An Efficient Video-On-Demand System

Thesis proposal

Eli Brosh
Department of Computer Science
Columbia University
elibroshe@cs.columbia.edu

December 27, 2008
Abstract

Video-on-demand (VoD) is an attractive service that has already gained popularity in the Internet by allowing users to view a video from a catalog of popular choices at any time. A limitation of existing VoD services, both using traditional design as well as peer-to-peer (P2P), is their inability to support efficient streamed access to all content at high quality, regardless of demand: not just the blockbusters that millions want to see, but also niche content that has relatively small demand.

This proposal’s main focus is to explore how to extend the design of successful P2P VoD systems to enable support for efficient delivery of a large library of high (movie) quality stored VoD. The goal is that the long tail of unpopular videos should be playable by any client without interruption within seconds of its request. We seek to achieve this objective by applying three design principles: leverage P2P technologies, altruistic cross-content caching and serving, and playback-point weighted caching strategies. We use mathematical analysis and simulation to provide guidelines for how to develop such a system. We provide several areas of future work such as developing on-line adaptive distributed caching algorithms.

In addition, this thesis seeks to study the performance of TCP for media applications. TCP has traditionally been considered inappropriate for media applications. Nonetheless, commercial VoD and streaming services as well as Voice Over IP applications use TCP since UDP packets cannot pass through many NATs and firewalls. Motivated by this observation, we study the delay performance of TCP for real-time media flows. We develop an analytical performance model for the delay of TCP. We use extensive experiments to validate the model and to evaluate the impact of various TCP mechanisms on its delay performance. Based on our results, we derive the working region for media applications and provide guidelines for delay-friendly TCP settings.
Contents

1 Introduction 1
  1.1 Problem Statement ................................................. 1
  1.2 Proposal Structure ............................................. 3

2 Related work 3
  2.1 Current VoD Systems ............................................. 3
  2.2 Leveraging P2P .................................................... 4
  2.3 TCP’s Analysis for Media Applications ......................... 5

3 Envisioned VoD System 5
  3.1 A Simple BitTorrent Extension ................................. 6
  3.2 General Model ..................................................... 7
    3.2.1 General Formalization ...................................... 7
    3.2.2 Optimization Objectives .................................... 8
    3.2.3 Global Optimization Objective ............................ 8
    3.2.4 Assigning pieces to peers’ ACC ............................ 9
  3.3 Research Plan ..................................................... 9
    3.3.1 Models and Experiments .................................... 10
  3.4 Phase 1: single client, single video .......................... 10
    3.4.1 Modeling ..................................................... 10
    3.4.2 Deriving a good cache distribution ....................... 12
    3.4.3 Remaining work within Phase 1 ........................... 13
  3.5 Phase 2: multiple clients, single video ....................... 14
    3.5.1 Distribution of video pieces within the ACC ............ 14
    3.5.2 Choosing requests to serve ................................ 15
    3.5.3 Earliest-First/Rarest-First Hybrids ...................... 15
  3.6 Phase 3: multiple clients, multiple videos .................... 16

4 TCP Performance for Media Applications 16
  4.1 TCP Delay Components .......................................... 17
  4.2 Modeling TCP Delay .............................................. 18
    4.2.1 A TCP Model ................................................. 18
    4.2.2 A Delay Performance Model .............................. 20
    4.2.3 Backlog and Window Size Evolution ....................... 22
  4.3 Model Validation ................................................. 22
  4.4 Results .......................................................... 24

5 Research Plan Summary 25
1 Introduction

1.1 Problem Statement

Video-on-demand (VoD) is an attractive service that has already gained popularity in the Internet [18] by allowing users to view a video from a catalog of popular choices at any time. However, current VoD design requires a large amount of costly server resources and significant bandwidth to support its users. The peer-to-peer (P2P) approach has been proven to be an effective solution for scalable content distribution without imposing a significant burden on a centralized infrastructure [30, 9]. In a P2P-based VoD system, users receive streamed videos from VoD servers as well as from the peers. The ability of peers that are viewing the videos to collaborate with each other reduces the load on serving infrastructure.

A limitation of existing VoD services, both using traditional design as well as P2P is their inability to support efficient streamed access to all content at high quality, regardless of demand: not just the blockbusters that millions want to see, but also niche content that has relatively small demand in comparison to their highly popular counterparts: re-runs of TV shows, non-blockbuster releases, replays of sporting events, speeches, and so on. Since the P2P model’s effectiveness is a function of the number of peers sharing content [18], P2P design benefits the service of popular videos. Since these videos are requested by many peers, many copies of them are available in the system. While the ability to serve a single niche video is insignificant from both a resource-consumption and revenue-producing perspective, there is ample evidence [7] that that there is a “long tail” of low-demand items, so that not serving them translates to lost revenues, and serving them using conventional client-server and P2P mechanisms is inefficient and costly.

This proposal’s main focus is to explore how to extend the design successful P2P VoD systems to enable support for efficient delivery of a large library of high (movie) quality stored VoD. The goal is that regardless of a video’s popularity, a high-quality version should be playable by any client attached to the network without interruption within seconds of its request. This high-quality experience is provided by current VoD systems, but only for a small subset of popular videos at any given time, and the broader experience is provided by services such as YouTube, but at significantly reduced quality.

We believe that this objective is achievable by applying three design principles:

- **Leverage P2P technologies**: Our proposed system follows the recent trend of incorporating P2P technology to help deliver video, i.e., the clients (viewers of video) also act as servers for their counterparts. It has been shown that such technologies scale remarkably well to the exchange of popular content.

- **Altruistic cross-content caching and serving**: Unlike traditional P2P systems in which clients are greedy and participate mainly in transfer of content that is of immediate interest to themselves, our system requires clients to contribute a portion of their resources altruistically, strictly for the purpose of serving others. The large shared demand for popular content is naturally handled by the greedy component, while the proportionally smaller demand for niche content is handled by the altruistic component. We leverage off of existing VoD frameworks where client participation in such a system would be less challenging.

- **Playback-point weighted caching strategies**: Earlier portions of videos must be retrieved shortly after the request for the video, whereas later portions can tolerate larger delivery delays. We
reduce the expected delay of retrieving earlier portions by forcing a higher replication rate of the earlier parts of the video than the later parts in the altruistic caching component.

Our research is guided by a particular system design, which merges properties of VoD-like systems with P2P mechanisms. While there is significant literature exploring this merge, our work has the following novel features:

• our emphasis is on including support for the long-tail of unpopular videos.

• we explore novel weighted, altruistic caching strategies in a P2P framework specific to efficient delivery of niche content.

Our research consists of a combination of mathematical analysis and simulation. Analysis is used to develop theory that provides a grounded guideline for how to develop such a system. This analysis coupled with simulation allows us to explore a wide range of the design space. More sophisticated simulation will demonstrate proof-of-concept and enable refinements to the analysis.

The broad overarching question we seek to answer is: How effective can a system be at providing an enjoyable streamed, stored video experience as a function of available global cache, upload and download rates, and initial startup delay? More specifically, the problems we explore are:

• **Cache allocation**: How much global cache (across P2P participants) should be allocated to a particular video as a function the video’s popularity? How should this cache then be weighted toward the various portions of the video, given that earlier portions must be retrieved with lower delay than their later portion counterparts? How should a video be dispersed within the global, distributed cache allocated to it? Should it be chopped into small pieces and spread among numerous peers, or should it be concentrated in larger pieces among a relatively smaller set of peers? How can these caching strategies be implemented dynamically?

• **Video Distribution**: How should peers share their time and cache between the video they are actively viewing and downloading, and videos that are cached altruistically on the behalf of their counterparts?

In addition, this thesis seeks to investigate the performance of TCP for media applications. The conventional wisdom is that TCP may be inappropriate for media applications because its congestion controlled reliable delivery may lead to excessive end-to-end delays that violate the real-time requirements of these applications. This has led to the design of alternative unreliable transport protocols [51, 28, 36] that favor timely data delivery over reliability while still providing mechanisms for congestion control.

Despite the perceived shortcomings of TCP, it has been reported that majority of commercial streaming traffic is carried over TCP [27]. Existing VoD services, even those based on s peer-to-peer design, deliver some percentage of the VoD content as unicast traffic to serve real-time customer requests [17, 21]. These commercial VoD services, as well as popular media applications such as Skype [10] and Windows Media Services [27], use TCP due to the wide-deployment of NATs and firewalls that often prevent UDP traffic. Furthermore, TCP is a mature and widely-tested protocol whose performance can be fine tuned.

