

2013

# Optimizing on-demand resource deployment for peer-assisted content delivery

---

<https://hdl.handle.net/2144/13123>

*"Downloaded from OpenBU. Boston University's institutional repository."*

BOSTON UNIVERSITY  
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

**OPTIMIZING ON-DEMAND RESOURCE DEPLOYMENT FOR  
PEER-ASSISTED CONTENT DELIVERY**

by

**RAYMOND SWEHA**

B.S., Alexandria University, Egypt, 2005  
M.S., Boston University, 2009

Submitted in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy

2013

Approved by

First Reader

---

Azer Bestavros, PhD  
Professor of Computer Science

Second Reader

---

Abraham Matta, PhD  
Professor of Computer Science

Third Reader

---

Mark Crovella, PhD  
Professor of Computer Science

*The most difficult subjects can be explained to the most slow-witted man if he has not formed any idea of them already; but the simplest thing cannot be made clear to the most intelligent man if he is firmly persuaded that he knows already, without a shadow of doubt, what is laid before him.*

Leo Tolstoy in *The Kingdom of God is Within You*.

## **Acknowledgments**

First of all, I thank God for all the gifts He has given me. Second, I thank my parents who made me into the man I am today and Sherry, the love of my life, who motivated me every time I was discouraged. Third, I thank my advisors, Azer and Abraham, for their guidance and for being great role models. At last, I thank the rest of my family and friends for their continued support and understanding. Special thanks to Vatche Ishakian, my friend and co-author, for his relentless support and professionalism.



system is to maintain a given streaming quality. The third application is Tor, the anonymous onion routing network, in which the goal of the system is to boost performance (increase throughput and reduce latency) throughout the network, and especially for clients running bandwidth-intensive applications.

For each of the above applications, we develop analytical models that efficiently allocate the already available resources. They also efficiently allocate additional on-demand resources to achieve a desired level of service. Our analytical models and efficient constructions depend on some simplifying, yet impractical, assumptions. Thus, influenced by our models and constructions, we develop practical techniques that we incorporate into prototypical peer-assisted angel-enabled cloud services. We evaluate these techniques through simulation and/or implementation.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Thesis Premise . . . . .	2
1.3	Analytical Models . . . . .	4
1.4	Practical Content Delivery Strategies . . . . .	6
1.5	Experimental Evaluation . . . . .	7
1.6	Thesis Organization . . . . .	9
<b>2</b>	<b>Related Work</b>	<b>11</b>
<b>3</b>	<b>CloudAngels: A Peer-Assisted Bulk-Synchronous Content Distribution Service</b>	<b>16</b>
3.1	Motivation and Problem Statement . . . . .	16
3.2	Optimal MDT Construction . . . . .	19
3.2.1	MDT Problem Statement . . . . .	19
3.2.2	Main Results . . . . .	20
3.2.3	Implications on the Role of Angels . . . . .	26
3.3	Practical MDT Construction . . . . .	27
3.3.1	The Group Tree Coordinated Swarming Strategy . . . . .	28
3.3.2	Other Distribution Strategies . . . . .	31
3.4	Simulation Results . . . . .	32
3.5	<b>CloudAngels: Blueprint for System Design</b> . . . . .	35
3.6	<b>CloudAngels: Experimental Evaluation</b> . . . . .	39
3.7	Related Work . . . . .	42
3.8	Summary of Contributions . . . . .	45

<b>4</b>	<b>AngelCast: Peer-Assisted Live Streaming Using Optimized Multi-Tree Construction</b>	<b>46</b>
4.1	Motivation and Problem Statement . . . . .	47
4.2	Theoretical Bounds . . . . .	49
4.2.1	Optimal Angel Allocation . . . . .	50
4.2.2	Startup Delay . . . . .	52
4.2.3	Implications on the Role of Angels . . . . .	54
4.3	A Practical Construction . . . . .	56
4.3.1	Bounding the Angel Upstream Capacity . . . . .	56
4.3.2	Bounding the Startup Delay . . . . .	60
4.4	AngelCast Architecture . . . . .	61
4.4.1	Membership Management: The Registrar . . . . .	61
4.4.2	The AngelCast Protocol . . . . .	66
4.5	Experimental Evaluation . . . . .	68
4.6	Related Work . . . . .	74
4.7	Summary of Contributions . . . . .	78
<b>5</b>	<b>TorAssist: Enhancing Tor Performance For Bandwidth-Intensive Applications</b>	<b>79</b>
5.1	Introduction . . . . .	80
5.2	Background and Related Work . . . . .	82
5.3	A Measurement Study on Live Tor . . . . .	86
5.4	Tor Circuit Throughput: Analytical Model . . . . .	90
5.4.1	Variability of Tor Circuit Throughput . . . . .	91
5.4.2	Effect of “Best-Of-K” Circuit Selection . . . . .	91
5.4.3	Effect of Filtering out Probes with Below-Average Throughput on the Performance of the PID Controller . . . . .	93
5.5	Performance Evaluation . . . . .	94
5.5.1	The Simulator . . . . .	95
5.5.2	Simulation Results . . . . .	97
5.6	Bandwidth Boosting using Multi-Path Routing . . . . .	100

5.7	Summary of Contributions . . . . .	102
<b>6</b>	<b>Summary</b>	<b>104</b>
<b>7</b>	<b>Appendices</b>	<b>107</b>
7.1	Calculate $\mu_k$ for Exponential Distribution . . . . .	107
7.2	Calculate $var_k(x)$ for Exponential Distribution . . . . .	108
7.3	The Model Assuming a Uniform Distribution . . . . .	109
	<b>References</b>	<b>113</b>
	<b>Curriculum Vitae</b>	<b>119</b>

# List of Tables

2.1 A taxonomy of P2P literature based on: structure, centralization and topology-  
awareness . . . . . 12

# List of Figures

- 1·1 Our proposed conceptual framework. . . . . 8
- 3·1 Clients/angel forward content received from provider to other clients. . . . . 21
- 3·2 During phase one, the provider sends each client its perspective block which in its turn forwards part of it to the other clients. . . . . 22
- 3·3 The state of the file reception after time  $W$  for client 1 . . . . . 23
- 3·4 Illustration of the GT strategy. In phase 1,  $k = 4$  different segments are disseminated downward through  $k = 4$  binary distribution trees. In phase 2, segments are disseminated laterally within cliques of size  $k = 4$ . . . . . 29
- 3·5 A comparison between GT with fan-out 2 and GT with fan-out 3. . . . . 31
- 3·6 Performance of a direct implementation of the MDT construction and of the GT strategy (also showing the analytical MDT lower bound). . . . . 33
- 3·7 Performance of GT as a function of the number of angels. . . . . 34
- 3·8 MDT for various strategies as a function of the number of clients. . . . . 35
- 3·9 Average distribution time as a function of the number of clients. . . . . 36
- 3·10 The relationship between MDT and the number of clients for swarming-based (CLOUDANGELS and BT) versus non-swarming-based (client-server) approaches. . . . . 40
- 3·11 The average distribution time and the number of clients for swarming-based (CLOUDANGELS and BT) versus non-swarming-based (client-server) approaches. . . . . 41
- 3·12 The impact of angel’s upstream capacity on MDT . . . . . 42
- 3·13 MDT as a function of number of clients for fixed angel capacity. . . . . 43
- 4·1 Illustrative examples of the optimal construction: 3 clients not in need of any angels (left) and 2 clients in need of one angel (right). . . . . 53

4.2	Our proposed ecosystem; clients download more than they can upload, and angels download as little as possible. . . . .	54
4.3	The number of angels versus the number of servers needed to achieve a desired playout rate for 100 clients. . . . .	56
4.4	The number of clients that can be supported by a fixed number of angels or servers. . . . .	57
4.5	Node F joins the tree, the registrar updates the data structure accordingly. . . . .	64
4.6	A hypothetical scenario illustrating the formation of <i>AngelCast</i> trees when clients join, leave or change parents. . . . .	65
4.7	An interaction diagram showing the exchange of messages between a new client, the registrar and the parents. . . . .	68
4.8	The frame drop ratios of <i>AngelCast</i> vs SopCast. . . . .	70
4.9	Minimal amount of angel capacity is sufficient to achieve good stream rates. . . . .	70
4.10	The performance of <i>AngelCast</i> under churn. . . . .	71
4.11	The performance of <i>AngelCast</i> under churn where clients rejoin the stream after they leave. . . . .	72
4.12	The frame drop ratio for non-churning clients against varying churning rate for other clients. . . . .	73
4.13	The frame drop ratio of clients against an increasing start-up buffer. Different curves denote different churning levels. . . . .	74
4.14	The frame drop ratio of clients against an increasing number of substreams (trees). . . . .	75
5.1	The correlation between the duration to download a file (y-axis) and the sum of the geo-distances between the relays in the circuit (x-axis). . . . .	87
5.2	The correlation between the duration to download a file (y-axis) and the delay to download just one byte using the same circuit (x-axis). . . . .	88
5.3	The correlation between the throughput of a circuit (y-axis) and the raw capacity of its constituent relays (x-axis). . . . .	89
5.4	PDF of relay available capacities, decaying exponentially. . . . .	90

5·5	The CV of the throughput of the best-of- $k$ circuit. . . . .	93
5·6	The ratio between the CV of the signal sent the PID controller with filtering and without filtering as a function in the number of probes. . . . .	95
5·7	The performance of different probe filtering techniques against an increasing number of circuits from which the best is chosen. . . . .	97
5·8	The PDF and CDF of relay available capacities under two configurations: (1) Filtering, $k = 1$ , and (2) No filtering, $k = 6$ . . . . .	98
5·9	The effects of various parameters on CV of circuit throughput (top) and utilization of relays (bottom). . . . .	99
5·10	The effects of splitting a circuit into multi-paths. . . . .	102

## List of Abbreviations

BSP	.....	Bulk Synchronous Parallel computing model
CDF	.....	Cumulative Distribution Function
CDN	.....	Content Distribution Network
CP	.....	Content Provider
CV	.....	Coefficient of Variation
DA	.....	Directory Authority
DASH	.....	Dynamic Adaptive Streaming over HTTP
FTP	.....	File Transfer Protocol
GRF	.....	Global Rarest First
GT	.....	Group Tree
HTTP	.....	HyperText Transfer Protocol
LRF	.....	Local Rarest First
MDT	.....	Minimum Distribution Time
P2P	.....	Peer-to-Peer
PDF	.....	Probability Distribution Function
PID	.....	ProportionalIntegralDerivative
QoS	.....	Quality of Service
RTT	.....	Round-Trip Time
TCP	.....	Transmission Control Protocol
TOR	.....	The Onion Router
USM	.....	Uplink Sharing Model

# Chapter 1

## Introduction

### 1.1 Motivation

Peer-assisted systems are systems where clients not only consume resources but also contribute resources. Such systems have been increasingly deployed especially in the context of peer-assisted content delivery, the focus of this thesis. The main benefits of peer-assisted systems are: (1) their ability to mitigate the costs associated with relying solely on infrastructure resources, (2) their ability to scale, and (3) their resilient against failures. The main hindrances to the wide deployment of peer-assisted systems are: (1) the intermittency of client availability, (2) the need to give clients incentives to contribute their resource, as well as (3) the trust, security and privacy issues associated with depending on non-trusted clients.

Nowhere the potential of peer-assisted systems is more prevalent than the area of peer-assisted content delivery, where the system utilizes the client resources in addition to infrastructure resources. Examples of such systems include Akamai's Netsession (aka, 2012), Octoshape Infinite Edge (oct, 2012), Pando (pan, 2012), and BitTorrent DNA (bit, 2012). This is due to two reasons: (1) the increased usage of bandwidth-intensive application, highlighting the benefits of the P2P paradigm and (2) the persistent gap between the resources contributed versus consumed by clients, highlighting the continued need for infrastructure resources. As for the former reason, delivering bandwidth-intensive content over the Internet is becoming a standard expectation of clients, posing significantly different challenges for today's Content Providers (CPs) as well as Content Distribution Networks (CDNs). For example, Netflix reported that it delivered one billion hours of video streaming to its clients during the month of June 2012 (san, 2012), an earlier study shows that Netflix delivers video at a rate between 1.4Mbps and 3.0Mbps (Net, 2010). The same report (san,

2012) suggests that internet traffic in the US increased by 64% during the second half of 2012, 70% of which is attributed to real-time entertainment and file-sharing applications.

