ln k_i$ is negative. q_1 is valid if $0 \leq q_1 \leq 1$. Therefore, q_1 is valid if,

$$\begin{aligned} 0 &\leq \left(\frac{1}{c} - \frac{1}{cE[Q_{cache}]} \sum_{i=2}^{c} \ln(k_i)\right) + \frac{(c-1)\ln(k_1)}{cE[Q_{cache}]} \\ &\leq E[Q_{cache}] - \sum_{i=2}^{c} \ln(k_i) + (c-1)\ln(k_1) \\ \sum_{i=2}^{c} \ln(k_i) - (c-1)\ln(k_1) &\leq E[Q_{cache}] \end{aligned} \tag{4.41}$$

and,

$$\begin{aligned}
\left(\frac{1}{c} - \frac{1}{cE[Q_{cache}]}\sum_{i=2}^{c}\ln(k_i)\right) + \frac{(c-1)\ln(k_1)}{cE[Q_{cache}]} &\leq 1 \\
E[Q_{cache}] - \sum_{i=2}^{c}\ln(k_i) + (c-1)\ln(k_1) &\leq cE[Q_{cache}] \\
-\sum_{i=2}^{c}\ln(k_i) + (c-1)\ln(k_1) &\leq (c-1)E[Q_{cache}] \\
-\frac{1}{c-1}\sum_{i=2}^{c}\ln(k_i) + \ln(k_1) &\leq E[Q_{cache}] \\
\ln(k_1) - \frac{1}{c-1}\sum_{i=2}^{c}\ln(k_i) &\leq E[Q_{cache}] \qquad (4.42)
\end{aligned}$$

If q_1 is not valid by Equations 4.41 and 4.42 for any value of $c > 1$, then $c = 1$, $q_1 = 1$ and $q_{i\geq 2} = 0$. In practise, this is should never be the case if there are a reasonable number of idlers. Usually, there may be many popular videos with high arrival rates such that $k_i \approx 1$ for these videos. Suppose there are $c_{k=1}$ videos with $k = 1$ and $\ln(k) = 0$. If the cutoff is chosen to be $c_{k=1}$, then Equations 4.41 and 4.42 becomes,

$$\begin{aligned}
0 \leq \frac{1}{c_{k=1}} - \frac{1}{c_{k=1}E[Q_{cache}]}\sum_{i=2}^{c_{k=1}}\ln(k_i) + \frac{(c_{k=1}-1)\ln(k_1)}{c_{k=1}E[Q_{cache}]} \leq 1 \\
0 \leq \frac{1}{c_{k=1}} \leq 1 \qquad (4.43)
\end{aligned}$$

In fact, if there are many videos with the same arrival rate and the cache only includes these videos, then naturally, these videos should receive equal probability to be in the cache. Therefore, $q_1 = q_2 = \cdots = q_{c_{k=1}} = \frac{1}{c_{k=1}}$.

4.3.3 $q_{j,\ 2 \leq j \leq c}$

If q_1 is valid, then from Equation 4.38 for $\forall q_{j,\ 2\leq j\leq c}$,

$$\begin{aligned}
q_j &= q_1 - \frac{\ln \frac{k_1}{k_j}}{E[Q_{cache}]} \\
&= q_1 - \frac{\ln k_1}{E[Q_{cache}]} + \frac{\ln k_j}{E[Q_{cache}]} \\
&= \frac{1}{c} + \frac{(c-1)\ln(k_1)}{cE[Q_{cache}]} - \frac{1}{cE[Q_{cache}]}\sum_{i=2}^{c}\ln(k_i) - \frac{\ln k_1}{E[Q_{cache}]} + \frac{\ln k_j}{E[Q_{cache}]} \\
&= \frac{1}{c} - \frac{\ln(k_1)}{cE[Q_{cache}]} - \frac{1}{cE[Q_{cache}]}\sum_{i=2}^{c}\ln(k_i) + \frac{\ln k_j}{E[Q_{cache}]} \\
&= \frac{1}{c} - \frac{1}{cE[Q_{cache}]}\sum_{i=1}^{c}\ln(k_i) + \frac{\ln k_j}{E[Q_{cache}]} \qquad (4.44)
\end{aligned}$$

Again, for q_j to be valid, $0 \leq q_j \leq 1$.

$$\begin{aligned}
0 &\leq \frac{1}{c} - \frac{1}{cE[Q_{cache}]}\sum_{i=1}^{c}\ln(k_i) + \frac{\ln k_j}{E[Q_{cache}]} \leq 1 \\
0 &\leq E[Q_{cache}] - \sum_{i=1}^{c}\ln(k_i) + c\ln(k_j) \leq cE[Q_{cache}] \\
-E[Q_{cache}] &\leq -\sum_{i=1}^{c}\ln(k_i) + c\ln(k_j) \leq (c-1)E[Q_{cache}] \qquad (4.45)
\end{aligned}$$

4.3.4 Cutoff

In general, the cutoff c is dependent on k_i and $E[Q_{cache}]$. If q_1 is valid, then Equation 4.45 can be used to validate all q_js up to q_c. A cutoff can be used if $\forall q_{j,\ 1\leq j\leq c},\ 0 \leq q_j \leq 1$.

$$-E[Q_{cache}] \leq \underbrace{-\sum_{i=1}^{c}\ln(k_i)}_{\text{positive}} + \underbrace{c\ln(k_j)}_{\text{negative}} \leq (c-1)E[Q_{cache}]$$

Let

$$Z_c(j) = -\sum_{i=1}^{c}\ln(k_i) + c\ln(k_j) \qquad (4.46)$$

where the first term is constant for some c and k_is and the second term is dependent on k_j. If Equation 4.45 is to be valid for all j, then the maximum and minimum values of $Z_c(j)$ must be valid. Then, everything in between will also be valid.

For some c, $Z_c(j)$ is maximum when $c\ln(k_j)$ is the smallest negative number. This occurs when $j = 2$, since k_2 is the largest $k_j \leq 1$. Also, $Z_c(j)$ is minimum when $c\ln(k_j)$ is the largest negative value. This occurs when $j = c$. Therefore, for a given c, the two following requirements must be met,

$$Z_c(2) \leq (c-1)E[Q_{cache}] \tag{4.47}$$
$$-E[Q_{cache}] \leq Z_c(c) \tag{4.48}$$

Optimal cutoff

For a particular $\{k_i\}$ and $E[Q_{cache}]$, the optimal cutoff occurs for the largest c such that Equations 4.47 and 4.48 are valid. However, since $\{k_i\}$ and $E[Q_{cache}]$ may change over time and even during the same day, choosing c may be a difficult open problem.

4.4 Single Video Cache for YouTube

The total number of views on the date of data scrape is shown in Figure 4.7.