The gap between the perceived shortcomings of TCP and its wide adoption in real-world implementations motivated us to investigate the delay performance of TCP. The questions we seek to answer are: (1) Under what conditions can TCP satisfy the delay requirements of media applications? (2) Why does TCP work for real-time media transmission? We address these questions
in the context of two real-time media applications that are characterized by timely and continuous data delivery: live video streaming\(^1\) and Voice over IP (VoIP).

To understand a broad set of aspects of the performance of real-time media applications, we conduct an extensive performance study using both an analytical model and real-world experiments. The analytical model allows us to systematically explore the delay performance over a wide range of parameter settings, a challenging process when relying on experimentation alone. While there exists an extensive literature on TCP modeling and analysis, it is geared towards file transfers [47, 46, 16] and video streaming [57, 35] rather than delay. We use both test-bed and Internet experiments to validate the model over a wide range of network environments. We analyze how the delay depends on the congestion control and reliable delivery mechanisms of TCP. We further study the impact of recent extensions such as window validation [29] and limited transmit [5]. The results obtained yield guidelines for delay-friendly TCP settings and may further be used to compare the performance of TCP with alternative protocols [28, 36] and experimental real-time enhancements for TCP [26, 42, 39].

1.2 Proposal Structure

The organization of the proposal proceeds as follows. In Section 2, we study prior work on VoD system design and the performance of TCP. Section 3 describes our envisioned VoD framework. Section 3.2 presents a formalization of the VoD system. Section 3.3 presents our research plan for our VoD system. We describe our current results and future work in Sections 3.4–3.6.

We then move to describe our proposal for studying the performance of TCP for media applications (Section 4). We describe the media application setting in Section 4.1, a performance model and its validation in Sections 4.2 and 4.3, respectively, and our key finding and results in Section 4.4. Finally, Section 5 outlines the research plan for completing this thesis.

2 Related work

VoD system designs and media streaming mechanisms have been independently studied to provide efficient streaming of media content. We first describe the previous studies on VoD system designs upon which our proposed VoD architecture and mechanisms are built (Section 2.1 and 2.2). Then, we review the studies related to the performance of TCP for media applications (Section 2.3).

2.1 Current VoD Systems

VoD systems of today can be divided into two basic classes:

- **High-quality, limited content:** Cable and telco providers have a pervasive infrastructure in place and a large user base [4], which is supported via proxy caching [19, 4] or content distribution networks [50] to offload the delivery of streaming video content onto (geographically) distributed

\(^1\)Note that in many commercial VoD systems, the originating VoD server limits the rate at which a stored video is being sent. This allows the system to reserve bandwidth for video streams and to provide quality of service to its users. Even though the videos have already been generated and stored at the server, it can be viewed as if the server is generating the content in real-time, i.e., as live video streaming, in the sense that the transmission is constrained by the sending rate of the video server rather than by the achievable network throughput.
servers, and then locally unicasting that content [17]. The bound on such local storage limits the amount of high quality content that can be simultaneously served.

- **Low-quality, diverse content**: Reducing the quality (size and file and streamed bandwidth) of what is forwarded enables systems, like YouTube [63], to support a wide body of smaller, lower-bandwidth videos, without having to implement caching near the endpoints.

Neither of these platforms can support high-quality delivery of a diverse body of content that our research is designed to address.

### 2.2 Leveraging P2P

**File-sharing**: Initial P2P systems [34, 23, 25] were designed to transfer files as an atomic unit, such that there was no advantage to delivering a file in-order. This attribute was utilized by the widely successful and popular BitTorrent [14], which utilizes swarms of peers interested in a common file, employing an elegant *tit-for-tat* strategy: a file is a priori divided up into pieces and peers exchange pieces (out of order), leading to a very efficient exchange and eventual delivery to all participating peers.

**Live streaming**: Subsequent work has demonstrated that BitTorrent-like mechanisms are also efficient at distributing live, streamed media [59, 64, 2]. Because transmission is live, at all times the entire swarm focuses exchange on a very narrow band of video pieces.

**Stored streaming (our focus)**: Recent work has explored applicability of BitTorrent-like mechanisms for (popular) stored, streamed media. The novel challenge is that earlier pieces of the video are needed sooner, so simple tit-for-tat trading across all pieces is less effective. Most recently, there have been several attempts to provide large-scale P2P-based VoD services over the Internet, such as Joost. The user quality of experience in these systems are still not comparable to that of traditional TV service provided by cable and telco providers. In particular, the video streaming quality is poor when the number of users watching a video is small [38]. Various hybrid policies [32, 48] have been shown to be effective at quickly distributing video while keeping individual peers’ distributions sufficiently different [67] in order to enable almost seamless playback.

To the best of our knowledge only few attempts have been made so far by the research community to investigate the possibility of P2P streaming of stored media. [33] designed a P2P architecture for set-top boxes based on an IP network including admission control and locality aware content fetching. However, the paper does not mention simulation experiments or quantitative results. [4] proposed to use the storage space of set-top boxes of a P2P VoD system as a cache. It uses real measurements to evaluate whether a currently deployed cable network can support a VoD service that scales with user population or catalog size. However, the evaluation is based on a single data trace. [53] proposed a push-to-peer scheme where the primary copies of the video catalog are pushed on set-top boxes that are used for video on demand. However, the scalability of the system is not of concern. The work in [13] analyzed the conditions for catalog scalability under a push-to-peer scheme. It focuses on the problem of how many distinct videos can be stored and served in a homogenous push-based system.

A handful of commercial box-based VoD solutions have been introduced recently. One example is NDS’s solution [45], which adds P2P and distributed DVR functionality to its IPTV middleman. Another is a prototype DVR system developed by BBC in cooperation with Promise Ltd that embeds a Bittorrent-like peer-to-peer file sharing client for content distribution [11]. Two
others are Vudu’s IPTV platform [55] and Voddler’s video service [56]. These solutions rely on
proprietary file download or streaming peer-to-peer technologies.

2.3 TCP’s Analysis for Media Applications

There is an extensive literature on analytical and experimental evaluation of TCP. We present only
those studies closely related to ours and refer the reader to [46] for a comprehensive survey of
TCP modeling. The majority of TCP modeling studies are geared towards file transfers assuming
either persistent [47] or short-lived flows [16]. Our work differs from past work in that we consider
non-greedy rate-limited flows with real-time delivery constraints.

More recently, the performance of TCP-based video streaming has been analytically analyzed
by [57]. The receive buffer size requirement for TCP streaming has been determined in [35].
These papers combine TCP throughput and application-layer buffering models to compute the
portion of late packets, whereas we directly model the transport-layer delay of TCP. Our work
further differs from those above in that we consider applications with tight delay constraints such
as video conferencing and VoIP.

Goel et al. [26] present an empirical study of kernel-level TCP enhancements to reduce the
delays induced by congestion-control for streaming flows. The performance of TCP for real-time
flows has also been considered by [42, 39]. However, unlike our study, these papers propose
a modification to the TCP stack. Application-layer heuristics for improving the loss recovery
latency of TCP have been suggested [41]. These heuristics are geared towards bursty traffic flows
and hence may not be effective for real-time flows.

3 Envisioned VoD System

While the mechanism we intend to employ to deliver streamed video to clients resembles BitTor-
rent, we assume that the setting in which the VoD architecture is to be deployed has behaviors and
user requirements that more closely resemble a traditional VoD architecture than that of generic
P2P file exchange. In this regard, each client’s set-top box implements the VoD functionality re-
mains owned and under the control of the provider (e.g., like a cable set-top box or TiVo box [3]).
This control enables the provider to implement distributed functionality, improving global good,
rather than just benefitting the local peer.

- **Client behavior**: clients, whose reason for participating in the system is to be able to watch videos,
spend only a portion of their time actively watching videos. However, even when not watching a
video, their equipment remains powered and connected to the network. This connectivity can be
taken advantage of to provide storage and forwarding within a P2P setting.

- **Provider behavior**: The provider has access to a limitless number of videos (using conventional
non-P2P technology), but without P2P support lacks the resources to provide on-demand access
for any video at any time. Because the peer’s boxes remain owned and under the control of the
provider, the provider has the ability to control/coordinate various algorithms run by the peers, as
well as the content that is stored in parts of the peers’ caches.

**Mechanism**: Figure 1(a) presents a high-level view of our vision of the overall architecture
that puts a P2P mechanism into an environment that remains controlled by the content provider.
Solid arrows indicate exchange of actual video information and dashed arrows represent exchange of control information. A large video repository chops each video into pieces, and these pieces are distributed by dedicated servers across peer nodes, who are then able to exchange these videos on demand (the servers can also help with the on-demand exchange when necessary). A transfer tracker has functionality similar to what exists in BitTorrent’s tracker - helping peers locate other peers who have cached the content which they seek [31].