The latter reason, the persistent gap between the resources contributed versus consumed by clients, is of an equal importance. Typically, ISPs offer two or three times downstream capacity more than upstream capacity. As of November 2012, Verizon FiOS offers five packages, 15/5 Mbps downstream/upstream capacity, 50/25 Mbps, 75/35 Mbps, 150/65 Mbps, and 300/65 Mbps. Comcast Xfinity offers 20/4, 50/15, and 105/20 Mbps. This has been enshrined in the ADSL standard of frequency allocation by dedicating 31 frequency bins to upstream connections while allocating 224 frequency bins to downstream connections (Grube et al., 1996). The aforementioned numbers illustrate an inherent shortcoming in pure P2P systems and a potential for peer-assisted systems. The persistent gap between the average downstream and the average upstream capacity of clients creates a persistent gap between the rate at which clients are expecting to consume content and the rate at which clients are able to contribute content, resulting in an imbalanced ecosystem.

This gap cannot be overlooked in the case of applications with streaming nature, in which content has to be fresh. Clients at any point in time will demand significantly higher average download rate than their average upstream capacity. This means a pure P2P architecture is not likely to be able to maintain a delivery rate that matches the clients' downstream capacity. Other P2P applications, such as BitTorrent and VoD, are less likely to significantly suffer from this gap because some peers linger in the system after completing their download, allowing other clients to utilize them as seeders.

## 1.2 Thesis Premise

The premise of this thesis is to build cloud-based peer-assisted optimized content delivery systems for applications with fresh content. The systems we build are: (1) peer-assisted: to utilize the clients' resources, (2) cloud-based: to deploy additional cloud resources, we call them *angels*, to maintain a certain fidelity of service, (3) coordinated: because adding more resources without the judicious use of the already available resources leads to inefficiencies. Our setup is most

suitable for fresh content delivery applications where content cannot be cached and the amount of contributed resources by clients, *i.e.*, upstream capacity, is significantly less than the consumed resources, *i.e.*, downstream capacity.

In this thesis, we address three applications, each is studied in depth in a separate chapter. The first application is bulk-synchronous content distribution: where a content provider aims to minimize the maximum distribution time (MDT) of a piece of content to a set of clients. Achieving MDT is crucial for bulk-synchronous applications, in which every client must wait for all other clients to finish their download before being able to make use of the downloaded content. Examples of such applications are: enterprize-wide system administration requiring synchronous software patching or data replication, virtual community games and simulated reality environments requiring common content such as terrain information to be accessible to all players before any progress could be made, publish-subscribe networks and distributed data stores requiring consistency across multiple sites, among many others. In bulk-synchronous applications, clients contribute resources (upstream capacity) and our system complements them with resources (more upstream capacity) from the cloud (angels). We coordinate the content delivery by choreographing the connectivity of nodes to each other and the data dissemination rate between them.

The second application is live video streaming where a content provider wants to stream a live video to a set of clients at a desired bit-rate without dropping too many video frames. Because a typical client has an average upstream capacity less than the stream bit-rate, to overcome this shortcoming, we coordinate the deployment of the client's resources, complementing them with angels from the cloud.

The third application is improving the performance of bandwidth-intensive applications over Tor, the onion routing protocol. In Tor, the problem of clients limited upstream capacity is compounded by the fact that for a client to download a single byte it does not consume just one byte of the aggregate upstream capacity in the system as in bulk-synchronous or live video streaming, but it has to consume a byte of the upstream capacity of each of the circuit's constituent relays. We address the problem of performance unpredictability experienced by Tor clients, and the potential of improving the performance of bandwidth-intensive applications through efficient utilization

of clients' and relays' resources as well as the on-demand deployment of angels.

For each of these three applications we start by developing analytical models, in order to understand how to efficiently allocate the already available resources as well as the added angels. Section 1.3 of the introduction summarizes the analytical models developed across the three applications. Then, we develop practical dissemination mechanisms to achieve the required level of service fidelity while utilizing near minimal angel capacity. Section 1.4 provides a cross-section overview of the practical dissemination strategies for the three applications. We finish with evaluating the proposed systems through system building/evaluation or extensive simulation. Section 1.5 summarizes the experimental evaluation across the three applications.

### 1.3 Analytical Models

The analytical models used in the first two applications; bulk-synchronous and live video streaming, are based on the Uplink Sharing Model (USM) presented by Munding et al. (Munding et al., 2008), wherein each client is defined solely by its upstream and downstream capacities. The client is free to divide its upstream/downstream capacity arbitrarily among the other nodes as long as the aggregate upload/download rates do not exceed the upstream/downstream capacity. In this ecosystem, the aggregate download rate has to equal the aggregate upload rate. But the client constrained upstream capacity cannot match the desired download rate for the content. Thus, we need to add angels to this ecosystem to supplement this capacity gap. The analytical model in torAssist is not based on USM as a central authority cannot dictate the circuit connectivity in order not to compromise client anonymity. Nevertheless, each Tor relay is defined solely based on the distribution of its available upstream capacity.

For bulk-synchronous applications, the model suggests that MDT is bounded by one of three potential bottlenecks: (1) the provider's upstream capacity, (2) the slowest client downstream capacity, or (3) the aggregate upstream capacity of all nodes. We note that the third of these bottlenecks, the *aggregate upstream capacity*, is likely to be the most prevalent hence the potential of angels to improve the MDT. Assuming the ability to infinitesimally divide the capacities, we con-

struct an optimal dissemination strategy that allows us to minimize the MDT, achieving the lower bound. The optimal dissemination strategy choreographs the exact connection between nodes, dictating the exact data transfer rate between them. The key observation here is that angels are optimally utilized when they download a small fraction of the content and forward it to as many clients as possible.

For live video streaming applications, using USM, we provide a lower bound on the minimum amount of angel capacity needed to maintain a desired bit-rate to all clients. We also develop an optimal fluid-model construction that achieves this lower bound. We show that, under the optimal construction, the start-up delay grows linearly with the number of clients but this quantity diminishes to zero under the fluid assumption. Similar to the bulk-synchronous applications, the angels are most utilized when they download a fraction of the stream and upload it to as many clients as possible.

Before developing analytical models for Tor’s bandwidth-intensive applications, we need to investigate the source of performance degradation and the inability of Tor network to utilize the readily available resources efficiently. Thus, we start by conducting an in-vivo measurement study to investigate the mechanisms used to distribute the load evenly on Tor relays. The findings are: (1) The throughput of different circuits is widely variable, (2) the throughput of a circuit is dominated by the available capacity of its constituent relays, not the throughput of the links between relays, and (3) the available capacity of relays can be approximated using an exponential distribution. These observations highlight the deficiencies of Tor’s load distribution mechanisms resulting in: (1) overloading certain relays, leading to circuits with poor throughput, and (2) under-utilizing certain relays resulting in a reduction in the overall system utilization – both of which are undesirable attributes.

To improve Tor’s load balancing, we need to understand it first. Tor uses three interactive mechanisms to achieve a balanced load distribution: (1) a probing mechanism to estimate the available capacity of relays that is not assigned to any circuit, (2) a feedback mechanism to compute a normalized value that corresponds to the total capacity of the relay, and (3) a circuit construction mechanism that assigns more circuits to under-utilized relays.

To capture the interaction between these three mechanisms, we develop an analytical model for the feedback control inherent in Tor’s load distribution mechanisms. Assuming that the relay available throughput follows a certain probability distribution, *i.e.*, exponential or uniform, an order statistics model demonstrates that a significant improvement, in terms of overall network utilization as well as lower variability of individual circuit throughput, can be achieved through changing the probing and the circuit selection mechanism. The order statistics mode also shows that using angels as exit nodes for multi-path routing decreases the variance in circuit throughput.

#### 1.4 Practical Content Delivery Strategies

In this section we provide a cross-section overview of the practical content delivery strategies used in the three application, starting with the bulk-synchronous transfer. Due to the impracticalities of the optimal construction, namely the ability to infinitesimally divide a node’s capacity and the need for each client to download from all the other clients concurrently, we develop a practical coordinated dissemination strategy involving angels, we call it *Group Tree* (GT). Phase one of GT divides the clients, based on their upstream capacity, into different multi-cast trees. Each tree is responsible for the dissemination of a segment of the file whose size is proportional to the average client upstream capacity in it. Angels form their own tree. Phase two of GT forms cliques of clients. Each node in a clique comes from a different tree and is responsible for disseminating its segment to the other clients in the clique. Angels join all cliques in phase two and upload their segment without downloading more content. GT enables all clients to start downloading a segment of the file right away, uploading it to other clients shortly after. This allows us to take advantage of each client’s upstream capacity which is the scarce resource in the system. Also, GT instructs angels to download just one segment of the file to conserve the system scarce resource, the upstream capacity.

We now discuss the practical dissemination strategy for our second application, live video streaming. Realizing the limitations of the fluid model construction – namely, susceptibility to potentially arbitrary start-up delays, the need to form a full mesh resulting in massive disruption in case of churn – we present a practical multi-tree construction that captures the spirit of the optimal

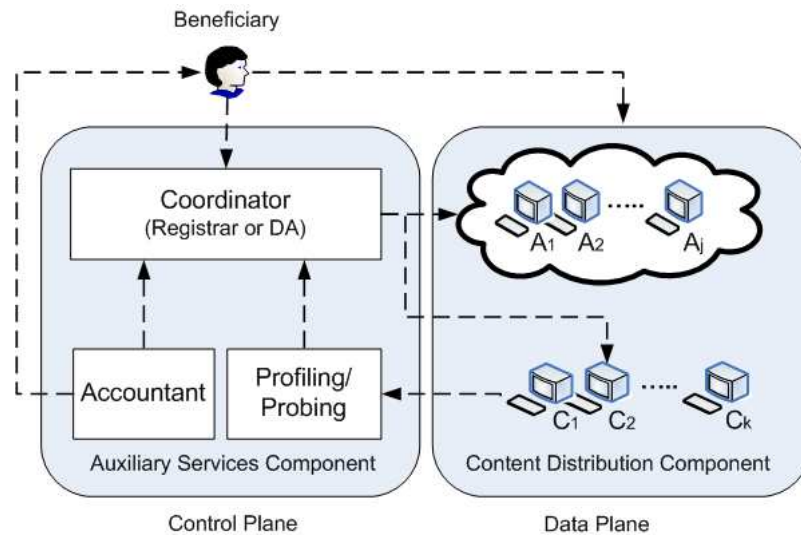
construction, while avoiding its limitations. The stream is divided into a number of substreams of equal rates, each of which is disseminated in a multicast tree. This is slightly different from phase one of GT, our dissemination strategy for bulk transfer applications, where each tree has a different rate. The idea of streaming content over multiple trees was used before in SplitStream (Castro et al., 2003) and CoopNet (Padmanabhan et al., 2003). In our practical construction, a new client contacts a coordinating node, the *registrar*, which instructs it as to which node to download each substream from. The registrar maintains information about the capacity and connectivity of nodes allowing it to handle membership management tasks, such as arrival, departure and degradation of service. Our *AngelCast* protocol achieves near optimal performance (compared to the fluid-model construction) while ensuring a low startup delay by maintaining a logarithmic-length path between any client and the provider, while gracefully dealing with churn by adopting a flexible membership management approach.

Our third application is Tor’s bandwidth-intensive applications. While all clients can benefit from the addition of angels to a highly utilized and well-balanced Tor network, we still lack a mechanism to dedicate this added capacity towards bandwidth-intensive applications. To this end, we introduce a simple angel functionality that boosts the performance of bandwidth-intensive circuits using multi-path routing. Unlike CLOUDANGELS nor *AngelCast*, Tor clients are not interested in the same content. Thus, multi-path, not a multi-tree construction is adequate. Whereas An angel would act as an exit node and split the reply to the client across multiple paths, maximizing the circuit throughput. This modification requires special clients and angels, but the relays can be agnostic to this modification allowing us to use the existing Tor network resources.