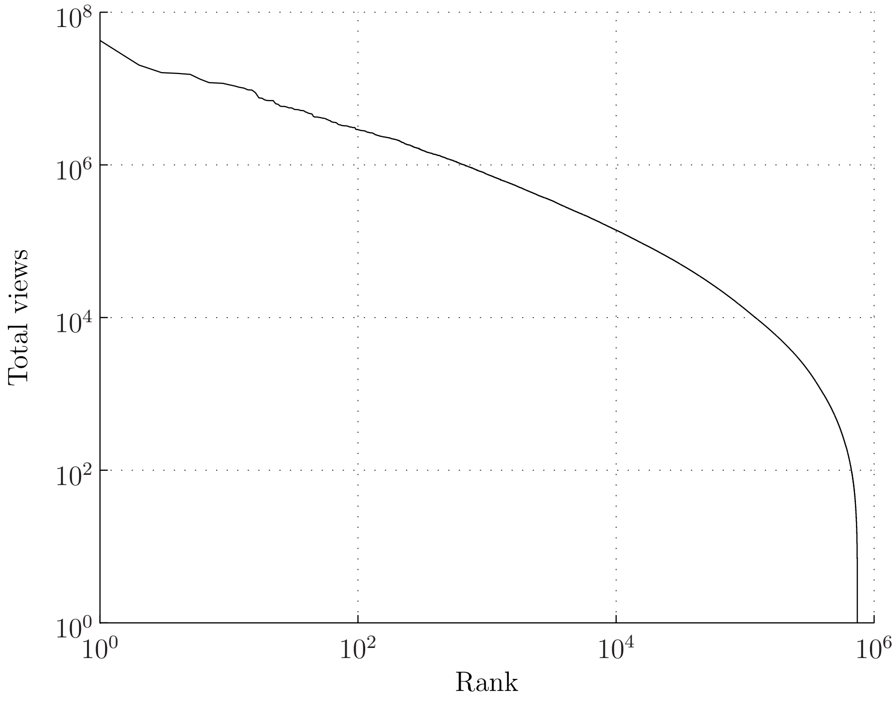

Figure 4.7: Total views by rank.

The mean arrival rate of requests is shown in Figure 4.8. This is the number of views divided by the amount of time the video has been on the site.

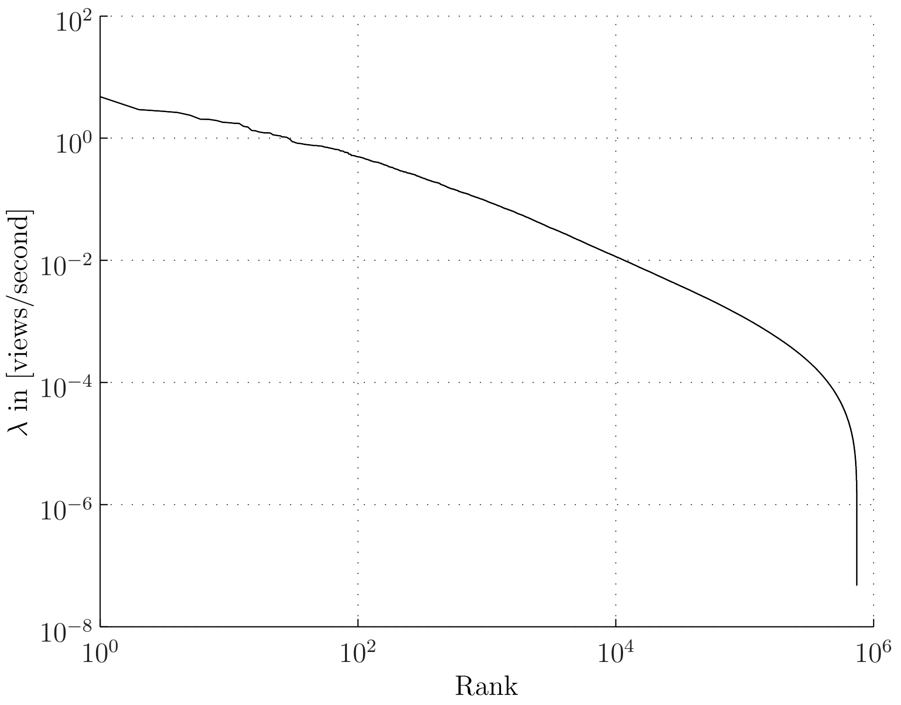

Figure 4.8: Arrival rate by rank.

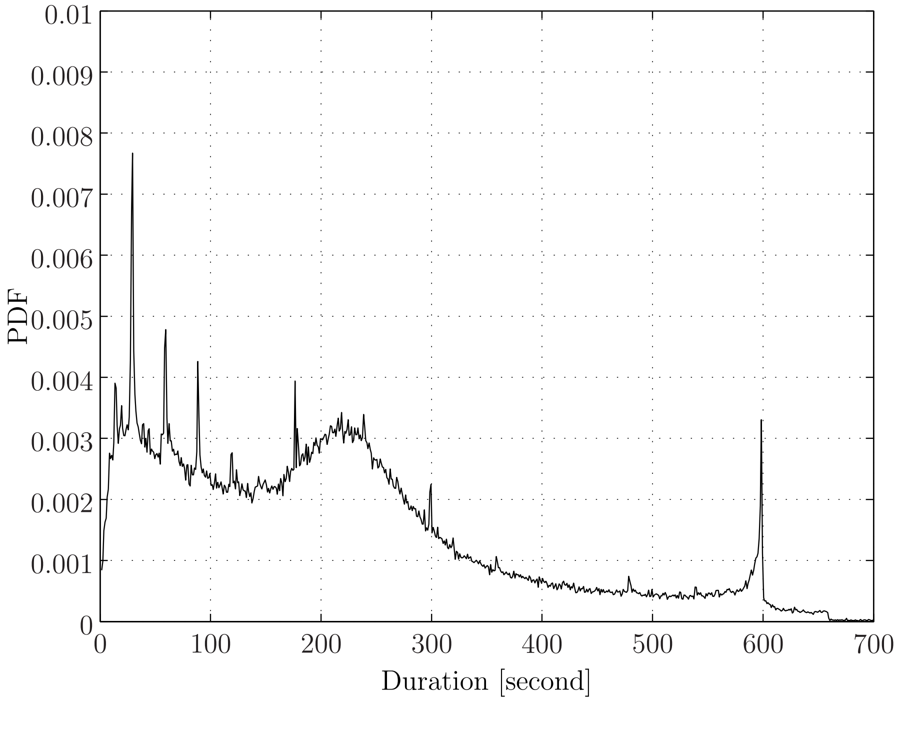

Figure 4.9: Video length distribution.

The distribution of video lengths is shown in Figure 4.9. Fortunately, this is very similar to Figure 3 of [25].

The distribution of ρ is shown in Figure 4.10. The distribution is logarithmically centered at about 0.03.