Because the infrastructure remains under the control of the content provider, an additional cache tracker is employed whose job is to monitor the state of the global cache and push as well as redistribute the various video pieces across the peers and servers. Our research focus is the development of the algorithms implemented by the cache tracker; we believe that our lack of understanding of how to do the caching is the largest hurdle in enabling the ability to serve All Videos on Demand.

Figure 1(b) depicts the architecture of a peer, focusing specifically on the peer’s cache. The cache is split into two parts:

- A part, called the personal cache component (PCC), that remains under complete control of a user to store his/her recently watched videos or favorites, like part of a TiVo system.
- A second part (which is basically kept hidden from the user), which we call the altruistic cache component (ACC), is seeded with video pieces according to the cache tracker’s instructions on what to store. The inclusion of an AC deviates from traditional VoD architecture.

3.1 A Simple BitTorrent Extension

Figure 2: Traditional and modified choke
We make a simple modification to BitTorrent-like protocols so that they participate not only in exchanging content in their PCC, but also altruistically serve users whose requests are also serveable from the ACC. Figure 2(a) depicts a traditional BitTorrent connection [49]. Traditionally, a peer who is downloading a file (or video) \( V_1 \) directs all of its resources toward this swarm [9, 22, 62]. It connects (unchokes) to several other “good” peers with whom it has established a good exchanging relationship. Posing a limit on the number of concurrent peers being served ensures an acceptable startup delay and low protocol overhead [13]. To continually search for “good” peers, it randomly selects a potential peer with whom it has not yet established a good relationship and engages in a transfer. Such behavior is selfish in that each peer is constantly seeking out new, better pairings, but it also serves an altruistic purpose in that it gives peers an opportunity to bootstrap, or to receive parts of the file even when not offering anything in return: the receiving rate is significantly reduced.

Figure 2(b) depicts our extension: add one additional upload connection, but where the additional connection is for an alternate video \( V_2 \) that is not the focus of the current swarm, but is within the (ACC or PCC) cache of the uploading peer.

- This additional connection is very useful to peers who are interested in niche video where the size of the swarm is too small to sustain delivery.
- We hypothesize (and expect our research to prove) that the intrusion of this additional connection into a swarm is minimal and should not significantly affect the performance of a popular, active swarm.

The intuition behind the large gain to niche content with small cost to popular content is that popular swarms have more than enough capacity that a small percentage can be sacrificed without noticeable effect, while niche content’s transfer is improved dramatically with the little bit of extra help.

3.2 General Model

We begin by presenting a formalization of our system that will enable us to formalize our design objectives, as well as placement strategies.

3.2.1 General Formalization

There is a set of videos \( V \) where videos are enumerated \( 1, \ldots, |V| \), where video \( j \in V \) is requested at a rate of \( \lambda_j \), with larger \( \lambda_j \) indicating a more popular video. Videos are chopped into pieces (pieces of videos are assumed to be equal-sized unless explicitly stated otherwise). There are \( N \) peers in our system which we simply enumerate \( 1 \) through \( N \). Peer \( i \) provides an ACC of size \( C_i \) that can be used by the system to store videos, such that the aggregate ACC has size \( \sum_{i=1}^{N} C_i \). Peer \( i \) has an (average) upload rate of \( U_i \) and download at rate \( D_i \). Unless otherwise specified, we generally assume a homogeneous model with \( C_i, U_i \) and \( D_i \) respectively equalling fixed values \( C, U \) and \( D \).

Also, unless otherwise specified, we assume the video \( v \in V \) has a constant playback rate of \( r_v \), and when all pieces of the video are the same size, \( r_v \) can be measured in pieces per second.

In P2P systems, what generally constrains piece delivery is the upload capacity, \( U_i \), of a serving peer. This limit is often an artifact of the last-mile technology (e.g., cable, DSL), but can also be
imposed by the peer so as to keep bandwidth available for other tasks. We expect most of our evaluations to make the upload capacity the limiting factor.

Our general objective is to obtain seamless playback once the video starts playing. Often the user is willing to tolerate a short pre-buffer time (presumably on the order of seconds), which can vastly improve the remaining viewing experience. We let $d$ represent the amount of time between the request of the video by a peer to the time that playback should commence. A small $d$ is of course preferred.

Often, we will use time 0 as the time of a request for a video, so that $d$ is the time the video commences playback, and to achieve perfect, seamless playback, piece $i$ must be in the peer’s buffer by time $d + ir_v$.

### 3.2.2 Optimization Objectives

Since we do not expect to always have perfect, seamless playback, we must measure the performance of a video download. There are two basic measures, based on two different methods of video playback:

- **Skip-missing-piece**: playback skips over any piece not available at the appropriate point in playback (the video blacks out). If $X_i$ is an indicator r.v. that indicates when piece $i$ was not received in time for playout, then a simple objective is $\min E[\sum X_i]$. Note that this objective can be maximized by protecting delivery of later pieces (that have more time for receipt) at the expense of earlier pieces. An alternate objective, $\max X_i \min E[X_i]$, measures quality by the loss rate of the highest-lost piece. A strategy that equalizes all piece loss is optimal, and we conjecture (and show preliminary evidence here) that often this equalization strategy will serve both objectives simultaneously.

- **Waitfor-missing-piece**: when a piece is missing, playback stops until the piece is available. If $T_i$ is the time that piece $i$ is then played, a simple objective is minimize playback completion time: $\min E[T_{v-1}]$, where the $v$ pieces for the video are numbered $0, \cdots, v-1$.

While Waitfor-missing-piece is probably the more practical metric (preference is generally to pause rather than miss part of a stored video), Skip-missing-piece is often more tractable to analysis because of the reduced dependence of a piece’s outcome on previous piece outcomes. We conjecture that strategies that target optimizing Skip-missing-piece will often work well for Waitfor-missing-piece, and expect to demonstrate this conjecture through our research.

Note that these simple objectives do not take into consideration the burstiness of the intrusion, i.e., it can be argued that it is less frustrating to have on large skip or pause whose time (in pause or outage) equals the aggregate time lost due to several smaller skips or outages. We can extend the above measures for these cases. For instance, we could let $Y_j$ and $\tau_j$ respectively be the size (length in time, number of pieces) contiguously paused/missed, and minimize $\sum \log Y_j$ or $\sum \log \tau_j$, so that the measured penalty from a particular outage has the law of diminishing returns. While our final research may consider these more refined objectives, our preliminary formulations consider only the simpler objectives.

### 3.2.3 Global Optimization Objective

Let $M_j$ be the measure from above (i.e., $\sum X_i$ or $T_j$) that we wish to minimize for the $j$th request of a video within the system. When considering all videos, we can specify our global objective in
two ways:

- Minimize $E[\lambda_i M_j]$: minimize the average of the metric across all videos, where the weight of the video is proportional to its popularity. Note that the heavy tail of demand for niche videos means that their performance cannot be ignored. However, this metric biases better service toward popular videos at the expense of niche ones.
- Minimize $\sum_j E[M_j]$: treat all videos equally; optimal strategies are unbiased by the popularity of the video.

### 3.2.4 Assigning pieces to peers’ ACC

A major focus of our research is to determine the right strategy of assigning pieces of videos within peers’ ACC caches. To assist in formalizing our strategy we introduce the following notation:

- $R(i, v)$ is the number of (global) replicas of piece $i$ of video $v$. $0 \leq R(i, v) \leq N$ where $N$ is the number of peers in the network.
- $\phi(i, v, x)$ is the peer assignment of the $x$th copy of the $i$th piece of video $v$, $0 \leq x \leq R(i, v)$. Note that $\phi(i, v, x) \neq \phi(i, v, y)$ for $x \neq y$, i.e., we never assign to replicas of the same video piece to the same peer.\(^2\)

A cache distribution for a video $v$ refers to the values $\{R(i, v)/\sum_j R(j, v)\}$ over all pieces $i$. For instance, a uniform distribution would be one in which $R(i, v) = R(j, v)$ for all pieces $i \neq j$ in $v$.

A piece grouping strategy refers to the relative placement of various pieces of a video across peers’ ACCs. For example, one strategy is to minimize the number of ACCs used to support a video by grouping all pieces of a (partial) replica within a single ACC ($\phi(i, v, x) = \phi(j, v, x)$), or stripe pieces ($\phi(i, v, x) = \phi(i \mod S, v, x)$, $\phi(i, v, x) \neq \phi(j, v, x)$ when $j \neq i \mod S$ where $S$ is the number of stripes).