## 1.5 Experimental Evaluation

Figure 1-1 shows the blueprint of our proposed conceptual framework across all three applications. The beneficiary is the party interested in enhancing the performance. In the case of bulk-transfer or live video streaming it is the Content Provider (CP). In case of Tor, it is the set of the clients aiming at enhancing the performance for their bandwidth intensive applications. The coordinator is the main agent of our proposed system. The beneficiary requests help from the coordinator

and the clients are directed to contact the coordinator which uses profiling/probing to gauge their upstream capacity. Consequently, the coordinator manages the connectivity of clients to achieve efficient utilization of the resources already in the system. In the case of bulk-synchronous or live video streaming, the coordinator choreographs the connectivity of clients and at what rate data is exchanged between them. In the case of Tor, the coordinator publishes the estimated client/relay capacities in order to balance the load on them, increasing their upstream capacity utilization. Efficiently utilizing the already available resources allows us to minimize the amount of angel capacity the coordinator deploys on-demand from the cloud. We call these additional resources angels. The coordinator also manages the angels in order to most efficiently utilize their upstream capacity. The accountant charges the beneficiary the cost of the deployed angels. In the case of bulk-synchronous or live streaming, the price charged reflects the cost of deployed angels. In the case of Tor, the price charged could be a monthly subscription fee, as it is hard to charge clients for the exact deployed resources without compromising their anonymity.



**Figure 1-1:** Our proposed conceptual framework.

For experimental evaluation of the bulk synchronous application, we conduct experimental

evaluations to compare GT and a few other dissemination strategies. Our findings suggest that, in contrast to focusing only on piece or peer selection strategies, a strategy that incorporates both criteria holds the potential for significant performance gains. We present the architecture and implementation of CLOUDANGELS a service that allows the elastic, on-demand deployment of angels (in the cloud) to assist a content provider (off the cloud) in realizing its MDT objective. We deploy CLOUDANGELS in Emulab and show that it scales well with the size of the client set and that its performance is slightly better than that of BitTorrent even for the average distribution time and without deploying angels. We also show that as we increase the upstream capacity of angels the MDT decreases and the achieved MDT is near the lower bound.

For live video streaming, we implement *AngelCast* using python and deploy its agents in Planetlab and Emulab. Our implementation is capable of delivering Microsoft-encoded VC-1 live video streams to a dynamic set of clients. Our experimental results confirm the feasibility and performance potential of our *AngelCast* service when deployed on Emulab and PlanetLab. When the upstream capacity is abundant, the frame drop-ratio of *AngelCast* and SopCast, the popular P2P live streaming protocol, are comparable. When the upstream capacity is scarce, we find that; (1) *AngelCast* utilizes near minimal angel capacity to achieve the desired playout rate, and (2) our membership management algorithms are able to handle churn effectively.

For Tor, we simulate the whole network including relays, clients and the Directory Authority (the entity responsible for probing and balancing the load on relays). Our simulations show that the proposed modifications to both the probing mechanism and circuit-selection mechanism achieve tangible improvements. Also, in these simulations, assumptions are relaxed and various mechanisms are evaluated. These results show that all clients can benefit from the addition of angels to a highly utilized and well-balanced Tor network. They also show that multi-path routing can be used to dedicate the angel added capacity towards bandwidth-intensive applications.

## 1.6 Thesis Organization

The organization of the thesis is as follows: Chapter 1 is an introduction, in which we explain the motivation behind our work, present the premise of the thesis, and list the main contributions across

the three application we target. Chapter 2 presents a taxonomy of peer-assisted content delivery systems along three dimensions: structure, centralization and topology-awareness. In Chapter 3 we discuss CLOUDANGELS where we deploy angels to minimize the maximum distribution time for bulk-synchronous delivery. In Chapter 4 we discuss our second application; *AngelCast*, a cloud-based peer-assisted live streaming service. In Chapter 5, we discuss TorAssist where we deploy angels to enhance Tor's performance for bandwidth-intensive applications. Chapter 6 summarizes our contributions in this thesis.

## Chapter 2

### Related Work

In this thesis, we argue for the use of the peer-assisted content delivery systems especially for applications where content has to be fresh and cannot be cached. To dissect the plethora of P2P literature, we use three dimensions: (1) structured vs unstructured systems: we define a structure content delivery system as a system with *persistent connections* between peers where content keeps flowing over the same connections as long as they are reliable, perhaps for the duration of the content delivery, (2) centralized vs distributed systems: in centralized systems a central authority *decides* which nodes are connected and what content is transferred between them, in a distributed system, each client is responsible for taking these decisions, (3) topology-aware vs topology-agnostic: where the underlying topology and the connectivity conditions are part of connectivity decisions or not. For the three applications we target we favor structured peer-assisted content distribution systems where connections are persistent. The structure allows us to minimize the angel deployment while allowing us to maintain the required level of service quality. It is hard to minimally provision angel deployment in unstructured systems where it is infeasible to identify where, when and by how much angels' capacity is required. That being said, managing structured peer-assisted systems is hard in a distributed fashion thus we resort to using centralized authority that collects node information and decides on their utilization levels. In the case of bulk-synchronous and live streaming the central authority (the registrar) is responsible for deciding which nodes are connected and at what rate content is transferred between them. In the case of Tor, each client has to choose the relays in its circuit, as delegating this decision violates the anonymity requirement in Tor. Yet, we argue for enhancing Tor's centralized authority (the Directory Authority) to better Tor's load balancing mechanism, increasing the node utilization and minimizing the variance in circuit throughput.

The peer-assisted content distribution landscape is also divided between topology-aware and topology-agnostic systems. Our systems do not explicitly optimize based on the underlying network topology. Even though in the live streaming and Tor applications we deploy mechanisms to avoid degraded end-to-end connections, we do not actively optimize based on the network topology. In today’s internet, the bottleneck is more likely to be at the edge of the network (Cataltepe and Moghe, 2003), which is the main reason why CDNs like Akamai deploy their servers at the edge of the network. For P2P applications, the bottleneck is typically at either end of the overlay connection and rarely in the middle of the network, thus for the sake of simplicity we deploy topology-agnostic content delivery mechanisms.

Table 2.1 provides a taxonomy of P2P and peer-assisted content delivery systems based on the three aforementioned dimensions: structure, centralization and topology awareness. The following is a detailed comparison of these systems to the ones we introduce in this thesis.

P2P Systems	Structured	Centralized	Topology-aware
OSMOSIS (Bestavros and Jin, 2003)	Y	Y	Y
(Sweha et al., 2012b; Sweha et al., 2011) (Kumar et al., 2007; Kumar and Ross, 2006)	Y	Y	N
(Sherr et al., 2009) (Smaragdakis et al., 2008) dPAM (Sharma et al., 2005)	Y	N	Y
Splitstream (Castro et al., 2003) P2PCast (Nicolosi and Annapureddy, 2003) CoopNet (Padmanabhan et al., 2003) (Dhungel et al., 2010) (Sweha et al., 2012a) (Snader and Borisov, 2010)	Y	N	N
(Aggarwal et al., 2007)	N	Y	Y
dHCPS (Guo et al., 2008a) Antfarm (Peterson and Sirer, 2009) CYCLOPS (Michiardi et al., 2012)	N	Y	N
ONO (Choffnes and Bustamante, 2008) (Bindal et al., 2006)	N	N	Y
BitTorrent (Cohen, 2003) CoolStreaming (Zhang et al., 2005) (Wang et al., 2007)	N	N	N

**Table 2.1:** A taxonomy of P2P literature based on: structure, centralization and topology-awareness

The closest work to our bulk-synchronous and live streaming systems is by Kumar et al. (Kumar et al., 2007; Kumar and Ross, 2006) where they used a structured, centralized and topology-agnostic delivery mechanism like us. The first paper addresses the bulk-synchronous applications from an analytical point of view using the Uplink Sharing Model (Mundinger et al., 2008). We extend the analytical model to incorporate angels and then propose a practical dissemination strategy (GT) and build a system and deploy it in Emulab. For the second paper they used (Mundinger et al., 2008) again to find the maximum bit-rate of a live video streaming swarm. Again we extend the analytical model to include angels, build the *AngelCast* system and deploy it in both Emulab and Planetlab. OSMOSIS (Bestavros and Jin, 2003) is a structured, centralized and topology-aware cache-and-relay VoD delivery mechanism with playback functionality, *i.e.*, skip, rewind and fast forward. In this system a client searches for the closest peer with excess capacity to download the whole video from. OSMOSIS is a typical case of end-system multicast where an end-user acts as a source for other end-users uploading the whole content to them. dPAM (Sharma et al., 2005) is a distributed prefetch-and-relay version of OSMOSIS. The main difference between OSMOSIS and dPAM on the one hand and CLOUDANGELS and *AngelCast* on the other is that we deal with fresh content where there is no playback functionalities or a chance for prefetching.

(Smaragdakis et al., 2008) is a structured distributed topology-aware  $n$ -way broadcast protocol where each client in the swarm wants to disseminate synchronously a different file to the other  $n$  clients. This is different from bulk-synchronous application where all the clients want to download the same file. The main contribution of this paper is that despite the peer selection decision of each peer is made independently, the swarm manages to achieve its overarching goal of minimizing the time to download all the files to all clients. (Sherr et al., 2009) is another structured, distributed, topology-aware content distribution protocol that aims at enhancing Tor's performance through allowing clients to probe the underlay and construct circuits with small link-latency. We conducted a live measurement over the Tor network and found little evidence that choosing circuits based on link-latency would impact the Tor network positively. A node-based circuit selection yields higher node utilization and less variance in circuit throughput. Thus we opt for a topology-agnostic circuit selection strategy in Tor.

Similar to our TorAssist protocol (Sweha et al., 2012a), (Dhungel et al., 2010) and (Snader and Borisov, 2010) argue for a topology-agnostic circuit selection in Tor. One protocol is different as it allows for the establishment of multi-path routing to enhance the performance of bandwidth-intensive applications, the other differences are the changes we propose for the Directory Authority’s probing and circuit selection mechanisms.

The same category of structured, distributed and topology-agnostic includes a few video streaming systems, such as SplitStream (Castro et al., 2003), P2PCast (Nicolosi and Annapureddy, 2003), and CoopNet (Padmanabhan et al., 2003). They are similar to *AngelCast* is that they rely on a multi-tree construction, *AngelCast* is different in its reliance on a central authority (the registrar) to make connectivity decisions. We argue that having such central authority enables *AngelCast* to make better resource provisioning decisions. We safeguard against performance degradation due to churn by relying on proactive membership management. We discuss these systems in detail in Section 4.6 supporting our argument with experimental evaluation.

The second half of the table is for unstructured P2P systems where connections are not persistent and each chunk of the content is most likely to traverse a different spanning tree. The poster child of these systems is the widely used and studied BitTorrent protocol (Cohen, 2003). Cool-Streaming (Zhang et al., 2005) is the video streaming version of BitTorrent. The difference is that the deadline of a chunk playtime is a factor in the piece selection algorithm. Wang et al (Wang et al., 2007) proposed adding powerful peers to BitTorrent swarms to accelerate the download rate in an uncoordinated fashion. This is not efficient, because these “helpers” will unnecessarily consume the scarce upstream capacity in the swarm to download the whole file. All of the three aforementioned protocols are unstructured, distributed and topology-agnostic. There have been a large body of work around them, we focus on a few of them to showcase our taxonomy. For topology-aware protocols we discuss three protocols. First, (Aggarwal et al., 2007) relies on a central oracle that suggests connectivity information to clients based on the network topology. ONO (Choffnes and Bustamante, 2008) is the distributed version of that where each client uses

CDN-hints to decide which other clients are topologically close-by. (Bindal et al., 2006) is similar to ONO in being a distributed topology-aware unstructured protocol where clients bias their peer selection towards other clients in the same ISP to improve locality. This modification does not need a centralized authority. As for unstructured, centralized and topology-agnostic protocols, we start with dHCPS (Guo et al., 2008a) which resembles GT in that it relies on a hierarchical P2P dissemination mechanism for streaming, whereby each cluster has a Head (the source for this cluster) which is a member of a superNode swarm fed by the source. dHCPS solves the optimization problem for deciding how much upstream capacity each head should dedicate to the superNode swarm versus its own group. Antfarm (Peterson and Sirer, 2009) and CYCLOPS (Michiardi et al., 2012) are similar to GT and *AngelCast* in that a central authority decides on when to deploy extra resources and by how much. Antfarm is a content distribution system that measures the vital signs for each swarm by plotting the *Response Curve*, which is the relationship between the seeder upload rate and the average download rate in the swarm. Antfarm allocates more seeder rate for swarms with steeper response curve. CYCLOPS deploys the minimum amount of cloud resources to maintain the availability of content in a BitTorrent swarm, *i.e.*, as long as chunks are available in the swarm CYCLOPS does not need to make these chunks available through the cloud.