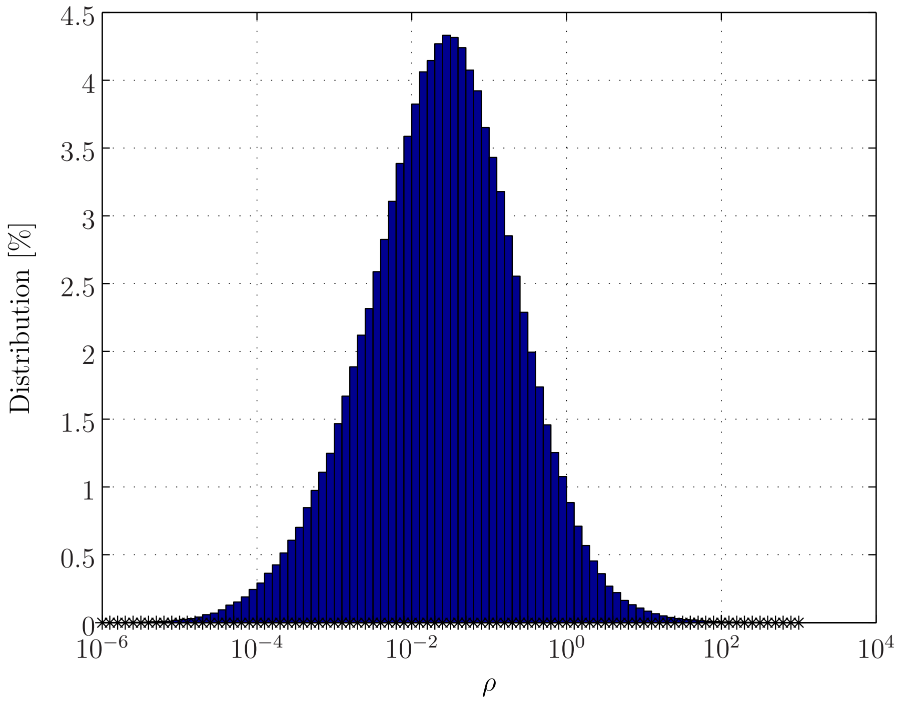

Figure 4.10: Distribution of ρ for trace data. 10 logarithmically spaced bins were used per decade.

Let $k_i = 1 - e^{-\rho_i}$. Then the distribution of k_i is given in Figure 4.11. The spike at $k = 1$ represents videos that receive 0 or very few views.

$\ln(k)$ is the most important parameter for caching purposes. The distribution of $\ln(k)$ is given in Figure 4.12.

Based on the data for $\ln(k)$, it is possible to find a suitable cutoff c for various values of $E[Q_{cache}]$. From Equation 4.28, $E[Q_{cache}] = \eta \sum_{i=1}^{N} \rho_i$. Also from Equations 4.47 and 4.48, the bounds on Z_c is given as,

$$\begin{aligned} Z_c(2) &\leq (c-1)E[Q_{cache}] \\ -E[Q_{cache}] &\leq Z_c(c) \end{aligned}$$

where,

$$Z_c(j) = -\sum_{i=1}^{c} \ln(k_i) + c\ln(k_j)$$

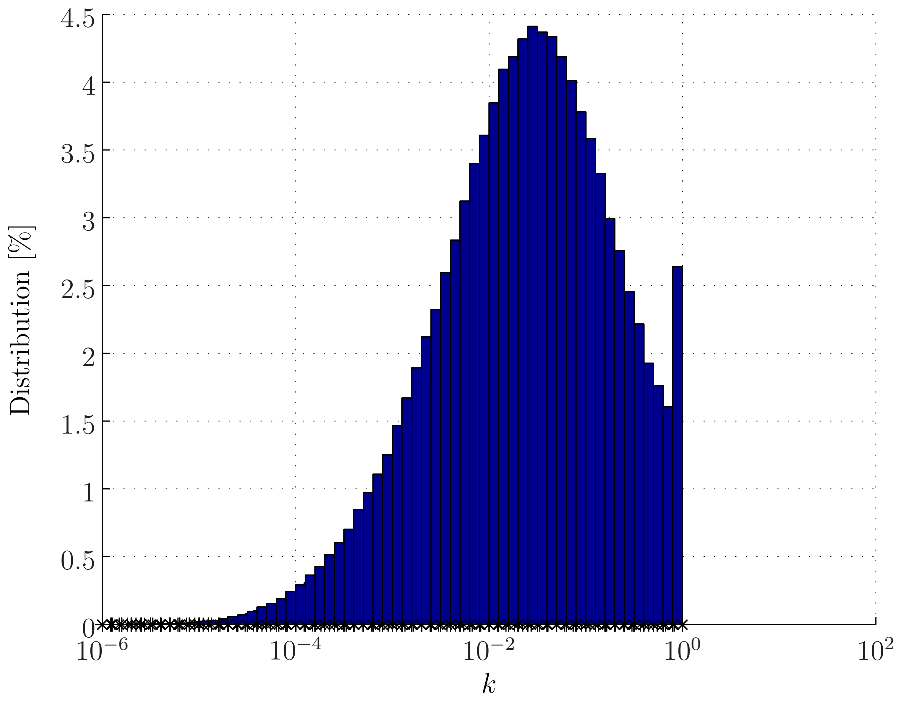

Figure 4.11: Distribution of k for trace data. 10 logarithmically spaced bins were used per decade.

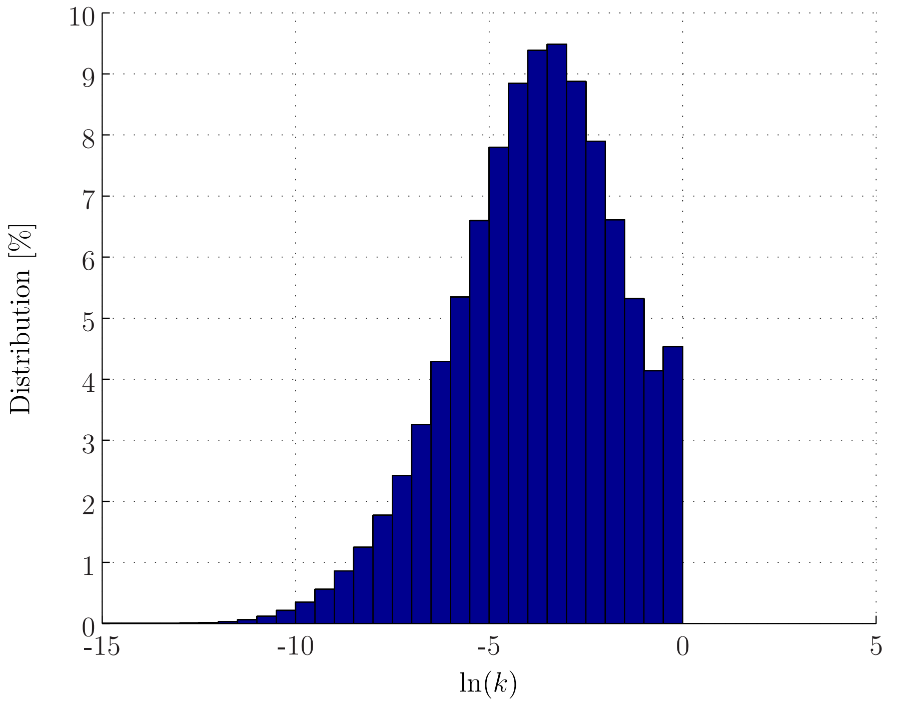

Figure 4.12: Distribution of $\ln(k)$.

Substituting and rearranging these equations yields the following two necessary conditions:

Condition 1.

$$\begin{aligned} Z_c(2) &\leq (c-1)E[Q_{cache}] \\ \frac{Z_c(2)}{c-1} &\leq E[Q_{cache}] \\ \frac{Z_c(2)}{c-1} &\leq \eta \sum_{i=1}^{N} \rho_i \\ \frac{1}{\sum_{i=1}^{N} \rho_i} \frac{Z_c(2)}{c-1} &\leq \eta \end{aligned} \tag{4.49}$$

Condition 2.

$$\begin{aligned} -E[Q_{cache}] &\leq Z_c(c) \\ -Z_c(c) &\leq E[Q_{cache}] \\ \frac{-Z_c(c)}{\sum_{i=1}^{N} \rho_i} &\leq \eta \end{aligned} \tag{4.50}$$

Plotting these conditions using empirical data produces Figure 4.13. It can be seen that as η, or the amount of idlers in the system increases, more videos should be cached globally. It can also be seen that Condition 2 is dominant, meaning that if Condition 2 is satisfied, Condition 1 is automatically satisfied for this data set. As such, the optimal c occurs when $\frac{-Z_c(c)}{\sum_{i=1}^{N} \rho_i} = \eta$.