### 3.3 Research Plan

Our research plan’s main thrust asks: “What is the design of the cache tracker’s algorithm: where does it put the various pieces of videos, and how many copies of each, based on playback position of a piece and its video popularity?” To facilitate exposition of the high-level objectives of the proposal, henceforth we will consider a simplified homogeneous model.

- **Phase 1**: we analyze a single client accessing a single video. The availability of other peers to serve this video is represented using a straightforward stochastic availability process. Here, we focus on the intra-video piece distribution: how to distribute the pieces of a movie as a function of this availability process to maximize the viewing objective measures.

- **Phase 2**: we analyze multiple clients all accessing a single video, again using the simple availability process model. Here, we focus on how the effect of a swarm’s ability to self-serve as a function of swarm size. Since increasing swarm size enables more efficient transfer of early movie pieces from other peers’ PCC rather than ACC), we explore how this phenomenon affects storage of pieces within the ACC.

\(^2\)There is no benefit to having two identical replicas in a single peer’s cache, since a single copy can be transmitted to multiple peers simultaneously.
Phase 3: we analyze multiple clients accessing multiple videos such that the availability process model is more realistic. Here, we focus on the inter-video piece distribution, i.e., how the ACC cache should be shared across the various videos of varying popularity (i.e., swarm size).

As we move through the phases, we anticipate that modelling will take on less of a role, and experimentation takes on more of a role.

3.3.1 Models and Experiments

Our analysis will be a combination of modeling and simulation. Here, we give an overview of the various methods to be used:

- **Modeling**: We will develop stochastic mathematical models that facilitate a broad exploration of the design space, understand fundamental behaviors of our proposed protocols, and facilitate transition of our results to an educational component. In particular, modeling will identify beneficial caching policies. Currently, we have preliminary models for Phase 1, and sketches of models for Phase 2.

- **Round-based simulator**: A homegrown, discrete event simulator (already developed) where events occur in rounds. The round-based simulator’s main purpose is basic but efficient testing the efficacy of the proposed caching strategies across a wide set of scenarios. The round-based simulator has already been constructed to be amenable to all research phases 1, 2, and 3.

- **BitTorrent simulator**: Extensions to a simulator that emulates “Real BitTorrent” behavior [12]. This simulator relaxes the round-based assumption, and our extensions enable it to support stream-oriented upload and download strategies (placing more emphasis on earliest-first than rarest-first). This simulator is used to validate previously identified “good” caching strategies in more realistic environments. This simulator has been built to cover phases 1 and 2, but needs further work to extend it to Phase 3 evaluation.

3.4 Phase 1: single client, single video

We have already initiated our Phase 1 study using mathematical analysis and extensive round-based simulation, as well as some limited BitTorrent simulation. The single client assumption greatly simplifies the problem in that:

- all pieces are retrieved from peers’ ACC, and no pieces are retrieved via an active swarm.
- there is no benefit to a “rarest-first” policy, since the client downloading client has no peer to trade pieces with. Hence, for this phase, we can assume that an earliest-first policy is implemented.

3.4.1 Modeling

Our preliminary model makes the following assumptions:

- **Global vantage point**: the downloading client knows the cached locations of all pieces of interest.
• **Uploader availability:** A key challenge in modeling this system is emulating the behavior that the peers whose ACC contains the single video’s pieces are also participating in their own swarms of interest, and/or are busy serving other videos (i.e., their additional unchoke connection is serving an alternate request). Our initial approach is to capture this behavior using a simple availability model in which each round, when a peer is issued a request for upload of a piece in its ACC, it is available to serve that request in that round with a probability $P_a < 1$.

• **Piece transfer model:** within a round, the client can determine every uploader’s availability. It can then schedule for download the most needed pieces (closest to playback deadline) for simultaneous download within the round that fit under its download constraint and all uploading peer upload constraints.

  We model playback continuity of a downloader in a distribution swarm that has $N$ seeds (including VoD servers). The state-space of our model consists of $\{(i,k)|i = 0, \ldots, m−1; k = 0, \ldots, m + d − 1\}$ where $i$ is the piece index, $k$ is the playback position of the downloader and $d$ is the startup delay in rounds. The states $\{(.,k)|k = 0, \ldots, d−1\}$ correspond to the rounds a newly-arrived peer waits an buffers data before commencing playback.

  $p(i,k)$ denotes the probability that a downloader at round $k$ has successfully acquired piece $i$. We assume the method of playback is Skip-missing-piece, and define the playback continuity of piece $i$, denoted by $p_i$, as the probability the piece has been acquired before its deadline $i + d$. Hence, the playback continuity of piece $i$ is $p_i = p(i, i + d)$, and note that for $X_i$ used to define our metric skip-missing-piece, $P(X_i) = 1 − p_i$. We adapt a similar methodology as that in [67] to compute $p(i,k)$ by determining its steady-state behavior. We can express $p(i,k)$ as

  $$p(i,k) = p(i,k−1) + \begin{cases} q(i,k) & \text{if } k \leq i + d \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

  with $q(i,k)$ as the product of three components:

  $$q(i,k) = W(i,k)F(i)S(i,k), \quad W(i,k) = 1 − p(i,k−1), \quad F(i) = 1 − (1 − P_a)^fN \tag{2}$$

  $$S(i,k) = \prod_{j=max(0,k−d)}^{i−1} 1 − \frac{F(j)}{P_a N} (1 − p(j,k−1))$$

  where $f_i$ is the fraction of peers who cache piece $i$. $W(i,k)$ is the probability that the downloader does not have piece $i$ at round $k$, $F(i)$ is the probability that all peers with piece $i$ are unavailable, and $S(i,k)$ is the probability that piece $i$ has the earliest deadline. The downloader will not download piece $i$ from a seed if the seed has an earlier piece $j$, $j < i$ that the downloader wants and the seed has been selected as its provider. The probability that a downloader wants piece $j$ is $1 − p(j,k−1)$. Recall that in each round, only one seed is selected as a provider of a piece to prevent multiple seeds from simultaneously sending the same piece back to a downloader. The probability that the seed is selected as the provider for $j$ is taken to be the probability that there is at least one provider in the system normalized by the total number of suppliers $P_a N$. Hence, it is $F(j)/P_a N$. The expression for $S(i,k)$ in Eq. (2) is thus the probability that the seed has no piece $j$ that it prefers to serve instead of $i$.

  This formula simplifies to

  $$p_i = 1 − \prod_{k=0}^{i+d} (1 − F(i)S(i,k))$$

  where $F(i)$ and $S(i,k)$ are defined in Eq. (2). Intuitively, it can be understood as follows. The playback continuity is the probability a
piece is obtained in a sequence of at most \( i + d + 1 \) rounds, where the probability of obtaining piece \( i \) in the \( k \)th round is \( F(i)S(i,k) \). The term \( F(i) \) can be viewed as the probability to obtain piece \( i \) without interference from other pieces and \( S(i,k) \) as the interference level in the \( k \)th round.

### 3.4.2 Deriving a good cache distribution

Using our derivation of \( p_i \), we can derive closed-form bounds on the continuity for Skip-missing-piece. First, we have shown that when the allocation of pieces to cache is uniform \( R(i,v) = R(j,v) \) for all \( i,j \), then \( p_i \) is an increasing function of \( i \). This result justifies the intuitive reasoning that earlier pieces are more likely to miss their deadlines than later pieces when cache is uniformly allocated. Next, we can bound continuity using the functions previously defined:

**Theorem 3.1** The playback continuity function \( p_i \) is bounded by

\[
(1 - F(i))^{i+d+1} \leq 1 - p_i \leq (1 - F(i)S(i))^{i+d+1} \quad \text{where} \quad S(i) = \frac{1}{i+d+1} \sum_{k=0}^{i+d} \prod_{j=\max(0,k-d)}^{i-1} 1 - \frac{F(j)}{P_a N}
\]

Theorem 3.1 indicates that the playback continuity function has an upper bound whose shape is exponential with respect to the piece’s position in the video. Furthermore, numerical analysis shows that we can approximate the playback performance using the upper bound in Eq. (3.1), namely, \( p_i \approx 1 - (1 - P_a)^{Nf_i(i+d+1)} \).