To conclude, our taxonomy of the peer-assisted content delivery literature is based on three dimensions, structure, centrality and topology-awareness. Each of the following three chapters has a related work section that contrasts details pertaining to our specific approach to other approaches in the literature.

## Chapter 3

# CloudAngels: A Peer-Assisted Bulk-Synchronous Content Distribution Service

Leveraging client upstream capacity through peer-assisted content distribution was shown to decrease the load on content providers, while also improving average distribution times. These benefits, however, are limited by the disparity between client upstream and downstream capacities, especially in scenarios requiring a minimum distribution time (MDT) of a fresh piece of content to a set of clients. Achieving MDT is crucial for *bulk-synchronous* applications, when every client in a set must wait for all other clients in the set to finish their downloads before being able to make use of the downloaded content. In this chapter, we propose the use of dedicated servers, which we call *angels* to accelerate peer-assisted content distribution in general, and to minimize MDT in particular. An angel is not itself the content origin, nor is it interested in fully downloading the content; its only purpose is to enable a peer-assisted content distribution scheme to approach the theoretical lower-bound for MDT. To overcome scalability issues inherent in an optimal MDT construction, we propose and evaluate a content exchange strategy involving angels, which we call *Group Tree*. In addition to simulation results that demonstrate the near optimal performance of our proposed approach, we present the architecture and implementation of CLOUDANGELS – a service that allows the elastic, on-the-fly deployment of angels (in the cloud) to assist a content provider (off the cloud) in realizing its MDT objective.

### 3.1 Motivation and Problem Statement

Existing peer-assisted content distribution systems can be seen as “ad hoc” or “best effort” in the sense that a client upstream capacity is tapped only to the extent that other swarming clients are

able to utilize it. Indeed, most mechanisms proposed and implemented for peer selection do not *primarily* aim to maximize the utility of a swarm’s aggregate capacity, but rather focus on other goals, including dealing with freeloaders using rational tit-for-tat exchanges (Cohen, 2003), or reducing inter-ISP traffic using geographic (or topological) locality (ope, 2008). These approaches are well justified when peer-assisted content distribution is not under the control of a single authority or does not cater to a common overarching “socially optimal” (not to mention legal) objective.<sup>1</sup> We contend that for many emerging applications and settings, either or both of these conditions may not hold. In particular, when content providers employ peer-assisted delivery mechanisms, they are in fact acting as a single authority that choreographs the operation of all clients involved, using specially-developed software clients. Moreover, for many emerging cooperative applications, optimizing the performance of individual clients may not be a rational objective as it may lead to suboptimal group performance.

Motivated by the above observations, in this chapter we tackle the problem of peer-assisted content delivery in a setting where it is assumed that (1) the content provider choreographs the participation of all clients in a swarm, and/or (2) the objective of the content delivery system is to minimize the time it takes to distribute a common piece of content to all clients in the swarm.

**Bulk-Synchronous Content Distribution:** Our focus in this chapter is on systems with a *Minimum Distribution Time* (MDT) objective. An MDT objective implies that the overarching common goal of the provider *and* clients is to minimize the time by which *all* clients finish their download. The need for such a bulk synchronous mode of operation (which is not new (BSP, 2007)) is paramount, with applications to: enterprise-wide system administration requiring synchronous software patching or data replication, virtual community games and simulated reality environments requiring common content such as dynamic simulated terrain information to be accessible to all players before any progress could be made, publish-subscribe networks and distributed data stores requiring consistency across multiple sites, among many others. In addition to applications where synchronization is over content, the MDT objective may be desirable for distributed multi-

---

<sup>1</sup>A file-sharing application is a canonical example where both of these conditions hold. Indeed, early research on swarming protocols was singularly geared to issues stemming from file-sharing applications.

agent applications in which the delay in one agent’s response may negatively impact the overall fidelity of the system, *e.g.*, an advanced alert system composed of a set of distributed monitoring agents that download and process a common live feed: independently analyzing it using a different anomaly detection approach and reporting any security breaches to a central authority.

To achieve an MDT objective for a group of clients requires a judicious use of client upstream and downstream capacities, which are typically highly asymmetric as most ISPs offer their clients download speeds that are significantly larger than upload speeds. Using an ad-hoc, best-effort swarming mechanism, this disparity may result in an under-utilization of the downstream capacity to the clients, even if the upstream capacity of clients in the swarm is fully tapped. In a video dissemination system, this may not be a problem as long as the download rate to individual clients is larger than the playback rate (Huang et al., 2007). But, for bulk content delivery that is not subject to a nominal playback rate, under-utilizing the swarm capacity is problematic.

Kumar and Ross (Kumar and Ross, 2006) derived a theoretical lower-bound on MDT and proposed a fluid model that achieves this bound. Their result suggests that MDT is limited by one of three potential bottlenecks in any P2P overlay: (1) the seeder upstream capacity, (2) the slowest client downstream capacity, or (3) the aggregate upstream capacity of all clients in the swarm. We note that the third of these bottlenecks, namely the *swarm upstream capacity*, is likely to be the most prevalent in many settings, due to the aforementioned disparity between the upstream and downstream capacities of most clients.<sup>2</sup>

**Chapter Contribution and Outline:** In support of the MDT objective for bulk-synchronous content distribution, in this chapter we investigate the potential benefit from the on-demand deployment of cloud resources to alleviate the swarm upstream capacity bottleneck. To that end, we propose the use of helper nodes – which we call *angels*.<sup>3</sup> An angel is not a client in the sense that it is not interested in receiving the entire content (file) to be distributed, but rather it is interested in minimizing the MDT to all clients. As such, an angel uses its storage and up/down-link capacity to

---

<sup>2</sup>We also note that nothing can be done “inside the network” (*e.g.*, by leveraging cloud resources) to alleviate the first two potential bottlenecks.

<sup>3</sup>The idea of introducing helper nodes was also considered in (Wang et al., 2007), where the focus was on using idle nodes to improve BitTorrent’s steady-state performance as opposed to optimizing any specific objective.

cache and forward parts of the file to other peers, in such a way that the swarm upstream capacity ceases to be the limiting factor for MDT. Doing so ensures that the content provider is able to fully leverage the upstream/downstream capacities of the clients.

In Section 3.2, we extend the analytical results in (Kumar and Ross, 2006) to account for the presence of angels by deriving a new lower bound on MDT. Also, we show that this new lower bound is tight by constructing a distribution strategy under a fluid model assumption. Simulation results show that a direct implementation of our fluid construction is impractical. Therefore, in Section 3.3, we present a coordinated swarming strategy – called *Group Tree* (GT) – that addresses the impracticalities of the fluid model to fully utilize angels. Simulation results show that GT outperforms other strategies, scales well with the increase in the number of clients, and operates near the optimal theoretical bounds. These findings suggest that, in contrast to focusing only on piece or peer selection strategies, a strategy that incorporates both criteria holds the potential for significant performance gains. Armed with the promise from these simulation results, in Section 5.6, we present the blueprints of CLOUDANGELS – an “angels on demand” cloud service that complements peer-assisted content delivery to achieve an MDT objective. In Section 3.6, we report results from live Emulab experiments in which our CLOUDANGELS service is used in a real content distribution sessions.

## 3.2 Optimal MDT Construction

### 3.2.1 MDT Problem Statement

We aim to minimize the distribution time of a file of size  $F$  from a set of Content Providers (providers),  $p \in P$  to a set of Clients,  $c \in C$ , with the help of a set of angels,  $a \in A$ . Following the Uplink-Sharing fluid model presented in (Mundinger et al., 2008) and adopted by (Kumar and Ross, 2006), our goal is to minimize  $T$ , where  $T = \max_{i \in C} \{T_i\}$  and  $T_i$  is the time it takes client  $i$  to finish the download. The fluid model must make sure that each node’s total upload or download rate, does not exceeds neither its upstream capacity  $u(i)$  nor its downstream capacity  $d(i)$ , which under a fluid model can be infinitesimally divided. Specifically:

$$u(p) \geq \sum_{i \in \{C, A\}} x_{pi}, \text{ Provider upstream capacity} \quad (3.1)$$

where  $x_{pi}$  is the upload rate from  $p$  to  $i$

$$u(i) \geq \sum_{j \in \{C, A\}} x_{ij}, \text{ Client upstream capacity, } \forall i \in C \quad (3.2)$$

$$d(i) \geq x_{pi} + x_{ai} + \sum_{j \in \{C\}} x_{ji}, \text{ Client downstream capacity, } \forall i \in C \quad (3.3)$$

$$u(a) \geq \sum_{i \in \{C\}} x_{ai}, \text{ Angel upstream capacity} \quad (3.4)$$

$$d(a) \geq x_{pa} \text{ Angel downstream capacity} \quad (3.5)$$

The Uplink-Sharing model (Mundinger et al., 2008) does not account for network cross-traffic or losses. Thus, for notational convenience, we aggregate the set of providers and the set of angels into one provider and one angel, with the upstream/downstream capacity of the provider and of the angel set to the aggregate capacities of all the providers and of all the angels, respectively. Chiu *et al.* proved the correctness of such an abstraction (Chiu et al., 2006).

### 3.2.2 Main Results

In this section, we prove that under the aforementioned model, we can download the file in the minimum time possible while utilizing the full capacity of angels.

**Lemma 1.** *In the presence of angels, under a fluid Uplink-Sharing model, a distribution time  $T$  for a file of size  $F$  is achievable, where:*

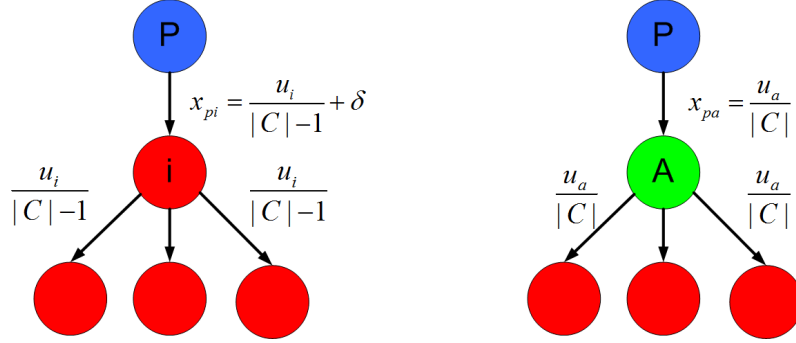
$$T = \frac{F}{\min\{u(P), d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}} \quad (3.6)$$

where  $d_{min} = \min_{i \in C} \{d(i)\}$

*Proof.* We provide a fluid construction that achieves the bound for  $T$ . This construction follows three cases corresponding to the nature of the underlying bottleneck (the denominator of  $T$ ).