If no users seed, then $\eta = 0$. This is the worst case scenario. Another scenario is if users seed until they make a decision to go to another web page. According to [96], users tend to wait about 9 to 12 seconds before choosing another video. This is due to the time for thumbnails of related videos to show up. For the trace, the mean duration of videos is 208 seconds. Therefore, an idling time of 10 seconds amounts to $\eta = \frac{10}{208} \approx 0.05$. A 2009 Nielsen report [97] suggests that on average, users spend 29 hours and 15 minutes per month on the Internet. Of this, 3 hours is spent watching videos. Therefore, if users kindly seed for the whole time on the Internet while not spent watching videos, this would translate to $\eta = \frac{29.25-3}{3} = 8.75$. This is essentially the best case scenario. Table 4.2 shows the optimal c for some values of η. The total number of videos in this data set is 743308.

For each $\{\eta, c\}$ pair, it is possible to calculate the optimal q_is using Equations 4.40 and 4.44. The resulting q_is which together represent the probability density function are shown in Figure 4.14.

Finally, it is possible to calculate the expected server traffic per unit time from Equation 4.30. Normalizing by the server traffic when no P2P traffic is allowed, the result gives the expected server usage.

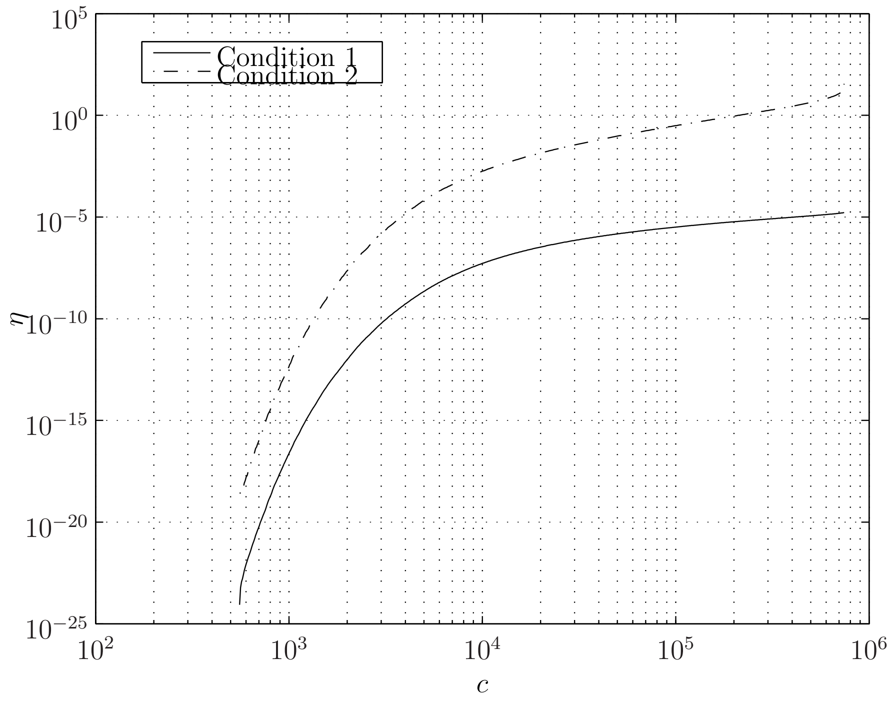

Figure 4.13: Conditions on Z_c for η by c.

Table 4.2: η and optimal c

η	c	% of total videos
0	0	0
0.05	35797	4.8
1	211819	28.5
2	328218	44.2
4	482974	65.0
8.75	655740	88.2

This data suggests that server usage can be dramatically reduced when peers upload to others.

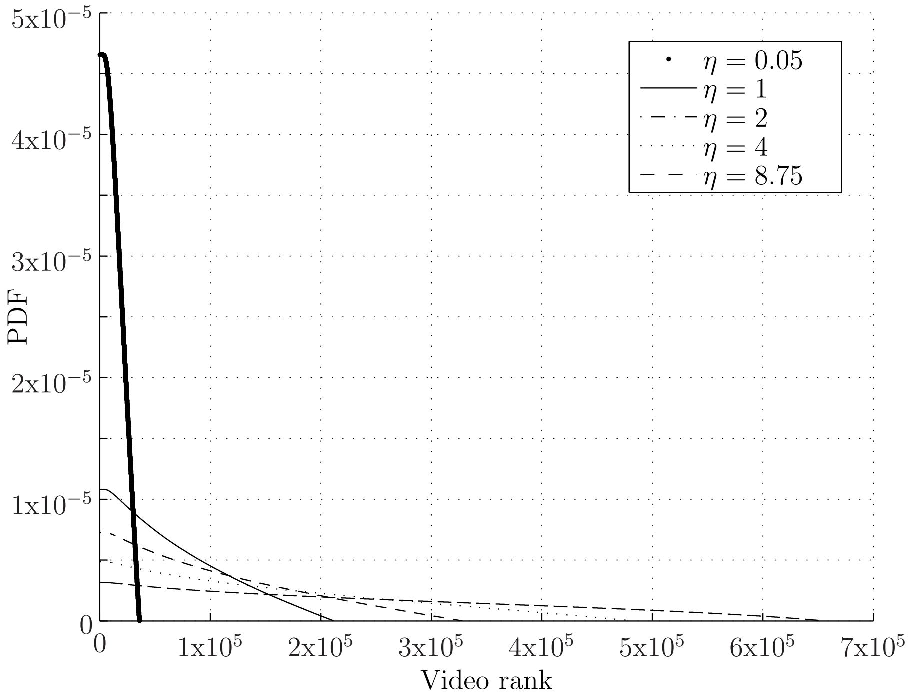

Figure 4.14: Video probability density for different proportion of idlers.

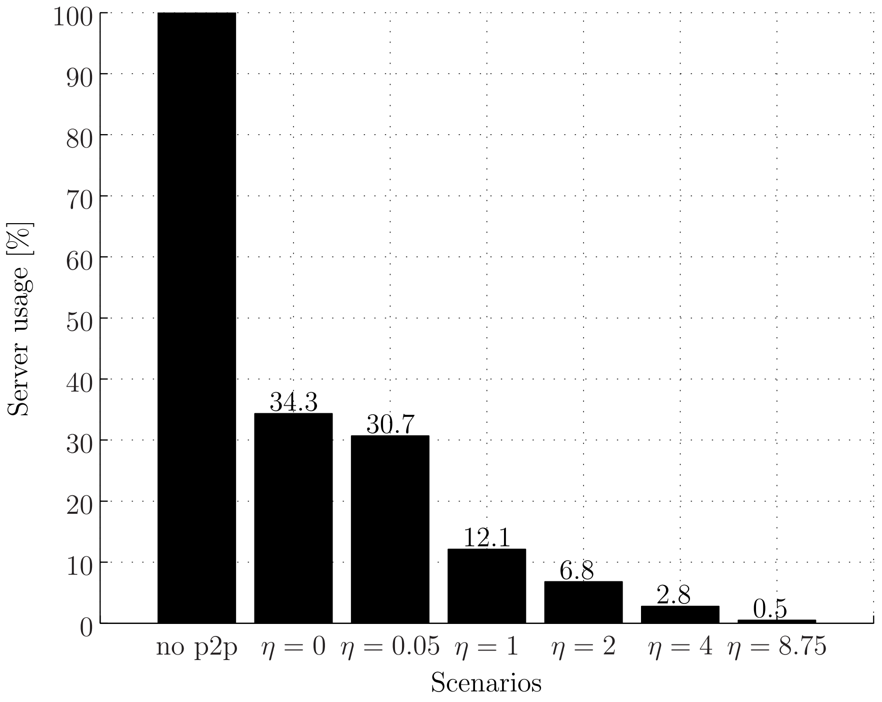

Figure 4.15: Server usage for different idler proportions.