An approximation to the optimal replica distribution function, where all pieces are equally likely to be retrieved, is obtained by setting all \( p_i = D \) for some constant \( D \) and solving for \( f_i \), yielding:

\[
f_i = \frac{\log(1-D)}{\log(1-P_a)N(i+d+1)} \quad \text{(3)}
\]

Noting that \( \sum_{i=0}^{m-1} f_i N = K \) where \( K \) is the size of the aggregate cache reserved this video in the ACC, we can thereby identify the appropriate value for \( D \).

Figure 3 depicts experimental results that show the benefit of a biased distribution. In these experiments, the upload rate per peer and the playback rate of the video is one piece per round, while the downloading peer can download five pieces per round. \( d = 0 \), and seed availability is \( P_a = 0.15 \).

The top graph in Figure 3(a) depicts how we distribute replicas of pieces within a 100-piece video when the allocated cache in the ACC is double the size of the video (200 pieces). For the uniform policy, there are two copies of each piece. As a second strategy, we implement a biased strategy that replicates pieces proportional to the rule provided by Equation 3, which reduces the number of replicas of pieces 41-100 to 1 to significantly increase the number of replicas of the first 10 pieces, and slight increases in the next 10.

The bottom graph of Figure 3(a) depicts results from a round-based simulation that evaluates the caching strategy. The playback continuity, plotted on the y-axis, is the likelihood that the piece is available during non-stop playout. We see significant gains in the continuity of the first few pieces, at a negligible drop in the pieces near the 40’s. Clearly, both measures used to evaluate skip-missing-piece (min-sum and max-min) are increased via the biased policy.

---

3Note this result is not trivial, as a piece is only requested when fewer than \( D \) pieces are both needed and available before it, where \( D \) is the maximum number of downloaded pieces in a round.
Intuitively, the benefit stems from the fact that the peers that can provide pieces are only available a fraction of the time. Figure 3(b) plots relative improvement of the front-weighted distribution in comparison to uniform distribution as a function of the availability of the seeds (top graph) and with the total cache size (bottom graph). The gain is most dramatic when seeds are rarely available (i.e., very active in their own swarm or serving other videos) and when the cache that can be allocated to a video is minimal.

3.4.3 Remaining work within Phase 1

There are a variety of open issues we wish to explore within Phase 1 that we have yet to commence:

- **Cache Sizing**: the larger we make the cache, the more a video can be globally replicated, the greater its availability within the system. Hence, given we know the right distribution, how large must the cache be for sufficient availability? Since our formula tells us the continuity rate of the video as a function of peer availability, $P_a$, we simply need to determine a desirable continuity rate, and given the availability rate, we can compute the cache size. This simple calculation of necessary cache size is useful in the later phases.

- **Extensions to Waitfor-missing-piece**: Skip-missing-piece is more amenable to modeling, so we have started there. However, we have also demonstrated through simulation that the same cache distribution provides significant gains for Waitfor-missing-piece, even though it is not optimized directly for that setting. Furthermore, our on-line algorithm that normalizes loss rates should automatically adjust itself within this alternative framework, fitting the cache distribution directly to observed loss rates.

- **Delayed startup**: Increasing tolerance for the initial startup delay $d$ lessens the demand for the immediate availability of the early pieces, and is expected to have a flattening effect on the distribution. We expect to explore this effect for reasonable values of $d$ (i.e., on the order of a few seconds).
• **2-value availability model**: peers belonging are in households actively watching a movie are likely to offer less availability to alternate swarms than peers belonging to households that effectively serve as a seed across all videos stored in the ACC. These two modes of operation can be represented in a homogeneous setting.

• **Striped vs. aggregate caching**: We return to the issue of piece grouping strategy ($\phi$). Consider the two extremes described in Section 3.2.4, where we assume there are enough peers so that each peer holds no more than one replica. For a fixed availability rate $P_a$, the wide striping is better. This is because the client can receive copies from peers in parallel, so multiple peers being available to serve simultaneously adds value. However, the availability rate $P_a$ may depend on how many different videos the peer is responsible for serving. By striping, the amount of cache in the ACC dedicated to the observed video reduces, and this reduction can be significant to cancel the above noticed gains from striping. We intend to identify the “cross-point”. In other words, what functional behavior must $P_a$ have with respect to the fraction of cache allocated to the video whereby the two caching strategies offer identical continuity rates?

### 3.5 Phase 2: multiple clients, single video

Phase 2 of the research explores the impact of the size of the swarm on the caching policy. Our interest here is in videos that have lower average demand, but at an “average” time, there is a high likelihood that more than one client is streaming the movie. As a result, there is increased reliance on the PCC buffer within the swarm to serve the video. With respect to Phase 1, our Phase 2 research addresses the following open issues:

#### 3.5.1 Distribution of video pieces within the ACC

![Figure 4: Growing swarm size](image)

Figure 4 shows how the number of replicas per piece in the cache affects the playback continuity for various numbers of clients downloading a video, $V$. We consider a single-video swarm with one or more client and 40 peers who are not actively downloading $V$, but have space in their ACC to store it. Each such peer provides availability of $P_a = 0.1$, and the aggregate cache (cross all seeds) is four times that of the video size. We assume that clients interested in $V$ arrive according to a poison process.

The x-axis shows the average number of clients in the system. The y-axis is the improvement in playback continuity of our playback-point weighted replication compared to that obtained using a naive uniform replication. The plotted results refer the relative improvement in playback continuity...
for the first 10% the video. We see that the distribution of the cache plays a less critical role as the size of the swarm grows. The swarm becomes “self-supporting”.

This phenomenon can be explained by the fact that peers only actively participate in a swarm while actively downloading the video. Once finished, the peer treats all requests for videos within its cache equally. Thus, peers actively participating in the swarm are at some midpoint in the movie playback, and this naturally creates a higher density of earlier pieces of the movie.

We can capture this phenomenon in our simple Boolean availability model by creating two more classes of peers in addition to the two classes of peers that have the cached content in their ACC, with one class (class A) active in a swarm and the other (class B) not. These new classes include:

- (class C) peers who have completed the download of the movie, having all pieces cached in their PCC: the distribution of pieces within the aggregate collection of class C peers caches is thus uniform.
- (class D) fellow peers who are in the process of downloading this stream, and hence aggressively prioritize this swarm, but with the distribution of pieces over the aggregate naturally weighted toward the front.

Class D peers would have a much higher availability than Class C and B, whose availability is higher than that of class A. While the distributions of the video within A and B are under our control, and C is known to be uniform, we must determine the distribution of the class C distribution before having an effective model.

3.5.2 Choosing requests to serve

During this phase, when a peer becomes available to serve, it may not have the resources to simultaneously serve multiple clients, especially when active in its own swarm. Hence, we need to consider a selection policy for how it prioritizes these incoming requests. A natural first candidate is minimum time to playout, i.e., the requestor with the minimum \( i - j \) where the request is for piece \( i \) and playout of that requestor is currently at \( j \).

However, minimum time to playout ignores the possibility that the requestor is at a point in the movie where it is rare for the minimum playout time to be so low. Really the right metric should be the probability of retrieving the requested piece in time. However, this measure depends more on estimates of retrieval rates of various pieces, and it remains to be seen if the additional work based on these estimates offers something significantly better.

3.5.3 Earliest-First/Rarest-First Hybrids

Recent related work in P2P stored, streamed media (e.g., [8, 21, 66]), as well as our own unpublished work [32] has demonstrated that having nodes served pieces in a strict, earliest-first (EF) fashion limits the diversity of pieces among peers and reduces the effectiveness of the BitTorrent swarm. On the other hand, a pure rarest-first (RF) policy makes the swarm effective, but often does not deliver the earlier pieces in time for playback. Instead, a variety of EF-RF hybrid schemes have been shown to provide significant gains.

Presumably, adjustments in the hybrid (e.g., the fraction of time an RF-like request is made versus an EF-like request) affect the desirable distribution of pieces. Furthermore, increasing startup
delay \( d \) permits more weighting of the mix toward the RF component. We expect to perform significant study during Phase 2 to understand this hybrid, coupled with startup delay, affects the optimal caching distribution. We predict that altering the cache distribution lessens the performance impact that results from varying the RF/EF mixture. Further investigation is nonetheless required to evaluate our prediction.

### 3.6 Phase 3: multiple clients, multiple videos

Phase 3 objectives focus mainly on inter-video issues. It consists mainly of implementing the policies and mechanisms derived during previous phases, and substantial validation via testing and simulation.

- **Validation of the availability model**: Phases 1 and 2 use a simple Boolean process to model availability of peers serving other videos. Phase 3 will explore the validity of such a fit via simulation experimentation across a large video suite.