**Case A:** When the upstream capacity of the content provider  $P$  is the bottleneck, *i.e.*,  $u(P) \leq \min\{d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}$ ,  $P$  sends fresh data to each client  $i$  with rate  $x_{pi}$ , and also sends fresh data to the angel with rate  $x_{pa}$ . Each client forwards data it receives from the provider

to the other  $|C| - 1$  clients. Similarly, the angel forwards data it receives from the provider to all  $|C|$  clients. Notice that once data is sent to one client, all other clients would successfully receive it from that client as long as  $x_{pi} = x_{ij} \forall j \in C$ . Figure 4.1 illustrates the idea. To complete



**Figure 3.1:** Clients/angel forward content received from provider to other clients.

our construction, we show that there are values for  $x_{pi}$  and  $x_{pa}$  that ensure that all clients will successfully download the data with rates  $y_i$  equal to the provider's capacity, *i.e.*,  $y_i = u(P)$ ,  $\forall i \in C$ , *without violating the upstream/downstream capacity constraints of the various nodes*. To that end, consider the following rates:

$$x_{pi} = (1 - \delta) \frac{u_i}{|C| - 1}$$

$$x_{pa} = (1 - \delta) \frac{u_a}{|C|}$$

where

$$\delta = \frac{\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|} - u(P)}{\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}}$$

These rates respect the five aforementioned constraints and achieve the desired MDT. It is straightforward to verify that the choice of these rates ensures that the clients and the angel will not exceed their upstream capacities (constraints 2 and 4), and that the total data sending rate of the provider

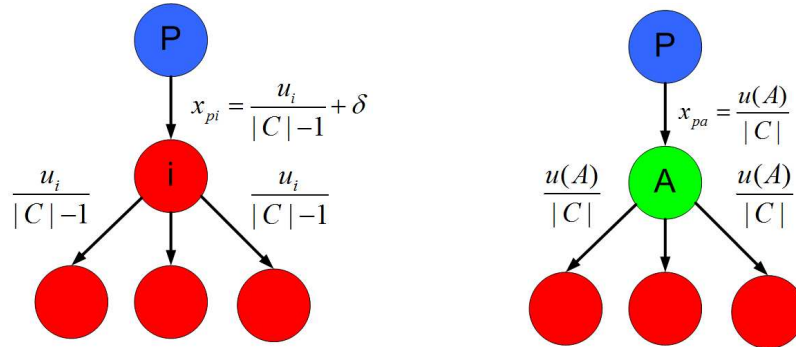
$x_p$  (given below) satisfies the seeder's capacity constraint (constraint 1).

$$\begin{aligned}
 x_p &= \sum_{i \in C, A} x_{pi} = (1 - \delta) \left( \frac{u(C)}{|C| - 1} + \frac{u(A)}{|C|} \right) \\
 &= \frac{u(P)}{\frac{u(C)}{|C| - 1} + \frac{u(A)}{|C|}} * \frac{u(C)}{|C| - 1} + \frac{u(A)}{|C|} \\
 &= u(P)
 \end{aligned}$$

**Case B:** The clients's share of the aggregate upstream capacity is the bottleneck, *i.e.*,  $\frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2} \leq \min\{u(P), d_{min}\}$ . Our construction in this case requires a new dissemination strategy, which goes in two phases as outlined below.

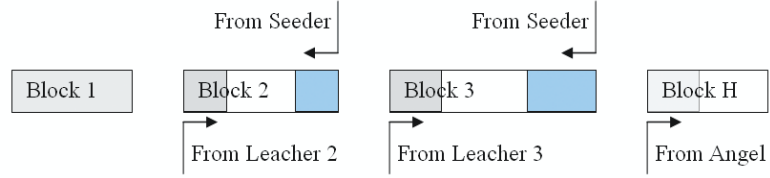
*Phase 1:* The provider divides the file into  $|C| + 1$  unequal size blocks proportional to the upstream capacities of the  $|C|$  clients and of the angel. Next, the provider sends block  $i$  to the  $i$ -th client and block  $i = |C| + 1$  to the angel. Unlike in Case A, due to its upstream capacity constraint, a client is not able to disseminate *all* the data it receives, from the provider, to the other clients. Hence clients can only share parts of their blocks with each other, *i.e.*,  $x_{pi} \geq x_{ij} \quad \forall j \in C$ . Let  $W$  denote the duration of phase 1. The block sizes are chosen to guarantee that at time  $W$  each client would have completed the download of its corresponding block from the provider and only parts of the other clients' blocks.

Fig.3-2 illustrates this dissemination strategy, where it is clear that the client's download rate from the provider differs from its upload rate to the other clients.



**Figure 3-2:** During phase one, the provider sends each client its perspective block which in its turn forwards part of it to the other clients.

*Phase 2:* In this phase, *i.e.*, for  $t > W$ , the clients continue to send the data they got from the provider in Phase 1 to all other clients, while the provider uses its upstream capacity to help out with the dissemination of incomplete blocks. Figure 3-3 illustrates the state of file reception for client 1 during Phase 2.



**Figure 3-3:** The state of the file reception after time  $W$  for client 1

To complete our construction, we propose the following rate allocation scheme, which we show to match the distribution time  $T$  in Eq.2, while satisfying imposed capacity constraints. To that end, consider the following assignments:

$$\begin{aligned}
 x_{pi} &= \frac{u_i}{|C| - 1} + \delta \quad \forall i \in C \\
 x_{pa} &= \frac{u(A)}{|C|} \\
 x_{ij} &= \frac{u_i}{|C| - 1} \quad \forall i \in C, i \neq j
 \end{aligned} \tag{3.7}$$

Setting  $x_{ij} = \frac{u_i}{|C| - 1}$  guarantees that clients will not violate their upstream capacity since  $x_i = (|C| - 1) * \sum_{j \in L} x_{ij} = u_i$  (constraint 2). Likewise the angels (constraint 4). Assuming that the angel's downstream capacity is greater than or equal to  $1/|C|$  of its upstream capacity ensures the satisfaction of constraint 5. Choosing  $\delta = \frac{u(P)}{|C|} - (\frac{u(C)}{|C|(|C|-1)} + \frac{u(A)}{|C|^2})$  guarantees that  $\delta \geq 0$  as  $u(P) \geq \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}$ .

Substituting for  $\delta$ , the aggregate rate by which the provider sends data  $x_p = \sum_{i \in C} x_{pi} + x_{pa} = \frac{u(C)}{|C|-1} + |C|\delta + \frac{u(A)}{|C|}$  is given by:  $x_p = \frac{u(C)}{|C|-1} + |C|(\frac{u(P)}{|C|} - (\frac{u(C)}{|C|(|C|-1)} + \frac{u(A)}{|C|^2})) + \frac{u(A)}{|C|} = u(P)$ , thus satisfying constraint 1.

Each client  $i$  will download with rate  $y_i = x_{pi} + \sum_{j \neq i \in C} \frac{u_j}{|C|-1} + x_{pa} = \frac{u(C)}{|C|-1} + \frac{u(P)}{|C|} -$

$(\frac{u(C)}{|C|(|C|-1)} + \frac{u(A)}{|C|^2}) + \frac{u(A)}{|C|} = \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}$ , thus satisfying constraint 3.

To summarize, when the network upstream capacity is the bottleneck, we showed that a rate  $y_i = \min\{u(P), d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\} \forall i \in C$  is achievable while satisfying the various imposed capacity constraints.

**Case C:** The downstream capacity of the slowest client is the bottleneck,

*i.e.*,  $d_{min} \leq \min\{u(P), \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}$ .

We consider two subcases, for which we provide a proof sketch below.

- Subcase C1:  $d_{min} \leq \frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}$
- Subcase C2:  $d_{min} > \frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}$

Proving achievability for Subcase C1 is similar to case A. We use the same fluid model of Fig. 4-1, except for a change in the value of  $\delta = \frac{\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|} - d_{min}}{\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}}$ . Similar to Case A, setting  $x_{pi} = (1 - \delta)\frac{u_i}{|C|-1}$  and  $x_{pa} = (1 - \delta)\frac{u(A)}{|L|}$  and making nodes forwarding received data to the other clients satisfies constraints 2 and 4. The amount of data the provider sends equals:

$$x_p = \sum_{\forall i \in C, A} x_{pi} = \sum_{\forall i \in C} ((1-\delta)\frac{u_i}{|C|-1}) + (1-\delta)\frac{u(A)}{|C|} = (1-\delta)(\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}) = d_{min} \leq u(P)$$

satisfying constraint 1.

The download rate of any client will be:

$$y_i = x_{pi} + \sum_{\forall j \in C, A \& j \neq i} x_{ji} = \sum_{\forall j \in C, A} x_{pj} = d_{min}$$

satisfying the distribution time  $T$  in Eq.2 and constraint 3.

To satisfy constraint 5, the angel downstream capacity must exceed a tiny fraction of the slowest client downstream capacity  $d_{min} \frac{\frac{u(A)}{|C|}}{\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}} \leq d_a$ . Thus these rates satisfies the 5 constraints and the desired client download rate.

Similarly, proving achievability for Subcase C2 is similar to case B, except for a change in the value of  $\delta = d_{min} - (\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|})$ . Again, constraints 2 and 4 follow through from the assigned rates in equations 3.7 because  $\delta \geq 0$ . Also, we assume that  $d_a \geq x_{pa} = \frac{u(A)}{|C|}$  which is a fair assumption to satisfy constraint 5.

The rate of data send from the provider equals:

$$\begin{aligned}
\sum_{\forall i \in C} x_{pi} + x_{pa} &= \frac{u(C)}{|C|-1} + |C|\delta + \frac{u(A)}{|C|} \\
&= \frac{u(C)}{|C|-1} + |C|d_{min} - |C|\left(\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}\right) + \frac{u(A)}{|C|} \\
&= |C|d_{min} - (u(C) + u(A)) + \frac{u(A)}{|C|}
\end{aligned}$$

But in this case  $d_{min}$  is the bottleneck, thus:

$$\begin{aligned}
\frac{u(P) + u(C) + u(A)}{|C|} - \frac{u(A)}{|C|^2} &\geq d_{min} \\
u(C) + u(P) + u(A) &\geq |C|d_{min} + \frac{u(A)}{|C|} \\
u(P) &\geq |C|d_{min} - (u(C) + u(A)) + \frac{u(A)}{|C|} \\
u(P) &\geq \sum_{\forall i \in C} x_{pi} + x_{pa} \\
&= x_p
\end{aligned}$$

Thus the constraint 1 is satisfied.

Now, lets check the rate of download of any client;  $y_i$ :

$$\begin{aligned}
y_i &= x_{pi} + \sum_{\forall j \neq i \in C} \frac{u_j}{|C|-1} + x_{pa} \\
&= \frac{u(C)}{|C|-1} + d_{min} - \left(\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}\right) + \frac{u(A)}{|C|} \\
&= d_{min}
\end{aligned}$$

Thus each client downloads with rate  $d_{min}$  satisfying constraint 3 and Equation 2.

Following the same steps in the proof for cases  $A$  and  $B$ , one can show that the distribution time  $T$  in Eq.2 is achievable for subcases  $C1$  and  $C2$ , respectively.  $\square$

**Lemma 2.** *In the presence of angels, under a fluid Uplink-Sharing model, the minimum distribution time,  $T_{min}$ , has a lower bound given by:*

$$T_{min} \geq \frac{F}{\min\{u(P), d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}} \quad (3.8)$$

*Proof.* The bound given on MDT implies that the download rate of any client,  $y_{max}$  is bounded as follows:  $y_{max} \leq \min\{u(P), d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}$ . This implies that the MDT bottleneck is due to one of three possibilities. The first is the upstream capacity of the provider:  $y_{max} \leq u(P)$ . In this case the provider's upstream capacity limits the minimum download rate of peers (*e.g.*, when clients are powerful and the provider is weak). The second possibility is the downstream capacity of the slowest client:  $y_{max} \leq d_{min}$ . Clearly the minimum download rate cannot exceed the minimum downstream capacity of any client. The third possibility is the aggregate upstream capacity of the swarm:  $y_{max} \leq \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}$ . In this case the aggregate upstream capacity of the swarm cannot saturate the downstream capacity of the clients.

Only the third of these cases is not obvious. First, we start with the case of no angels. We know that the aggregate download rate cannot exceed the aggregate upstream capacity in the swarm:  $\sum_{i \in C} y_i \leq u(P) + u(C)$ . To utilize a fraction  $z_a$  of the upstream capacity of the angel, the angel must download fresh data with a rate of at least  $\frac{z_a}{|C|}$ . Thus, in case of using angels, we get:  $\sum_{i \in C} y_i + \frac{z_a}{|C|} \leq u(P) + u(C) + z(A) \frac{\sum_{i \in C} y_i}{|C|} \leq \frac{u(P)+u(C)+z(A)}{|C|} - \frac{z(A)}{|C|^2}$

Since the minimum is always less than the mean and the upper bound of  $y_{max}$  is achieved when  $z(A) = U(A)$ , we conclude that  $y_{max} \leq \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}$ .  $\square$

**Theorem 1.** *The minimum distribution time,  $T_{min}$ , given in Lemma 2 is tight.*

*Proof.* The proof follows directly from the fact that the construction given in Lemma 1 achieves the lower bound stated in Lemma 2.  $\square$

### 3.2.3 Implications on the Role of Angels

It follows directly from the aforementioned theoretical results that when the bottleneck lies in the aggregate upstream capacity of the swarm and not in the source (provider's upstream capacity) or the sink (slowest client downstream capacity), adding upstream capacity to the swarm through the

use of angels will necessarily improve MDT. But, can angels achieve these gains if they behave just like normal clients (*i.e.*, download the entire file)? The answer is *no*. If angels download more than  $\frac{1}{|C|}$  of what they upload, they will not be able to upload the excess amount to clients. It would have been more beneficial to upload this excess data to the clients directly from the provider. For example, our optimal construction can speed up download rate to  $(\frac{u(A)}{|C|} - \frac{u(A)}{|C|^2})$ . If angels act like clients (and have capacities identical to these of clients) the download rate would be  $\frac{u(P)+u(C)+u(A)}{|C|+|A|}$ . This roughly equals  $\frac{u(P)+u(C)}{|C|}$ , which is identical to not having angels in the first place. Thus, adding angels that behave just like clients (by downloading the content in its entirety) only enlarges the swarm but does not optimize MDT. Even if the added angels' upstream capacity is made (say) double that of the clients' upstream capacity, the speed up from our construction would be double the speed up achieved by having angels act like clients.