Chapter 5

Summary

A model for peer-to-peer transfers of short online videos was discussed. Currently, YouTube is the most popular online video web site today. By some estimates, the bandwidth cost of YouTube will be $362 million in 2009 [4]. Therefore, offloading the server requirements to the users would be helpful. Existing peer-to-peer systems such as BitTorrent focuses on transferring files in bulk, where files can only be used after it has been entirely downloaded. By contrast, YouTube videos can be downloaded as it is being watched without any waiting.

The system model consists of three components: servers, viewers and idlers. Servers have infinite capacity. Viewers are peers who are currently viewing, and thus downloading a particular video. Idlers are those peers who are still in the system but are not actively downloading or watching a video. The model assumes that viewers can download and upload at the bitrate. Each new viewer of a particular video downloads from the viewer that arrived before it or the server if no such viewer exists. This new viewer in turn supplies the next viewer that arrives. Viewers stop uploading as soon as they finish downloading the video. Therefore, regardless of the number of users watching a particular video, the server must supply only one copy of the video.

An improvement of this model uses idling peers. Idlers are assumed to have a finite cache to store videos. To minimize the server traffic, the aim is to utilize idlers as much as possible. The model assumes a cache of size one to simplify the analysis and arrives at an optimum global cache distribution for videos based on their probability of being watched at any given time. Popular videos are well represented and extremely unpopular videos are never cached. It is shown that it is not optimal for idlers to cache their last watched video.

The results show that by using the peer-to-peer model without idlers, server bandwidth is reduced by 66%. If the average user idles for half the time while on YouTube and contributes their bandwidth, the server bandwidth is reduced by 88% as compared to no peer-to-peer. These are lower bounds on cost savings since server costs are usually attributed to peak traffic which happens to be the time when peer-to-peer systems usually perform the best.

Chapter 6

Future Work

There are many possible opportunities for future work. By using simulations, it would be possible to validate the model and to incorporate second order effects. One approach can look at the effect of heterogenous bandwidth capacities and bit rates. The bit rate was fixed in the analysis and it was assumed users uploads and downloads at the bit rate. In reality, most users can download at or above the bit rate. However, some may not be able to upload at the bit rate.

Churn occurs when users join or exit the system which usually has a detrimental effect on the efficiency of the peer-to-peer system. The effect of churn can be studied. The effect of users who do not finish viewing a video or watches multiple videos at the same time can also be studied.

This study limited the cache size to a single video. One problem with multiple videos is in determining how to properly allocate a user's upload bandwidth. Caching parts of a video can also be considered to increase the variety of videos cached. Of course the current caching scheme may no longer apply. Another consideration is the replacement strategy.

In a practical setting, users may download from any peer that has a piece that is useful instead of the immediately preceding user. This would require similar considerations as BitTorrent. However, the difference is that videos are watched as they are being downloaded so a rarest first piece selection strategy will not work.

Users may watch multiple videos in sequence during a visit. For a particular video, there may exist a list of other videos which users are likely to view next. This information would be useful in prefetching other videos and acquiring peers that have those videos.

There is an inherent incentive for viewers of the same video to upload to others because they are receiving data from other peers. However, in order for idlers to participate, an incentive system may be required. With YouTube, a large central server will likely exist which is quite different from existing peer-to-peer system which have limited server resources. Any credits gained can be used to access higher quality videos or a faster download rate from the server.

Searching for peers and videos are very important aspects of a peer-to-peer system. From a user experience point of view, both are more effectively implemented in a centralized fashion. However, some techniques from Kademlia may be useful. In addition, YouTube or copyright holders may want to put limits on whether videos can be obtained in a peer-to-peer fashion so the central servers can act as a security mechanism.

The peer-to-peer system must consider users who do not wish to participate in the peer-to-peer activity. Older browsers or users who do not wish to download additional software should be able to at least achieve the current level of service.

Since it takes some time to find suitable and willing peers to download from in a P2P network, it would be worthwhile to download the beginning of the video from the central server. Thereafter, the majority of the transfer should be from peers. In case peers leave unexpectedly, the server needs to help immediately. Generations and source coding may be useful. All of these practical considerations may yield interesting results.

Appendix A

BitTorrent .torrent file

A.1 BitTorrent specification

Although BitTorrent is now closed source, the protocol specification is available online. Two good sources are [98] and [99].

A.2 ubuntu-8.04.2-desktop-i386.iso.torrent

The following sample .torrent file contains a single image file for the open source Ubuntu Operating System 8.04.2 LTS (Hardy Heron) [100]:

```
d8:announce
39:http://torrent.ubuntu.com:6969/announce
7:comment
29:Ubuntu CD releases.ubuntu.com
13:creation datei1232655717e
4:info
d6:lengthi731869184e
4:name
30:ubuntu-8.04.2-desktop-i386.iso
12:piece lengthi524288e
6:pieces 27920:file hashes
```

The file hashes have been omitted.

Appendix B

Cheng et al YouTube trace

The data set for Cheng et al's paper, "Statistics and Social Network of YouTube Videos" [25] is available at `http://netsg.cs.sfu.ca/youtubedata/`.

B.1 Methodology

The following description is taken from the webpage: We consider all the YouTube videos to form a directed graph, where each video is a node in the graph. If a video b is in the related video list (first 20 only) of a video a, then there is a directed edge from a to b. Our crawler uses a breadth-first search to find videos in the graph. We define the initial set of 0-depth video IDs, which the crawler reads in to a queue at the beginning of the crawl. When processing each video, it checks the list of related videos and adds any new ones to the queue.

Given a video ID, the crawler first extracts information from the YouTube API, which contains all the meta-data except age, category and related videos. The crawler then scrapes the video's webpage to obtain the remaining information.We record the following information of a YouTube video in order; they are divided by '\t' in the data file.

Our first crawl was on February 22nd, 2007, and started with the initial set of videos from the list of "Recently Featured", "Most Viewed", "Top Rated" and "Most Discussed", for "Today", "This Week", "This Month" and "All Time", which totalled 189 unique videos on that day. The crawl went to more than four depths, finding approximately 750 thousand videos in about five days. In the following weeks we ran the the crawler every two to three days, each time defining the initial set of videos from the list of "Most Viewed", "Top Rated", and "Most Discussed", for "Today" and "This Week", which is about 200 to 300 videos. On average, the crawl finds 73 thousand distinct videos each time in less than 9 hours.