- **Required sizing of the ACC**: After gauging (in Phase 2) the individual cache requirements of videos, here we explore how much aggregate cache is required across videos. Furthermore, if the aggregate cache is limited, how do we share the space across all the videos with different popularities?

### 4 TCP Performance for Media Applications

We study a general real-time media application, with a Constant Bit Rate (CBR) source, that sends data across the network using TCP. CBR is the most basic and dominant encoding for media flows in the Internet [58]. Although our analysis is general, we focus on CBR sources corresponding to VoIP and live video streaming. The VoIP and live video streaming flows are application-limited, i.e., their sending rate is a function of media encoding and not the underlying network. This is in contrast to greedy flows, such as FTP, which are network-limited.

We refer to the transmission unit of TCP as a segment and to the TCP payload (i.e., the application-layer data unit) as a packet. The maximum segment size, MSS, is determined by the maximum transmission unit of the network path [52]. A common characteristic of real-time applications is their sensitivity to end-to-end delay which may vary from application to application. For live video streaming, there is usually minimal interactivity involved, so the application can afford a startup delay in the order of seconds [27]. For VoIP, low delay of up to 400 ms is required in order to maintain acceptable interactivity [26]. To reduce end-to-end delays, VoIP often uses small payloads (e.g., 160-byte packets) that correspond to 20 ms or 30 ms of audio. Thus, in the context of this paper, the difference between VoIP and live video streaming flows is their packet sizes and their tolerance of delay.

We define TCP delay as the time it takes the application to get a packet from source to destination through a TCP connection. We use the TCP delay distribution to evaluate the performance of real-time applications. From the delay distribution we derive the application-level packet loss rate which is the portion of packets that arrive beyond their playback time. This metric is closely correlated with user-perceived video and audio quality [54, 20], and hence is used as an approximate performance measure. The application-level packet loss metric is determined by the \( \alpha \)-percentile
delay bound, defined as follows. A delay value \( d \) of \( \alpha \)-percentile corresponds to \( 1 - \alpha \) portion of packets that are delayed more than \( d \) time units.

### 4.1 TCP Delay Components

Here we examine the various ways in which delay is introduced in a TCP connection with a CBR source. The delay in a TCP connection consists of two main components: (a) network delay, which is the time it takes a segment to get across the network; (b) TCP-level delay, which is an artifact of how TCP reacts to variations in the effective throughput. While throughput variations can occur due to application-level flow control, they are primarily the result of network congestion. To understand TCP-level delays, we briefly describe the transmission behavior of TCP. TCP is a window-based protocol that uses two main mechanisms to regulate its sending rate: Additive-increase-multiplicative-decrease (AIMD) and timeout. These mechanisms may delay data delivery because they require TCP to reduce its sending rate in response to network congestion. In addition, TCP uses packet retransmissions to provide lossless data delivery. This mechanism introduces additional delay for data delivery. A detailed discussion of the mechanisms in TCP can be found in [52].

TCP uses two buffers to provide congestion-controlled reliable data delivery; a send buffer and a receive buffer. The send buffer serves two functions [26]. It absorbs rate mismatches between the application sending rate and the transmission rate of TCP. It also stores a copy of the packets in transit in the network should they be retransmitted. Although these packets are buffered, they do not introduce additional queuing delay for unsent packets. Only the unsent packets held in the send buffer, hereafter referred to as the backlogged packets, contribute to the delay of newly admitted packets to the send buffer. The purpose of the receive buffer is to hold out-of-order packets while a loss is being recovered. This buffering results in head-of-line (HOL) blocking delay.

We only consider packet backlogging due to network congestion and ignore packet backlogging due to other causes, such as application-level flow-control (e.g., a receiving application that slows down an aggressive sender [52]). Applications usually minimize this backlogging by setting a large receive buffer and operating with non-blocking sockets. Packet backlogging can also occur due to Nagle’s algorithm [44] that was added to TCP to limit the transmission of small segments. This algorithm ensures that TCP sends data only when there are at least MSS bytes of available data, and hence improves throughput at the expense of increased transmission delay. In practice, many delay-sensitive applications disable this algorithm to reduce transmission delays [65]. We follow this practice in our work. Figure 5 illustrates the TCP-level delay components. The sender-side delay is caused by the congestion control and reliable delivery mechanisms in TCP, whereas the receiver-side delay is caused by the in-order delivery guarantee of TCP.

Figure 6(a) illustrates the delay behavior of a TCP flow driven by a CBR source. The CBR
source sends 50 MSS-sized packets per second over a symmetric network with a 200 ms round-trip time (RTT). An application-limited period is seen from 0 s to 0.5 s and from 2.4 s to 2.8 s. We define an application-limited period as a period where the TCP throughput satisfies the source’s rate requirement. In this period, the TCP delay is determined by the network delay. A network-limited period is seen from 0.5 s to 2.4 s. In this period, the TCP throughput no longer satisfies the source’s rate requirement, resulting in TCP-level delays. TCP moves to a network-limited period when a packet loss occurs. Within the network-limited period there are two subregions: loss recovery, seen from 0.5 s to 0.76 s, and packet backlogging, seen from 0.76 s to 2.4 s. TCP uses retransmission to recover the lost packet, which in turn causes head-of-line blocking delay at the receiver. The receipt of a packet loss indication at time 0.76 s triggers TCP to reduce its congestion window size, resulting in packet backlogging.

Unlike application-limited periods, in network-limited periods TCP probes for additional bandwidth to satisfy the source’s rate requirement. In our example, the transmission rate of TCP is governed by the AIMD mechanism and hence is linearly increasing, as seen in Figure 6(b). The mismatch between the input and output rates at the TCP sender results in the quadratic-like delay curve seen in Figure 6(a). TCP moves back to an application-limited period when the rates are matched.

4.2 Modeling TCP Delay

Our model builds upon the detailed TCP model in [60] that predicts the performance of TCP from the viewpoint of throughput. We extend this model in three ways. First, we include the TCP buffer dynamics in order to predict the delay performance of TCP. Second, we model the window behavior during application-limited periods [29] to accurately capture the loss recovery latency of TCP. Third, we capture the effect of window inflation [6] and the limited transmit mechanism [5] to improve the accuracy of the model for small congestion windows. We assume that the sender is using a NewReno TCP implementation, the predominant TCP variant in the Internet [40], and refer the interested reader to [24, 52] for a detailed description of TCP NewReno’s mechanisms.

4.2.1 A TCP Model

We consider a CBR source that sends fixed-size packets at regular intervals across the network using TCP. Throughout the paper, we assume that the average throughput provided by TCP satisfies
the rate requirement of the CBR source. However, transient congestion episodes in the network can still lead to TCP throughput fluctuations and hence to TCP-level delays. These episodes cause the TCP connection to alternate between application-limited and network-limited periods, as described in Section 4.1.

Mimicking the behavior of a real-world TCP flow, our model consists of two main states: application-limited and network-limited. It transitions from an application-limited state to a network-limited state when a loss occurs. TCP-level delays are introduced only during network-limited states. The model transitions back to an application-limited state when the TCP sender matches its input and output rates (e.g., when packet backlog is cleared). While in a network-limited state, the model moves among four states corresponding to TCP’s congestion control phases: slow-start (SS), congestion avoidance (CA), fast recovery (FR), and retransmission timeouts (TO). A high-level view of a model for a TCP connection with a CBR source is shown in Figure 7; for ease of presentation, we merged the timeout and fast recovery states into a single loss recovery (LR) state.

We make several simplifying assumptions in our model, as follows. First, we assume that TCP increases the congestion window by one packet per round-trip time, an assumption motivated by the wide-deployment of ACK-counting TCP implementations [40]. Second, we assume that the TCP implementation does not increase the congestion window when the TCP sender is application-limited, which is the behavior observed for Linux and Windows XP systems (see Section ??). Third, we assume that the slow start threshold is statically set to half of the source’s sending rate in packets per round-trip time. From our experience, using a static slow-start threshold rather than a dynamic one has a marginal impact on the model’s prediction accuracy. Last, we do not model the effect of delayed acknowledgements (ACKs). Nonetheless, our model can be easily extended to support delayed ACKs using a similar approach as in [47].

The CBR source is characterized by two parameters, the data generation rate in packets per second $f$ and the size of a generated packet $a$. We let $r$ denote the data generation rate in packets per round-trip time. For convenience, we summarized the notations used in this paper in Table 1. We model the behavior of a TCP source by a discrete-time Markov chain with a finite state space $S = \{(w, b, l)\}$ and a probability transition matrix $Q = [q_{ss'}], s, s' \in S$. Each state is associated with at most three outgoing transitions representing the following events: the receipt of a fast retransmit loss indication, the receipt of a timeout loss indication, and successful delivery of window data. Each transition is associated with a certain number of packet transmissions, and each packet in this transmission is associated with a delay.