To summarize, merely adding “participants” to a swarm will not result in lowering MDT, unless these participants behave in the specific manner prescribed in our construction. This is precisely the role we propose for angels: Angels are on-demand swarm participants that do not have and do not seek to obtain the content (they are not replicas or caches of the provider); they download just the right amount of the content (as prescribed in our construction) to minimize MDT.

### 3.3 Practical MDT Construction

The optimal fluid construction presented in the previous section makes two assumptions: (1) data dissemination is fluid instead of packetized, and (2) a client is able to open and use an arbitrary large number of concurrent connections, as large as the number of clients in the swarm. As we experimentally show later on (see section 3.4), a direct implementation of our fluid construction results in a very degraded performance due to the realities of OS and network stack implementations (*e.g.*, given the packet-based nature of transport protocols, and given the fact that setting up an unbounded number of connections leads to bandwidth fragmentation and TCP timeouts).

That said, our optimal MDT construction gives us two important insights that are crucial for building a near-optimal bulk-synchronous content distribution strategy: (1) the upstream capacity

of clients is the most scarce asset in the system, thus utilizing the clients upstream capacity fully and as soon as possible is key, (2) angels can utilize their upstream capacity fully to forward data even if they download data with rates significantly smaller than their downstream capacity.<sup>4</sup>

### 3.3.1 The Group Tree Coordinated Swarming Strategy

Taking into consideration the pros and cons of an optimal fluid construction, we build a coordinated swarming strategy that does not require any client to have more than  $k$  connections at any point in time. Our *Group Tree* (GT) strategy works in two phases as depicted in Figure 3.4. The first phase utilizes a tree-like structure with the aim of getting all nodes (angels and clients) to download, and hence upload, content as soon as possible. In the second phase, nodes swarm together to finish downloading the file. In the remainder of this section, we provide the details of these two phases of the GT strategy.

Initially, the provider (seeder) divides the file into  $k$  segments, where  $k$  is the upper-bound set on the out-degree of (number of connections allowed for) clients – a parameter of the GT strategy.

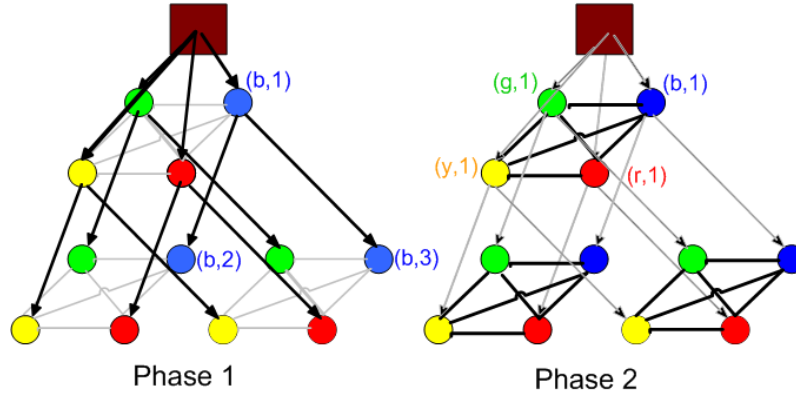
In Phase 1, the swarm is organized into  $k$  binary trees. One of the trees is dedicated to angels, whereas the other trees are populated by clients. Clients are matched up and assigned to trees in such a way that all tree participants have similar upstream capacities. Each tree is assigned a segment of the file that is proportional in its size to the nominal, individual upstream capacity of nodes (clients or angels) in the tree. The seeder sends the segments in chunks to the root of each tree, noting that the tree nodes operate in a pipelined fashion: Once a client/angel receives a packet, it forwards it to its children in the tree. For example, as depicted in Figure 3.4, client  $(b, j)$  downloads the segment  $b$  of the file from client  $(b, \lfloor \frac{j}{2} \rfloor)$ . It only takes  $\log(N)$  multiples of a packet transfer time for all clients to begin utilizing their upstream capacity.

In Phase 2, sets of  $k$  nodes (from the  $k$  different trees) each having received one of the  $k$  different segments of the file, form a clique for swarming purposes. By construction, each clique would include one angel.<sup>5</sup> Each clique uses our optimal MDT construction to disseminate the

---

<sup>4</sup>Actually, any excess download by an angel underscores an inefficiency, as a client could have used this wasted capacity.

<sup>5</sup>For implementation purposes, if the number of servers supporting angel functionality is small, the servers can



**Figure 3-4:** Illustration of the GT strategy. In phase 1,  $k = 4$  different segments are disseminated downward through  $k = 4$  binary distribution trees. In phase 2, segments are disseminated laterally within cliques of size  $k = 4$ .

file between its members. For example, as depicted in Figure 3-4, clients  $(b, 1)$ ,  $(r, 1)$ ,  $(g, 1)$  and  $(y, 1)$  form a clique. Notice that in this phase, the operation of angels and clients is different. In particular, each angel sends data to clients in its clique *without* receiving any data from these clients, thus saving the precious upstream capacity of clients.

In the first phase of our GT strategy, the content provider uses  $k$  binary subtrees (*i.e.*, fan-out = 2) to distribute a given segment to a particular set of nodes. Theoretically, the optimal fan-out for distributing segments in Phase 1 is the natural number,  $e$ . Simulation results show that a fan-out of 2 achieves better performance than a fan-out of 3, justifying our choice of a binary tree construction for our GT strategy.

Proof: The depth of an  $m$ -ary tree, *i.e.*, a group tree of fan-out  $m$ , which has  $N$  clients is  $d = \lceil \log_m(\frac{N}{k} + 1) \rceil$ . The time needed to distribute a chunk of size  $F$  from a client with upstream capacity of  $c$  to  $m$  children will be  $\frac{F}{c} * m$ , which is linear in  $m$ . This happens because as we increase the fan-out each child's share of its parent's bandwidth will decrease.

$\therefore T$ , the time to disseminate  $\frac{1}{k}$  of the file to each client, will be:

---

time-multiplex their capacities between multiple cliques, effectively allowing on angel per clique.

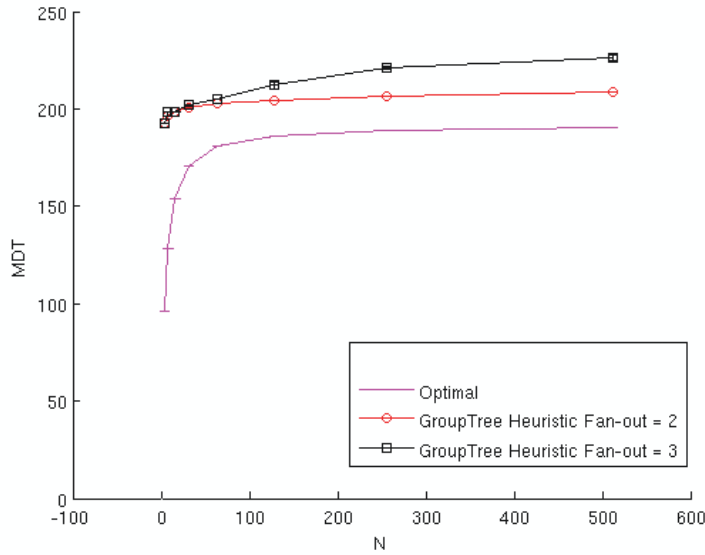
$$\begin{aligned}
T &= \frac{F}{c} * m * d \\
&= \frac{F}{c} * m * \lceil \log_m(\frac{N}{k} + 1) \rceil \\
T &\geq \frac{F}{c} * m * \log_m(\frac{N}{k} + 1) \\
&\geq \frac{F}{c} * \ln(\frac{N}{k} + 1) * (\frac{m}{\ln(m)})
\end{aligned}$$

The optimal value of  $m$  is when  $\frac{\partial T}{\partial m} = 0$

$$\begin{aligned}
\frac{\partial T}{\partial m} &= \frac{F}{c} * \ln(\frac{N}{k} + 1) * [(\frac{1}{\ln(m)}) + (m * \frac{-1}{\ln^2(m)} * \frac{1}{m})] \\
\text{At } \frac{\partial T}{\partial m} &= 0 \\
(\frac{1}{\ln(m^*)}) + (\frac{-1}{\ln^2(m^*)}) &= 0 \\
\ln(m^*) &= 1 \\
m^* &= e
\end{aligned}$$

Thus, the optimal fan-out of GT would be 2 or 3. We used our discrete time simulator to decide the optimal fan-out of the GT. Figure 3-5 compares the distribution time of a file size 192 MB to an exponentially increasing number of peers. The upstream and downstream capacities are similar to the ones used in section 3.4. The graph shows that a fan-out of 2 will be better than a fan-out of 3. From now on, we use a fan-out of 2 whenever we use GT.

In contrast to the *Recursive MDT* heuristic, using GT activates all nodes after  $t = T_{PKT} * \log_{k+1} |L|$  during phase 1, where  $T_{PKT}$  is the time to transfer one packet. The node activation time using GT is thus faster than that of the *Recursive MDT* heuristic by a factor equal to the ratio between the packet size and the file size, thus dominating in performance as the system will utilize its aggregate upstream capacity much faster.



**Figure 3-5:** A comparison between GT with fan-out 2 and GT with fan-out 3.

### 3.3.2 Other Distribution Strategies

In addition to our GT strategy, which is rooted in our MDT construction, we have considered a host of other practical distribution heuristics that rely on peer and piece selection strategies to assess their appropriateness for achieving the MDT objective.

The most basic (baseline) strategy we considered is *Random*, whereby the sender chooses a receiver at random, and sends a random piece to it given that this receiver needs this piece. This strategy does not employ any intelligence in peer or piece selection. That said, in our implementation we ensured that no sender remains idle as long as a receiver needs a piece that the sender has. This strategy fully utilizes the resources in the system but without coordination.

To evaluate the benefit from smarter piece selection strategies, we implemented a *Local Rarest First* (LRF) piece selection strategy (Legout et al., 2006), which sends the piece with the least number of copies that the receiver’s neighbors have. LRF tries to achieve balanced piece distribution depending on local information. In addition to LRF, we also implemented a *Global Rarest First* (GRF) piece selection strategy. While impractical at scale, GRF allows us to evaluate the full

benefit of optimizing piece selection depending on full piece distribution information. To evaluate the benefit of peer selection, we implemented a *Fair Peer Selection* strategy (FPS), which tries to get each client a fair-share of the network’s upstream capacity. A sender chooses the least fortunate recipient to send data to, *e.g.*, the client with the least fan-in given that all the clients have the same upstream capacity. Finally, we implemented a strategy that combines piece and peer selection. The FPS+GRF strategy chooses the most needy recipient, and then sends the GRF piece to it. This strategy can be seen as independently supporting intelligent piece and peer selection strategies.