All the 35 datasets can be downloaded from here. In each package, there are: (1) "0.txt", "1.txt", "2.txt" and "3.txt" (and "4.txt" in "0222.zip"), storing the

Table B.1: Cheng et al data set schema.

Data	Description
video ID	an 11-digit string, which is unique
uploader	a string of the video uploader's username
age	an integer number of days between the date when the video was uploaded and Feb.15, 2007 (YouTube's establishment)
category	a string of the video category chosen by the uploader
length	an integer number of the video length
views	an integer number of the views
rate	a float number of the video rate
ratings	an integer number of the ratings
comments	an integer number of the comments
related IDs	up to 20 strings of the related video IDs

video data of depth 0, 1, 2 and 3 respectively; (2) "log.txt", a log file indicating the start and finsh time, number of videos crawled and the duration of each depth.

B.2 Trace used

For the purpose of this study, the largest trace file from February 22, 2007 was used. It had a total of 749361 videos. The age column was used to calculate the number of days the video has been online as of February 22, 2007. The number of views together with this value was used to calculate the average video request rate.

Appendix C

Queueing Theory Background

C.1 M/M/∞ queue

A simple case of the M/G/∞ queue is the M/M/∞ queue. The only difference between the two is the distribution of the service times. With the M/M/∞ queue, service times are exponentially distributed with mean $1/\mu$. The flow diagram for the M/M/∞ queue is shown in Figure C.1 [6]. This system is represented as a birth-death process, where each state is the number of users. If a new arrival occurs, the number of users in the system ρ increases by one. The probability of this happening is a Poisson distribution with parameter λ. When there are n users in the system, the rate at which they leave is an exponential distribution with rate $n\mu$.

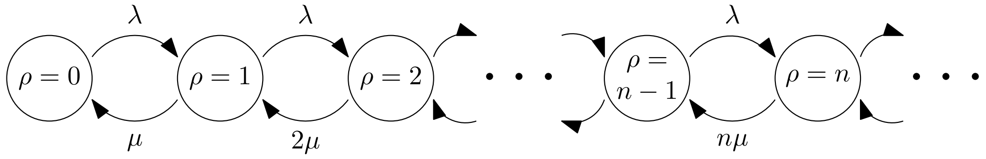

Figure C.1: Flow diagram for M/M/∞ model [6].

Let the probability that there are n viewers in the system be specified by p_n. In steady-state, queueing theory specifies that the flow from state $n-1$ to n and the flow from n to $n-1$ is equal [6]. Thus,

$$\begin{aligned} p_0\lambda &= \mu p_1 \\ p_1\lambda &= 2\mu p_2 \\ &\cdots \\ p_{n-2}\lambda &= (n-1)\mu p_{n-1} \\ p_{n-1}\lambda &= n\mu p_n \end{aligned} \tag{C.1}$$

Setting $\rho = \frac{\lambda}{\mu}$ and solving for p_n gives,

$$\begin{aligned} p_n &= \frac{\lambda}{n\mu} p_{n-1} \\ &= \frac{\rho}{n} p_{n-1} \\ &= \frac{\rho}{n}\left(\frac{\rho}{n-1} p_{n-2}\right) \\ &= \frac{\rho^2}{n(n-1)} p_{n-2} \\ &\cdots \\ &= \frac{\rho^n}{n!} p_0 \end{aligned} \tag{C.2}$$

Since the probabilities p_n add up to 1, it is possible to solve for p_0,

$$\begin{aligned} 1 &= \sum_{n=0}^{\infty} p_n \\ &= \sum_{n=0}^{\infty} \frac{\rho^n}{n!} p_0 \\ \frac{1}{p_0} &= \sum_{n=0}^{\infty} \frac{\rho^n}{n!} \\ &= e^{\rho} \end{aligned} \tag{C.3}$$

Thus,

$$p_0 = e^{-\rho} \tag{C.4}$$

and

$$p_n = \frac{\rho^n}{n!} e^{-\rho} \tag{C.5}$$

Therefore, the number of viewers watching a particular video has a Poisson distribution with mean ρ.

C.2 M/G/∞ queue

If the service times follow an arbitrary distribution, then this distribution can be described with service times b_i and probabilities q_i, such that the probability of the service time, B, being b_i is q_i [6].

$$P(B = b_i) = q_i, \quad i = 1, 2, \ldots, N \tag{C.6}$$

A Poisson arrival process with rate λ can be split into independent Poisson streams $i = 1, 2, \ldots, N$ so that the intensity of stream i is λq_i. Then, the viewers arriving at each of these streams then see a constant service time b_i.

For a constant service time b_i, the probability that there is exactly n customers at time t is equal to the probability that between $t - b_i$ and t exactly n customers arrived. Because the number of customers arriving in a time interval of length b_i is Poisson distributed with mean $\lambda q_i b_i$,

$$p_n(t) \quad = \quad \frac{(\lambda q_i b_i)^n}{n!} e^{-\lambda q_i b_i} \tag{C.7}$$

which is valid for all $t > b$, and so is valid in steady-state [6].

Because the sum of independent Poisson random variables is also Poisson, the total number of customers in the system is Poisson distributed with mean ρ,

$$\rho \quad = \quad \sum_{i=0}^{N} \lambda q_i b_i \tag{C.8}$$

Therefore, for any given service distribution, M/G/∞, with mean service time $1/\mu$, the number of viewers in the system is Poisson distributed with ρ:

$$p_n \quad = \quad \frac{\rho^n}{n!} e^{-\rho} \tag{C.9}$$

where $\rho = \lambda/\mu$. This is the same as the M/M/∞ case.

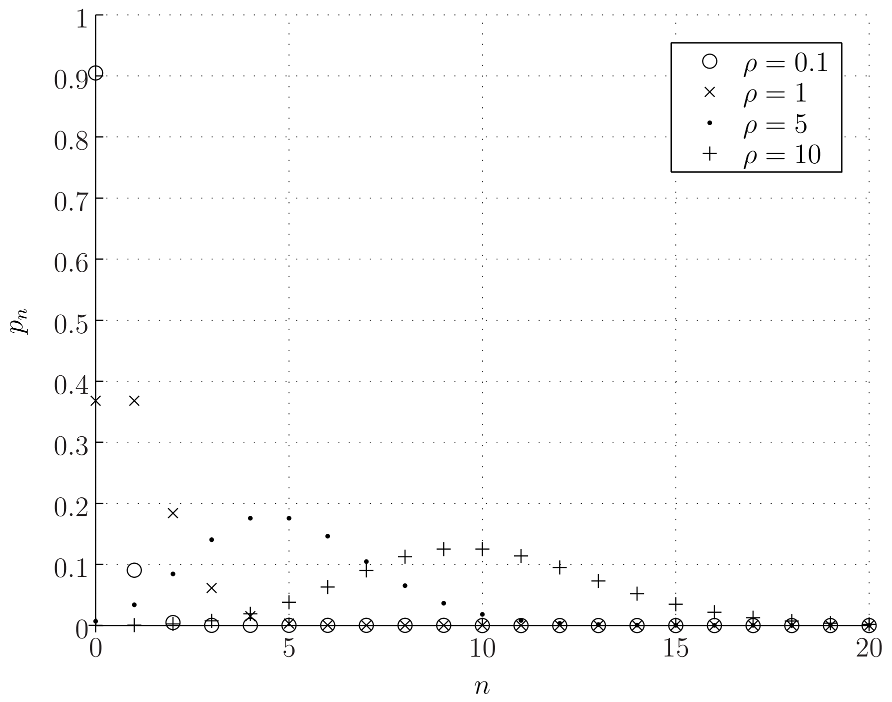

Figure C.2: p_n for different ρ. It can be seen that for ρ less than 1, p_n is dominated by small values of n. For larger ρ, p_0 and p_1 is very small.