In our model, each state is represented by an ordered triple $(w, b, l)$, where $w$ is the current congestion window size in segments, $b$ is the current backlog size in bytes, and $l$ indicates whether a loss has been detected and needs to be recovered from ($l > 0$) or not ($l = 0$). The backlog size value is used to indicate whether the sender is application-limited ($r \leq w, b = 0$) or network-limited pocket loss.
### Table 1: Summary of model notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>load in packets per second</td>
</tr>
<tr>
<td>(r)</td>
<td>load in packets per round-trip time</td>
</tr>
<tr>
<td>(a)</td>
<td>packet size (bytes)</td>
</tr>
<tr>
<td>(w)</td>
<td>congestion window size (in segments)</td>
</tr>
<tr>
<td>(b)</td>
<td>backlog size (bytes)</td>
</tr>
<tr>
<td>(l)</td>
<td>indicates whether loss recovery is required</td>
</tr>
<tr>
<td>(p)</td>
<td>segment loss probability</td>
</tr>
<tr>
<td>(L)</td>
<td>one-way network delay (seconds)</td>
</tr>
<tr>
<td>(MSS)</td>
<td>maximum segment size (bytes)</td>
</tr>
</tbody>
</table>

### Table 2: State classification

<table>
<thead>
<tr>
<th>Classification</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL (Application-limited)</td>
<td>(r \leq w, b = 0, l = 0)</td>
</tr>
<tr>
<td>NL (Network-limited)</td>
<td>(0 &lt; w, b \neq 0) or (w &lt; r, b = 0)</td>
</tr>
<tr>
<td>CA (Congestion avoidance)</td>
<td>(r/2 &lt; w, l = 0)</td>
</tr>
<tr>
<td>SS (Slow start)</td>
<td>(w \leq r/2, l = 0)</td>
</tr>
<tr>
<td>FR (Fast recovery)</td>
<td>(0 &lt; w, l = 1)</td>
</tr>
<tr>
<td>TO (Timeout)</td>
<td>(w = 0, l = 1, \ldots, 6)</td>
</tr>
</tbody>
</table>

Limited. The window size value is used to distinguish between the two loss recovery strategies employed by TCP: fast recovery \((w > 0, l = 1)\) and retransmission timeout \((w = 0, l \geq 1)\), where \(l\) indicates the current exponential back-off stage. Table 2 lists the rules for classifying an arbitrary state \(s = (w, b, l)\) according to the congestion control phases of TCP. We use the notation \(AL = \{(w, b, l) : r \leq w, b = 0, l = 0\}\) to denote the set of states for which the application-limited condition holds. The notations \(CA, SS, FR, TO\) are defined in a similar way, as shown in Table 2.

#### 4.2.2 A Delay Performance Model

In this section, we model the three ways in which TCP introduces delays: congestion control, retransmissions, and head-of-line blocking, as detailed in Section 4.1. We model the time packets are buffered at the sender (i.e., the congestion control delay) by scaling the backlog size by the source’s rate. This modeling approach can be explained by observing that the (unsent) buffered packets left behind after a packet is transmitted must have been admitted to the send buffer while the transmitted packet was buffered. Since we consider a data source with a constant rate, a transmitted packet that leaves behind a backlog of \(b\) bytes must have been buffered for at least \(b/(fa)\), the backlog size divided by the source’s rate in bytes per second. This approach introduces an error in the order of several packetization intervals. The error arises because the Markov chain captures the backlog size evolution in network-limited states at round-trip time granularity. This error can be reduced by keeping track of the inter-sending packet times. However, this will make the state space of the model prohibitively large and hence will limit its usefulness.

We determine the head-of-line and the retransmission delay by the loss recovery latency (i.e., the time it takes TCP to detect and recover a lost packet). TCP interprets receipt of three duplicate
ACKs as an indication of a packet loss. It immediately retransmits the lost packet upon the receipt of the third duplicate ACK. Hence, we take the time required to receive a fast retransmit loss indication to be $RTT + 3/f$; the $RTT$ term is the time needed for the first duplicate ACK feedback and the $3/f$ term is the maximum time to generate three duplicate ACKs, which is attained when the loss occurs in an application-limited state. For sake of simplicity, we assume that fast recovery always takes a single RTT regardless of the number of packets lost in a transmission window, as suggested by [16].

Using the above observations, we express the TCP delay of the $i^{th}$ packet sent in a transition from state $s$ to state $s'$ as:

$$d^{(i)}_{s,s'} = L + \begin{cases} 
  b/(fa) + RTT + (3+i)/f & \text{if } s' \in FR \\
  0 & \text{if } s' \in TO \\
  b/(fa) & \text{otherwise}
\end{cases}$$

(4)

where $L$ is the one-way sender to receiver network delay. For loss-free transitions, the delay added by TCP is determined by the backlogged packets, and hence is modeled as $b/(fa)$, as shown by the third case of (4). For transitions to fast recovery states, an additional delay of $RTT + (3+i)/f$ is introduced by the in-order delivery guarantee of TCP, as shown by the first case of (4). Since the TCP sender is likely to be idle during timeouts, we assume no packets are sent in transitions to timeout states.

The number of CBR packets sent in a transition from $s$ to $s'$ $n_{s,s'}$ is given by

$$n_{s,s'} = \begin{cases}
  1 & \text{if } s \in AL, s' \in AL \\
  \min (b, wMSS) / a & \text{if } s' \in \{CA|SS\} \\
  \min (b, (w+3)MSS) / a & \text{if } s' \in FR \\
  0 & \text{if } s' \in TO
\end{cases}$$

(5)

Since our model evolves at packet-level granularity while in an application-limited state (see Section 4.2.1), a single packet is sent in a loss-free transition from an application-limited state, as captured by the first case of (5). The second case models the number of packets sent in a loss-free transition from a network-limited state, which is determined by the number of backlogged packets that fit into the congestion window. The third case accounts for the extra transmissions due to the receipt of the duplicate ACKs needed to trigger a fast recovery, a mechanism known as window inflation [6].

We obtain the stationary distribution of the Markov chain for the TCP source, $\pi_s$, using standard steady-state discrete-time Markov analysis; see for example [61]. Let $N_t$ be the number of packets successfully sent in some time interval $[0,t]$ and let $N_t(d)$ be the number of packets out of $N_t$ that experience delay $d$. Then, the portion of packets sent that experience delay $d$ is given by $N_t(d)/N_t$. Let $D$ be the steady state delay distribution of a TCP connection with a CBR source. Assume $D$ is defined over some finite interval $A$. Using renewal theory [61], we can now compute the steady-state delay distribution.

$$D = d \text{ w.p. } \lim_{t \to \infty} \frac{N_t(d)}{N_t} \quad \forall d \in A$$

$$D = d \text{ w.p. } \frac{\sum_{s \in S} \pi_s \sum_{s' \in S} q_{s,s'} \sum_{i=1}^{n_{s,s'}} I_{d_{s,s'} = d}}{\sum_{s \in S} \pi_s \sum_{s' \in S} q_{s,s'} n_{s,s'}} \quad \forall d \in A$$

(6)
where $I$ is the indicator function, $\pi_s$ is the steady-state distribution of the chain, and $d_{s,s'}$ and $n_{s,s'}$ are given in (4) and (5), respectively. The numerator and denominator correspond, respectively, to the number of packets sent that experience delay $d$ in steady-state and the number of packets sent in steady-state. Equation (6) can be solved numerically to yield the performance statistics of TCP: the delay jitter $\sigma_D$ and the $\alpha$-delay percentile $\arg\max_x P\{D \leq x\} \leq \alpha$, along with other statistics such as the mean delay $E[D]$.

4.2.3 Backlog and Window Size Evolution

The discrete-time Markov model for the TCP source moves along several states, changing the congestion window size, send buffer size, and congestion control phase based on the packet loss feedback. For example, in the absence of packet loss, the TCP model transitions from state $(w,b,l)$ to state $(w+1,b',l')$ if the sender is in congestion avoidance.

Since TCP is a byte stream protocol, it can assemble a number of small application packets into one TCP segment. An application that uses small packets (e.g., VoIP) yields a TCP flow that dynamically varies its segment size, and hence the packet size on the wire, depending on the congestion in the network. During network-limited periods, the data backlog enables the TCP sender to use the maximum segment size. In application-limited periods, however, there is no backlog at the sender, and TCP matches the segment size to the application payload size. Let $M_s$ be the size of a segment transmitted in a transition from state $s$. Hence, $M_s = a$ if $s \in AL$ and $M_s = MSS$ otherwise.