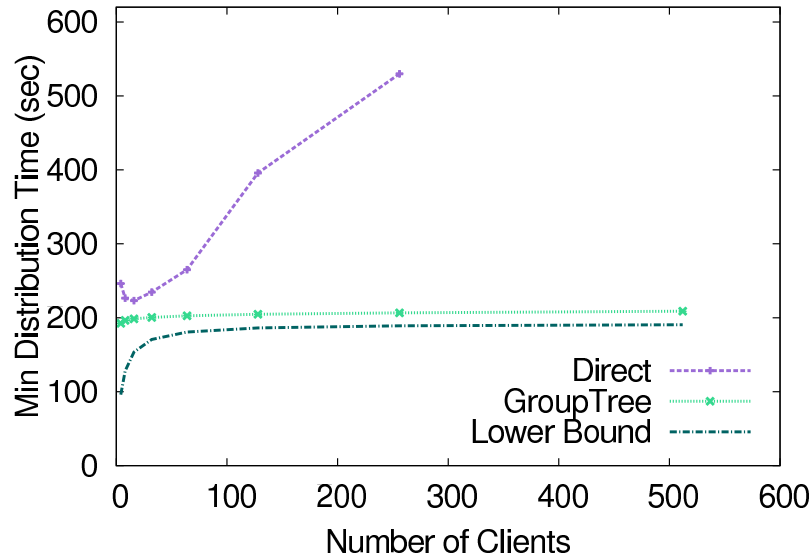
### 3.4 Simulation Results

In this section we present simulation results comparing our GT strategy to the other MDT-agnostic strategies. We built a custom discrete-event simulator that simulates a provider, a set of angels, and a set of clients downloading a file consisting of a set of blocks. Each node builds a set of connections to some (or all) of the other nodes. Each connection has a queue of blocks to be transferred sequentially over that connection (the delivery of one block marks the beginning of the transfer of the next block). Upon receipt of a block, a node decides whether or not it should forward that block to other peers. A connection is terminated as a result of one of two events: either the expiration of a randomized timeout parameter, or when there are no more blocks to transmit over the connection. Upon the termination of a connection, a node establishes a new connection possibly to a new peer, if necessary. Our simulation is done at the session layer, thus ignoring transport layer effects, *e.g.*, due to packet loss or cross traffic, which we captured at the session level by introducing variability in the connection speed over time.

In our experiments, we use the following model parameters: the file size  $F$  is set to 24 MB (with a block size of 128KB); the provider upstream capacity  $u_p$  is set to 512 Kbps; the client/angel upstream capacity  $u_i$  is set to 128 Kbps and the downstream capacity  $d_i$  is set to 256 Kbps,  $\forall i \in C, A$ . These parameters ensure that the aggregate upstream capacity of the swarm (without the angels) is the bottleneck, which is the scenario of interest as explained in section 4.2.3.

In our first set of experiments, we evaluate the performance of a direct implementation of an optimal MDT construction as the number of clients scales up, and compare the achievable

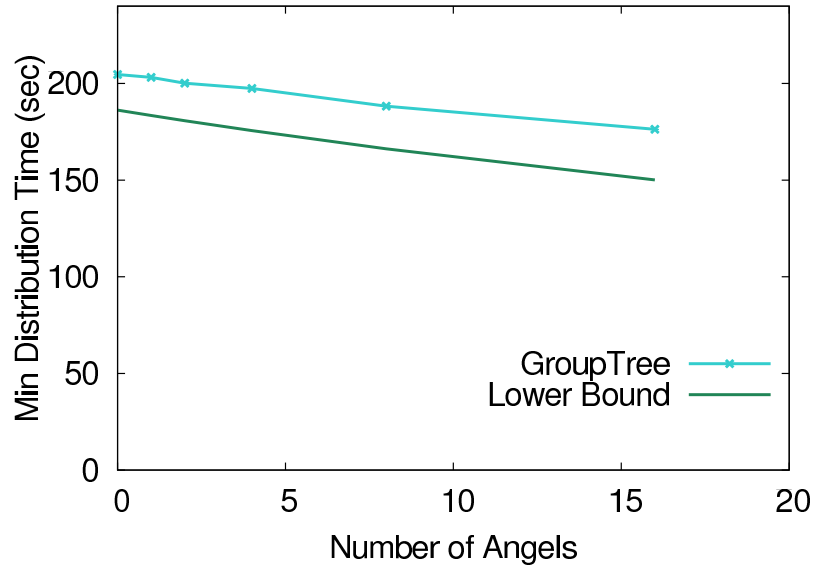
distribution time to the theoretical lower bound (predicted by our fluid model). Figure 3-6 shows that a direct implementation of the MDT construction does not scale. This is because the optimal construction assumes a fluid communication model, implying that a client can forward data the instant it receives it. In practice, clients need to transfer data in blocks. This means that as the number of clients in the swarm increases, the speed of each connection decreases correspondingly (since the client bandwidth is equally divided across all connections), causing the time to transfer a block to increase linearly. The figure also shows the results when the GT strategy is used under the same settings, showing near optimal MDT and scaling characteristics.



**Figure 3-6:** Performance of a direct implementation of the MDT construction and of the GT strategy (also showing the analytical MDT lower bound).

In our second set of experiments we fixed the number of clients to 128 and varied the number of angels between 0 and 16, to examine the potential improvement in MDT due to angels. Figure 3-7 shows the performance of the GT strategy versus the theoretical lower bound, which makes it evident that the performance of the GT strategy tracks that of the MDT lower bound as the number of angels increases, subject to a relatively fixed overhead for carrying out Phase 1 of the GT.

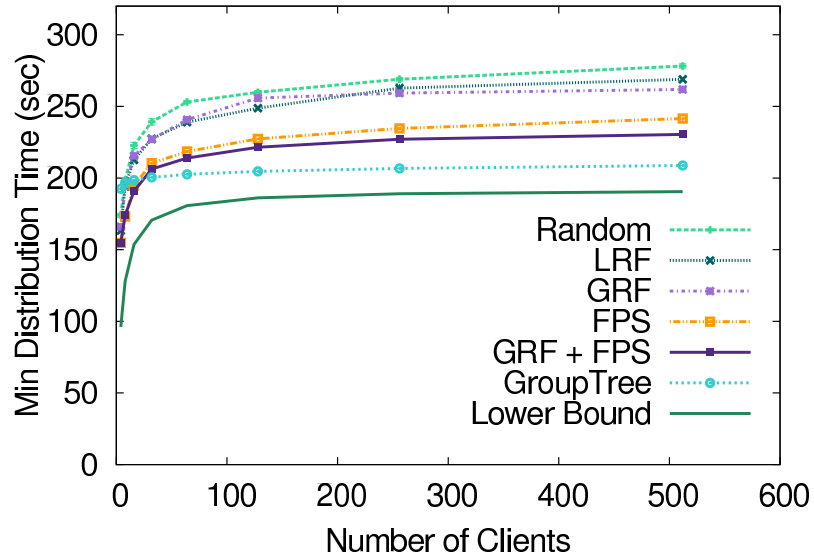
In our third set of experiments, we compared the performance of the GT strategy to these of the other distribution strategies presented in section 3.3.2, as the number of clients varies from 4 to 512.



**Figure 3-7:** Performance of GT as a function of the number of angels.

Figure 3-8 shows the results (as well as the theoretical MDT lower bound). These results indicate that our GT strategy outperforms all others and that it operates within striking distance of the optimal MDT bound. Another observation from this set of experiments is that the performance of the GRF+FPS strategy (utilizing both piece and peer selection, albeit independently) is decidedly superior to the stand-alone GRF and FPS strategies.

Minimizing the worst-case distribution time may result in a degradation of the average distribution time, especially in settings wherein the client downstream capacities are highly diverse, *e.g.*, a setting where a few clients have much lower (outlier) download speeds than most others. In such settings, aiming to reduce MDT will necessarily hurt the average distribution time. We argue that bulk-synchronous content distribution applications are unlikely to involve such disparate profiles for client upstream/downstream capacities. Rather, for our purposes, the set of clients involved in a bulk-synchronous download are likely to have comparable capabilities (*e.g.*, a set of broadband residential users in a virtual-reality gaming application, or a set of enterprise servers acting as consistent data repositories). In such settings, our experiments confirmed that optimizing MDT does indeed improve the average distribution time as well. Figure 3-9 shows the *average* distribution



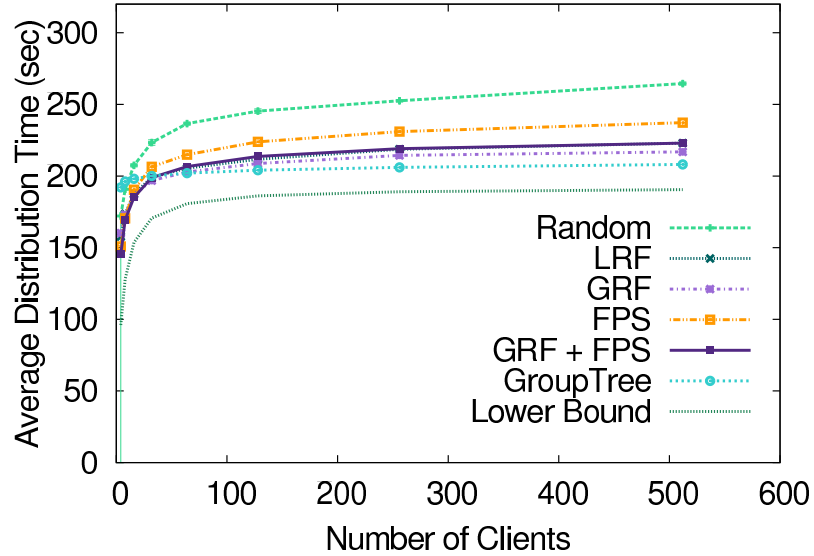
**Figure 3-8:** MDT for various strategies as a function of the number of clients.

time for the swarm considered in the last experiment; it shows that our GT strategy, which while primarily aiming to optimize MDT, outperforms the other strategies with respect to the average distribution time as well. This hints to the utility of coordinated swarming beyond MDT, but is not a focus of this work.

To summarize, our simulation experiments show that the GT strategy outperforms all other strategies, scales well, and operates near the optimal, theoretical MDT bound. Our findings also suggest that coordinated swarming strategies, which coordinate both piece and peer selection, such as GT result in much better performance than strategies that focus on either piece or peer selection (*e.g.*, the LRF, GRF, and FPS strategies) or these that consider both but in an uncoordinated fashion (*e.g.*, the FPS+GRF strategy).

### 3.5 CloudAngels: Blueprint for System Design

In this section we present the design and implementation of a cloud service that enables a content provider to utilize cloud resources to complement a peer-assisted content distribution system for the purpose of minimizing content distribution time. Our service, which we call CLOUDANGELS,



**Figure 3-9:** Average distribution time as a function of the number of clients.

is based on the GT strategy presented in section 3.3. We believe that a cloud offering is the most appropriate paradigm for an implementation of CLOUDANGELS because the service needs to be able to instantiate angels on-the-fly: As content providers seek assistance with MDT content delivery, the system would need to instantiate a set of Virtual Machines (VMs) to act as angels. The aggregate capacity of the deployed angels would depend on the content provider’s stated objectives – *e.g.*, how much upstream bandwidth to “add” to the swarm to achieve an optimal MDT, or to ensure that MDT is below a certain preset threshold.

We identify the following agents as the essential players in our system:

**Provider:** ( $P$ ) This is the entity in need of the CLOUDANGELS service’s help with the distribution of content (namely, a file within its possession) to a group of clients engaged in a bulk-synchronous application.

**Clients:** ( $C_1, C_2, \dots, C_k$ ) These are the agents constituting the swarm interested in the bulk-synchronous delivery of content from the provider. We assume that the provider relays to the CLOUDANGELS service the list of clients authorized to access the content.

**Angels:** ( $A_1, A_2, \dots, A_j$ ) These are the agents (VMs) that are created on-the-fly to help the con-

tent provider distribute the content subject to MDT objectives.

Figure 1-1 illustrates the design of our CLOUDANGELS system, which consists of two major components: (1) the Auxiliary Services Component (ASC) which acts within the control plane to choreograph the GT strategy, and (2) the Content Distribution Component (CDC) which acts within the data plane to distribute content to clients using the GT strategy.

In support of CDC, the ASC provides two sets of services denoted in Figure 1-1 by the “*Profiler*” and the “*Accountant*”. The CLOUDANGELS profiler is responsible for the collection of information about each client’s upstream and downstream capacity. Clients may either self-report their capacity<sup>6</sup> or else, the profiler may use other methods to estimate their upstream/downstream capacities. The CLOUDANGELS accountant is responsible for identifying the level of assistance (in terms of the number and capacity of angels, if any) that the bulk-synchronous content distribution group would need to meet its MDT objectives. This objective may simply be to minimize MDT, or alternately, it may be to ensure that MDT remains below a preset threshold (for a minimal quality of service dictated by the bulk-synchronous application). To do so, the accountant estimates the distribution time achievable without the help of angels, and based on the desired MDT objective, calculates the characteristics (number and capacities) of the angels that need to be deployed to meet this objective. The accountant keeps track of these aspects for purposes of charging the provider for the service.