References

[1] Ipoque, “Internet study 2008/2009,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009

[2] N. Anderson, “The youtube effect: Http traffic now eclipses p2p,” 2007, [Online; accessed 6-March-2009]. [Online]. Available: http://arstechnica.com/old/content/2007/06/the-youtube-effect-http-traffic-now-eclipses-p2p.ars

[3] Business Wire, “Ellacoya data shows web traffic overtakes peer-to-peer (p2p) as largest percentage of bandwidth on the network,” 2007, [Online; accessed 6-March-2009]. [Online]. Available: http://www.businesswire.com/portal/site/google/index.jsp?ndmViewId=news_view&newsId=20070618005912&newsLang=en

[4] T. Spangler, “Youtube may lose $470 million in 2009: Analysts,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.multichannel.com/article/191223-YouTube_May_Lose_470_Million_In_2009_Analysts.php

[5] IMDb.com Inc, “Imdb statistics,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.imdb.com/database_statistics

[6] I. Adan and J. Resing, “Queueing theory,” 2001, [Online; accessed 25-May-2009]. [Online]. Available: http://www.cs.duke.edu/~fishhai/misc/queue.pdf

[7] Arbor Networks, “Ddos protection — deep packet inspection — ip flow analysis — arbor networks,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.arbornetworks.com

[8] E. Palm, “Net traffic down on first day of swedish antipiracy law,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://news.cnet.com/8301-1023_3-10209544-93.html

[9] Y. Adegoke, “Youtube and universal talk on music video site,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.reuters.com/article/technologyNews/idUSTRE5240AZ20090305

[10] Wikipedia, “Youtube — wikipedia, the free encyclopedia,” 2009, [Online; accessed 6-April-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=YouTube&oldid=281978317

[11] USA Today, “Youtube serves up 100 million videos a day online,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.usatoday.com/tech/news/2006-07-16-youtube-views_x.htm

[12] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, “Youtube traffic characterization: a view from the edge,” in *IMC ’07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007, pp. 15–28.

[13] Adobe Systems Incorporated, “Adobe flash player version penetration,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

[14] ——, “Video learning guide for flash: Progressive and streaming video,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.adobe.com/devnet/flash/articles/video_guide_02.html

[15] T. Jacobs, “Foxtorrent::add-ons for firefox,” 2009, [Online; accessed 3-July-2009]. [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/4844

[16] Opera Software ASA, “Opera integrates bittorrent in upcoming browser,” 2009, [Online; accessed 3-July-2009]. [Online]. Available: http://www.opera.com/press/releases/2006/02/06/

[17] Little Shoot, “P2p file sharing browser plugin littleshoot bittorrent gnutella,” 2009, [Online; accessed 3-July-2009]. [Online]. Available: http://www.littleshoot.org/

[18] N. Gonzalez, “Slapvid: Peer to peer video in your browser,” 2007, [Online; accessed 3-July-2009]. [Online]. Available: http://www.techcrunch.com/2007/06/25/slapvid-peer-to-peer-video-in-your-browser/

[19] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, “I tube, you tube, everybody tubes: analyzing the world’s largest user generated content video system,” in *IMC ’07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007, pp. 1–14.

[20] YouTube Inc., “Youtube - broadcast yourself.” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.youtube.com/browse

[21] Blockbuster Inc, “Blockbuster online - gifts,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.blockbuster.com/gifts

[22] Zip.ca Inc., "Zip.ca," 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.zip.ca/dvd/index.aspx

[23] Netflix Inc., "Netflix: Action, blu-ray, comedy, drama, romance," 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.netflix.com/BrowseSelection

[24] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes," 2007, [Online; accessed 6-March-2009]. [Online]. Available: http://an.kaist.ac.kr/traces/IMC2007.html

[25] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, June 2008, pp. 229–238.

[26] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of youtube network traffic at a campus network - measurements, models, and implications," *Computer Networks*, vol. 53, no. 4, pp. 501 – 514, 2009, content Distribution Infrastructures for Community Networks. [Online]. Available: http://www.sciencedirect.com/science/article/B6VRG-4TVJNT3-2/2/7b9613c14704145d8647ceeeea233d77

[27] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: a cooperative bulk data transfer protocol," *INFOCOM 2004*, vol. 2, pp. 941–951, March 2004.

[28] Akamai Technologies, "Akamai: The leader in web application acceleration and performance management, streaming media services and content delivery," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.akamai.com

[29] Limelight Networks, "Limelight networks," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.limelightnetworks.com

[30] CDNetworks, "Cdnetworks :: Global cdn service leader," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.us.cdnetworks.com

[31] B. Cohen, "Bittorrent," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.bittorrent.com

[32] Octoshape, "Octoshape - large scale live streaming solutions," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.octoshape.com

[33] Wikipedia, "Application service provider — wikipedia, the free encyclopedia," 2009, [Online; accessed 9-February-2009]. [Online]. Available: http://en.wikipedia.org/wiki/Application_service_provider

[34] ——, "Content delivery network — wikipedia, the free encyclopedia," 2009, [Online; accessed 9-February-2009]. [Online]. Available: http://en.wikipedia.org/wiki/Content_Delivery_Network

[35] ——, "Domain name system — wikipedia, the free encyclopedia," 2009, [Online; accessed 9-February-2009]. [Online]. Available: http://en.wikipedia.org/wiki/Domain_name_system

[36] Slashdot, "Slashdot - news for nerds, stuff that matters," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.slashdot.org

[37] Digg Inc., "Digg - all news, videos, & images," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.digg.com

[38] Wikipedia, "Slashdot effect — wikipedia, the free encyclopedia," 2009, [Online; accessed 9-February-2009]. [Online]. Available: http://en.wikipedia.org/wiki/Slash_dot_effect

[39] Canonical Ltd., "Ubuntu home page — ubuntu," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.ubuntu.com

[40] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," *SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 205–217, 2002.

[41] Y.-H. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast (keynote address)," in *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, New York, NY, USA, 2000, pp. 1–12.

[42] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1489–1499, Oct 2002.

[43] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended lans," *ACM Transactions on Computer Systems*, vol. 8, no. 2, pp. 85–110, 1990.

[44] V. N. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 178–190.

[45] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computer Surveys*, vol. 36, no. 4, pp. 335–371, 2004.

[46] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[47] eBay, “Skype official website - download skype free now for free calls and internet calls,” 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.skype.com

[48] Stanford University, “Folding@home - main,” 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://folding.stanford.edu

[49] University of California at Berkeley, “Seti@home,” 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://setiathome.berkeley.edu/

[50] A. D. Keromytis, V. Misra, and D. Rubenstein, “Sos: An architecture for mitigating ddos attacks,” *IEEE Journal on Selected Areas of Communications*, vol. 22, pp. 176–188, 2004.

[51] M. Christen, “Kit search engine initiative,” 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://sciencenet.fzk.de

[52] N. H. Ryeng and K. Nørvåg, “Robust aggregation in peer-to-peer database systems,” in *IDEAS ’08: Proceedings of the 2008 international symposium on Database engineering and applications*. New York, NY, USA: ACM, 2008, pp. 29–37.