The backlog evolution (i.e., the TCP send buffer occupancy evolution) for two successive states, $s = (w,b,l)$ and $s' = (w',b',l')$, is modeled by

$$b' = \begin{cases} 
\max (0, b + a f t_{s,s'} - M_s) & \text{if } s \in AL, s' \in AL \\
\max (0, b + a f t_{s,s'} - wM_s) & \text{if } s' \in \{CA, SS\} \\
\max (0, b + a f t_{s,s'} - (w+3)M_s) & \text{if } s' \in FR \\
\max (0, b + a f t_{s,s'}) & \text{if } s' \in TO 
\end{cases}$$

(7)

where $t_{s,s'}$ is the time taken for the transition from $s$ to $s'$, which can be found in [15]. The first term in (7) $b + a f t_{s,s'}$ models the increase in backlog size due to newly admitted packets to the send buffer. The second term models the decrease in backlog size due to the transmission of segments, which is obtained by applying similar reasoning to that used to derive (5).

4.3 Model Validation

We evaluated the model using experiments in a controlled network environment and Internet experiments using PlanetLab and residential machines. We wrote a tool that can send and receive bidirectional CBR over TCP flows with different packet sizes and different packetization (inter-sending time) intervals. To validate our model we use CBR sources with packet sizes of 174, 724 and 1448 bytes, and packetization intervals of 20 ms and 30 ms, as these choices approximately reflect typical one-way voice [51], low bit rate interactive video [26] and live video streaming [27]. The size of the packet includes a 12 byte RTP header [51] and two bytes for framing RTP packets over TCP [37]. Hence, excluding the header size, the bit rate of the voice flows is 64 kb/s and 42 kb/s, that of interactive video is 284 kb/s and 187 kb/s, and that of live video streaming is
We validated the model in a controlled network environment using a test-bed that emulates a wide range of network settings. The topology of the test-bed is shown in Figure 8. Figure 8(a) shows the topology for model validation using configured drop rates. Here we consider a single CBR-TCP flow going through a router running NIST Net [1], a network emulation program. We configured NIST Net to drop packets uniformly at random independent of their size. NIST Net was configured with drop rates of 0.1%, 0.5%, 1%, 2%, 3%, 5% and 10%, and a fixed round-trip propagation delay of 20 ms, 100 ms, and 300 ms. Figure 8(b) shows the topology for model validation using a drop-tail router. Here we consider a scenario where multiple CBR-TCP flows compete with FTP and web flows for a bottleneck router with a drop-tail queueing scheme. The round-trip propagation delay was set to 100 ms for all experiments. The link capacity was set to 3 Mb/s and 30 Mb/s for voice and video CBR-TCP flows, respectively.

For each set of parameters of the controlled network environment, we ran experiment for five minutes and repeated each experiment ten times. For example, Figure 9(a) and Figure 9(b) present the predicted vs. measured mean and 95th percentile TCP delay, respectively, for VoIP and video flows for various configured packet loss rates and RTT of 100 ms and 300 ms. In general, our results show that the model provides satisfactory matching to the measurement for the majority of cases. We study the modeling accuracy across various loss rates and RTTs by measuring the relative prediction error of the average TCP delay with respect to the actual measurement for

573 kb/s and 378 kb/s. Unless stated otherwise, we refer to the voice flow with a bit rate of 64 kb/s as ‘VoIP’ and the live video streaming flow with a bit rate of 573 kb/s as ‘video’ flow.
VoIP and video flows. This error was below 10% for the majority of cases, hence validating the applicability of our model.

We also performed model validation using the PlanetLab environment and hosts connected to residential DSL and cable modems. Our results show that there is a good match between the model and the measured delay over the real Internet.

4.4 Results

We use mathematical modeling and experimentation to explore the delay performance of real-time delivery over TCP. We summarize the key findings as follows:

- **Working region**

  We derive the working region for VoIP and live streaming flows based on our model and experiments. For interactive applications, ITU G.114 recommends that the worst-case one-way delay should be 400 ms. Studies show that 200 ms is an acceptable one-way delay limit for VoIP applications [43]. The choice of the delay limit for video is more flexible because people can usually tolerate a few seconds of startup delay. For the analysis we consider a 5 s startup delay, as suggested by [27]. While VoIP can usually tolerate up to 5% of packets that miss their playout deadline without a significant effect on intelligibility [43], video viewing quality drops rapidly at 0.1% [57]. We follow these guidelines and define the working region for VoIP and live video streaming as the range of network loss rates and RTTs where the 95th percentile and maximum TCP delay is at most 200 ms and 5 s, respectively.

  Figure 10(a) plots the 95th percentile delay for various loss rates from 0.1% to 10% and RTTs of 20 ms, 100 ms, and 300 ms for a VoIP flow with a bit-rate of 64 kb/s. Observe that when the RTT is 100 ms, the delay tolerance for VoIP is satisfied when the network loss rate is at most 2%. However, when the RTT is only 20 ms, the results indicate a tolerance of up to 5%. At the boundary of the working region, the delay added by TCP causes 5% of the packets to miss their playback deadline. Figure 10(b) plots the maximum delay for a live video streaming flow with a
bit-rate of 573 kb/s. When the RTT is 100 ms, the streaming threshold is satisfied when the loss rate is at most 3%. For RTT of 300 ms, it is satisfied at a network loss rate of 0.1%. The jump in the maximum delay at a network loss rate of 0.5% and RTT of 300 ms occurs because the 5 s startup delay is no longer sufficient to completely mask TCP delays. This knee of the curve typically occurs when the achievable TCP throughput is close to the bit rate of the video flow.

In summary, we find that under the same network conditions, VoIP flows suffer from lower TCP delays than live video streaming flows. VoIP operates well when the network loss rate is at most 2% and RTT is at most 100 ms. Live video streaming operates well when the network loss rate is at most 3% and RTT is 100 ms.

**Why does TCP work for real-time media transmission?**

Our research reveals that real-time application performance over TCP may not be as delay-unfriendly as is commonly believed. One reason is that the congestion control mechanism used by TCP regulates rate as a function of the number of packets sent by the application. Such a packet-based congestion control mechanism results in a significant performance bias in favor of flows with small packet sizes, such as VoIP. Second, due to implementation artifacts, the average congestion window size can overestimate the actual load of a rate-limited flow. This overestimation reduces the likelihood of timeouts and consequently the resulting TCP delay.

**Delay-friendly TCP settings**

We use our model and experimentation to study the impact of various TCP mechanisms on the TCP delay. Based on the results, we can provide a comprehensive set of guidelines for delay-friendly settings of TCP and OS parameters. While several settings such as disabling Nagle’s algorithm and using large receive buffers are common practices in delay-sensitive applications, the use of others, specifically, window validation, byte counting and limited transmit was not well understood (in the context media applications) prior to this study.

## 5 Research Plan Summary

This thesis proposes to use a combination of mathematical modeling, simulation, and experimentation to develop and study the performance of efficient streaming mechanisms for media content. Table 3 shows my tentative research plan for completing the research. A more detailed description of the future work is outlined in Sections 3.4, 3.5, and 3.6.
**Table 3: Plan for completion of my research**

<table>
<thead>
<tr>
<th>Timeline</th>
<th>Work</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Performance analysis of TCP for media applications</td>
<td>completed</td>
</tr>
<tr>
<td>Jan 2009</td>
<td>Single video, single swarm</td>
<td>ongoing</td>
</tr>
<tr>
<td>Jan-Feb 2009</td>
<td>Multiple videos, single swarm</td>
<td>ongoing</td>
</tr>
<tr>
<td>Mar-Apr 2009</td>
<td>Multiple videos, multiple swarms</td>
<td>ongoing</td>
</tr>
<tr>
<td>May 2009</td>
<td>Thesis writing</td>
<td>ongoing</td>
</tr>
<tr>
<td>June 2009</td>
<td>Thesis defense</td>
<td>XXX</td>
</tr>
<tr>
<td>July 2009</td>
<td>Thesis revision and deposit</td>
<td>XXX</td>
</tr>
</tbody>
</table>

**References**


Nds homepage. [http://www.youtube.org](http://www.youtube.org).


Voddler homepage. [www.voddler.com](http://www.voddler.com).

Vudu homepage. [www.vudu.com](http://www.vudu.com).


Youtube homepage. [http://www.youtube.org](http://www.youtube.org).