[53] D. Stutzbach and R. Rejaie, “Understanding churn in peer-to-peer networks,” in *IMC ’06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2006, pp. 189–202.

[54] J. R. Douceur, “The sybil attack,” in *IPTPS ’01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 251–260.

[55] Napster LLC, “Napster.com - all the music you want. any way you want it.” 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.napster.com

[56] Wikipedia, “Napster — wikipedia, the free encyclopedia,” 2009, [Online; accessed 22-February-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Napster&oldid=271488033

[57] Howstuffworks, “How the old napster worked,” 2009, [Online; accessed 22-February-2009]. [Online]. Available: http://computer.howstuffworks.com/napster.htm/printable

[58] Limewire, “Gnutella protocol specification (gdf),” 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://wiki.limewire.org/index.php?title=GDF

[59] Nullsoft, “nullsoft.com,” 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.nullsoft.com

[60] AOL LLC, "Aol.com - welcome to aol," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.aol.com

[61] Wikipedia, "Gnutella — wikipedia, the free encyclopedia," 2009, [Online; accessed 23-February-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Gnutella&oldid=272390898

[62] Gnutelliums, LLC, "Gnutella forums," 2009, [Online; accessed 20-June-2009]. [Online]. Available: www.gnutellaforums.com

[63] Limewire, "Ultrapeers - limewire," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://wiki.limewire.org/index.php?title=Ultrapeers

[64] Musiclab LLC, "Free music downloads, download free mp3 music - bearshare.com music," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.bearshare.com

[65] Gnucleus, "Gnucleus," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.gnucleus.com

[66] Limewire, "Limewire official website & free download, bittorrent," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.limewire.com

[67] Streamcast Networks, "Morpheus (offline)," 2009, [Online; accessed 20-June-2009]. [Online]. Available: www.morpheus.com

[68] Wikipedia, "Fasttrack — wikipedia, the free encyclopedia," 2008, [Online; accessed 22-February-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=FastTrack&oldid=259345190

[69] Sharman Networks Ltd, "The anticipated next version kazaa releases this christmas 2008 download music 100% legal. get access to millions of songs. www.kazaa.com." 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.kazaa.com/us/

[70] Wikipedia, "Uuhash — wikipedia, the free encyclopedia," 2008, [Online; accessed 22-February-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=UUHash&oldid=222639704

[71] Grokster Ltd., "Grokster (offline)," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://www.grokster.com

[72] Wikipedia, "Grokster — wikipedia, the free encyclopedia," 2009, [Online; accessed 22-February-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Grokster&oldid=264116833

[73] ——, "Edonkey network — wikipedia, the free encyclopedia," 2009, [Online; accessed 24-February-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=EDonkey_network&oldid=272107387

[74] ——, “Shareaza — wikipedia, the free encyclopedia,” 2009, [Online; accessed 25-February-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Shareaza&oldid=272942869

[75] ——, “Emule — wikipedia, the free encyclopedia,” 2009, [Online; accessed 25-February-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=EMule&oldid=272625759

[76] Ernesto, “Bittorrent: The one third of all internet traffic myth,” 2006, [Online; accessed 26-February-2009]. [Online]. Available: http://torrentfreak.com/bittorrent-the-one-third-of-all-internet-traffic-myth

[77] The Pirate Bay, “Download music, movies, games, software! the pirate bay - the world’s largest bittorrent tracker,” 2009, [Online; accessed 25-February-2009]. [Online]. Available: http://www.thepiratebay.org

[78] BitTorrent Inc., “utorrent - a (very) tiny bittorrent client,” 2009, [Online; accessed 3-July-2009]. [Online]. Available: http://www.utorrent.com/

[79] Vuze, “Azureus, now called vuze:bittorrent client,” 2009, [Online; accessed 3-July-2009]. [Online]. Available: http://azureus.sourceforge.net/

[80] P. Maymounkov and D. Mazières, “Kademlia: A peer-to-peer information system based on the xor metric,” in *IPTPS ’01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 53–65.

[81] Wikipedia, “Prisoner’s dilemma — wikipedia, the free encyclopedia,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Prisoner%27s_dilemma&oldid=274374109

[82] ——, “Tit for tat — wikipedia, the free encyclopedia,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Tit_for_tat&oldid=267921348

[83] Demonoid, “Demonoid.com,” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.demonoid.com/

[84] T. Locher, P. Moor, S. Schmid, and R.Wattenhofer, “Free riding in bittorrent is cheap,” in *Proc. of HotNets-V*, Irvine, CA, USA, November 2006.

[85] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, “Do incentives build robustness in bittorrent,” in *In NSDI07*, 2007.

[86] Participatory Culture Foundation, “Miro hd video player — free internet tv and video podcast player.” 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.getmiro.com/

[87] Torrent Relay, "Torrent relay - downloads torrents online (web client)," 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www1.torrentrelay.com/fresh/web.pl?d=

[88] The XLattice Project, "Kademlia: A design specification," 2009, [Online; accessed 25-February-2009]. [Online]. Available: http://xlattice.sourceforge.net/components/protocol/kademlia/specs.pdf

[89] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht," *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–12, April 2006.

[90] A. Norberg, "Introduction to bittorrent," 2009, [Online; accessed 6-March-2009]. [Online]. Available: http://www.rasterbar.com/products/libtorrent/bittorrent.pdf

[91] The Tor Project, "Tor: anonymity online," 2009, [Online; accessed 3-July-2009]. [Online]. Available: http://www.torproject.org/

[92] Mozilla Foundation, "Firefox web browser — faster, more secure, & customizable," 2009, [Online; accessed 3-July-2009]. [Online]. Available: http://www.mozilla.com/en-US/firefox/ie.html

[93] The Tor Project, "Torbutton - quickly toggle firefox's use of the tor network," 2009, [Online; accessed 3-July-2009]. [Online]. Available: http://www.torproject.org/torbutton/index.html.en

[94] Wikipedia, "Log-normal distribution — wikipedia, the free encyclopedia," 2009, [Online; accessed 18-May-2009]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Log-normal_distribution&oldid=290409533

[95] A. Hallam, "Simple multivariate optimization," 2005, [Online; accessed 25-May-2009]. [Online]. Available: http://www.econ.iastate.edu/classes/econ500/hallam/documents/Opt_Simple_Multi_000.pdf

[96] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Characterizing user sessions on youtube," in *Fifteenth Annual Multimedia Computing and Networking Conference (MMCN)*, 2008.

[97] The Nielsen Company, "A2/m2 three screen report - 1st quarter 2009," 2009, [Online; accessed 3-June-2009]. [Online]. Available: http://it.nielsen.com/site/documents/A2M2_3Screens_1Q09_FINAL.pdf

[98] B. Cohen, "The bittorrent protocol specification," 2008, [Online; accessed 3-July-2009]. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html

[99] TheoryOrg, "Bitorrent protocol specification v1.0," 2006, [Online; accessed 3-July-2009]. [Online]. Available: http://wiki.theory.org/BitTorrentSpecification

[100] Ubuntu, "Ubuntu 8.0.4.2 lts (hardy heron)," 2009, [Online; accessed 20-June-2009]. [Online]. Available: http://releases.ubuntu.com/8.04/