The CDC supports the main CLOUDANGELS content distribution functionality as follows. Upon receipt of a request from the content provider to distribute a file, the CDC deploys a “*Registrar*” process to manage the request. The registrar does not get involved in the data transfer but is responsible for managing the bulk-synchronous swarm. According to the GT swarming strategy, the registrar assigns identifiers to clients to choreograph their connection to one another. The registrar issues each client a signed token allowing it to introduce itself as an authorized client in the swarm. This token includes the file name and a computed unique identifier (ID) for the client in the swarm. The client ID is in the form of a tuple  $(i, j)$  where  $i$  is the segment that the client needs

---

<sup>6</sup>We note that for bulk-synchronous applications, the MDT objective is both a rational/selfish objective for the individual clients, as well as a socially-optimal objective for the swarm, and thus clients are incentivized to truthfully report their upstream/downstream capacities.

to subscribe to in Phase 1 of the GT, and  $j$  is the location of the node in the distribution tree. The registrar determines this ID based on the upstream capacity of the client (among other possible information in support of additional objectives, *e.g.*, topological location for efficient intra-ISP swarming).

To distribute a new file  $x$ , the content provider divides the file into  $k$  segments  $(x_1, \dots, x_k)$  and starts  $k$  server processes to handle requests for each one of these segments – requests from the roots of each one of the  $k$  (binary) trees constituting the first phase of the GT strategy. When the root of tree  $i$  sends its signed token to the content provider, the appropriate process serves segment  $x_i$  in 64KB-block increments, eventually resulting in the  $k$  different segments being transmitted through the roots of the  $k$  distribution trees.

In the first phase of the GT strategy, a client with ID  $(i, j)$  needs to download segment  $x_i$ . To do so, the registrar sends the client the IP address of its parent in the binary tree – namely,  $(i, \lfloor \frac{j}{2} \rfloor)$ . The client then begins its download, and starts a server process that listens for upload requests from the client’s children in the tree (during the first GT phase) and from the client’s siblings in the clique (during the second GT phase). Every time a child or a sibling contacts the server, it forks a thread and starts uploading segment  $x_i$ , giving priority to requests for  $x_i$  from its children over its siblings.

In the second phase of the GT strategy, the client with ID  $(i, j)$  needs to download all the other segments  $x_l$   $l \neq i$ . For segment  $l$ , the registrar sends the client the IP address of its sibling in the  $l$  tree. Client  $(i, j)$  contacts its siblings to download the other segments as soon as these segments are available at the siblings.

To meet the MDT objective requested by the provider, the registrar instantiates (on the fly) the necessary number of appropriately provisioned angels (as prescribed by our MDT construction). Each angel is assigned an ID  $(a, j)$ , where  $a$  corresponds to segment  $x_a$  of the file that the angels are responsible to disseminate (recall, the size of  $x_a$  is a function of the MDT construction). In the first GT phase, angels act like clients, disseminating segment  $x_a$  among themselves using the binary tree. In the second GT phase, angels run a server listener and serve their clique siblings segment  $x_a$ , but unlike clients they do not request or download any of the other segments of the

file.

### 3.6 CloudAngels: Experimental Evaluation

In this section we present results from extensive experimental evaluations of our CLOUDANGELS service. Our main motivation in performing these experiments was to (1) establish confidence in the performance of our implementation by comparing its average performance to that of widely used swarming solutions, (2) establish the effectiveness of deploying angel in the cloud for the purpose of meeting MDT objectives, and (3) provide evidence of the scalability characteristics of our system.

**Experimental Setup:** We used Emulab (Emu, 2010) to acquire virtual nodes for our experimental needs. To marginalize hardware and operating system interference, we requested configurations in which all nodes were identical. Subject to this requirement, we were able to get a total of 36 Emulab nodes (all running on 64-bit Xeon processors with 2GB of RAM, with a 64-bit Fedora 8 Linux Distribution operating system). This allowed us to evaluate GT swarms organized in binary trees of depths 1, 2, 3, and 4.<sup>7</sup> To introduce variance among nodes in terms of their upstream and downstream capacities, we used Linux packet filters *tc*.

In our experiments, we divided the set of Emulab nodes under our control into three groups (for purposes of the first GT phase) with the nodes in each group assigned an upstream capacity of 5mbps, 6mbps, and 7mbps, and a downstream capacity of 15mbps, 18mbps, and 21mbps, respectively. This assignment is reasonable given the typical (factor of two to three) mismatch between broadband upstream and download speeds. In addition to these three groups of clients, a fourth group of nodes was set aside to serve as angels, with each angel node provisioned with a downstream capacity of 24mbps and an upstream capacity that we set in our experiments to one of the following values: 0, 6, 12, 18, and 24mbps (based on the MDT needs of the swarm).

For comparative purposes, we have also configured our Emulab nodes (subject to the same upstream and downstream provisions) to run an unmodified BitTorrent (BT) client to allow us to

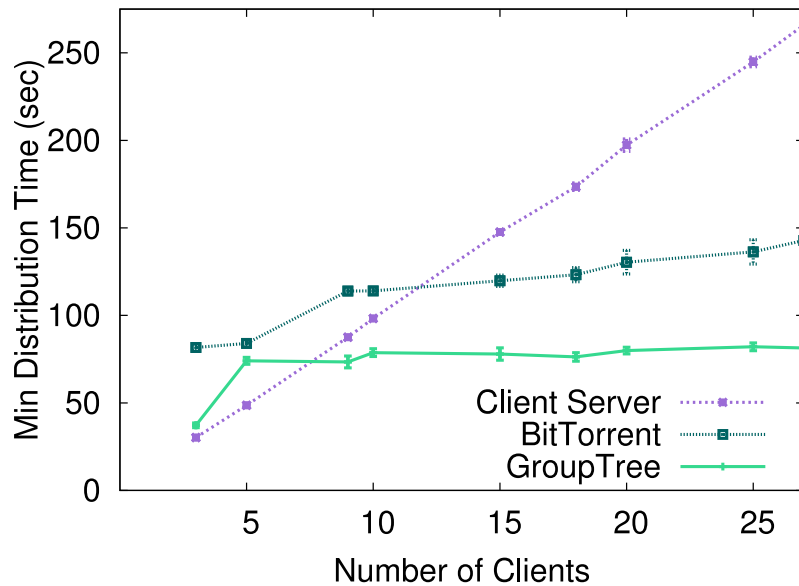
---

<sup>7</sup>A GT swarm with an out-degree bound of  $k$  organized in *full* binary trees of depth  $d$  could be as large as  $k * (2^d - 1)$ . With 36 nodes, we are able to consider *full* GT topologies of depth three, and only partially-full GT topologies beyond that.

establish a baseline model for MDT performance (against which the advantage of CLOUDANGELS could be evaluated). The specific client we used is the Instrumented BitTorrent mainline client (version 4.0.2) (ins, 2010).

In our experiment, the content provider is represented using a single seeding node, whose upstream capacity was set to 27mbps, and the file size is 30MB.

**Experimental Results:** To establish a performance baseline, Figure 3-10 shows the MDT achievable when content is distributed (1) using a client-server model, (2) using the unmodified BT client, and (3) using our CLOUDANGELS service without deploying *any* angels (*i.e.*, simply coordinating the content distribution according to the GT strategy). As expected, increasing the number of clients participating in the swarm results in a linear increase of the MDT for a client-server distribution strategy, but results in a sub-linear increase of the MDT for swarming strategies (namely our CLOUDANGELS service and BT), underscoring the benefit from tapping the upstream capacities of the clients.



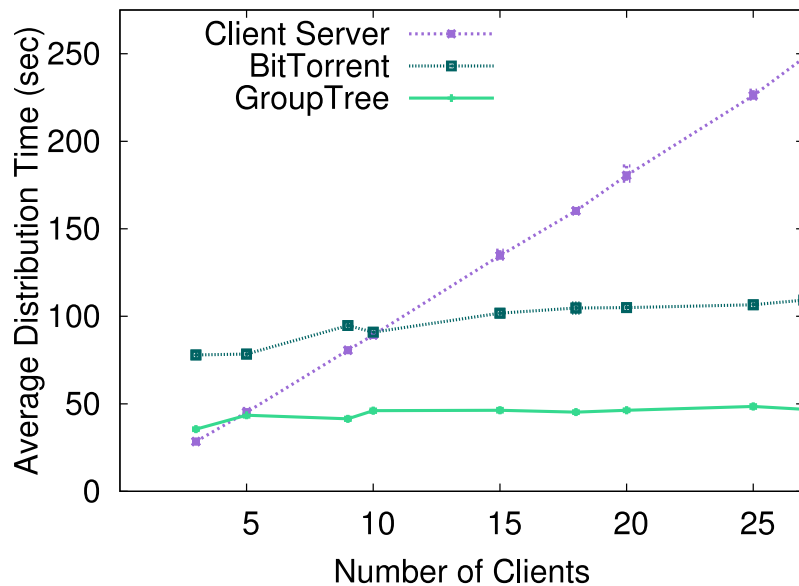
**Figure 3-10:** The relationship between MDT and the number of clients for swarming-based (CLOUDANGELS and BT) versus non-swarming-based (client-server) approaches.

Figure 3-10 also shows that the scaling characteristics of CLOUDANGELS (without angels) and

of BT are fairly similar in terms of their growth pattern, with the MDTs achievable by CLOUDANGELS consistently below those of BT. These results give us confidence that the implementation of our GT coordinated swarming strategy is efficient in the sense that it is competitive with (and indeed noticeably better than) vanilla BT swarming implementations. Needless to say, since in these experiments no angels were deployed, the lower MDT of CLOUDANGELS may be due to the superiority of the GT swarming strategy or it may be simply the result of a leaner implementation, which does not include all the bells and whistles (and hence overheads) associated with a BT client.

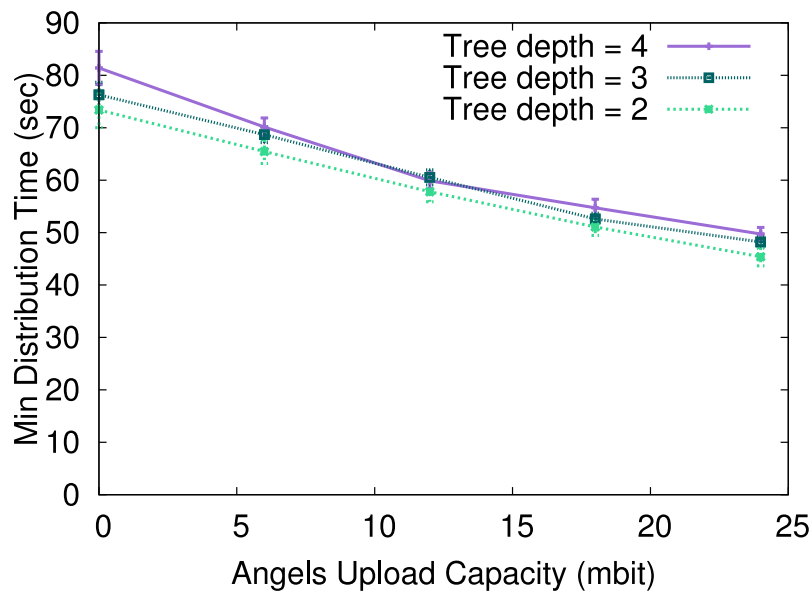
An important observation from the results in Figure 3-10 relate to the “step-wise” nature of the growth in CLOUDANGELS MDT as the number of clients is increased. This step-wise behavior is a direct result of the increase in the depth of the binary tree used in the first GT phase.

Figure 3-11 shows the average (as opposed to minimum) distribution time for the same experimental setup. It confirms the findings from our simulation experiments (see Figure 3-9) – namely, that the reduction in MDT does not come at the expense of average distribution time.



**Figure 3-11:** The average distribution time and the number of clients for swarming-based (CLOUDANGELS and BT) versus non-swarming-based (client-server) approaches.

We now proceed to presenting results in which angels were deployed by the CLOUDANGELS service. Figure 3-12 shows that as we increase the upstream capacity of angels the MDT decreases. Figure 3-13 shows the improvement in MDT as a result of deploying a fixed amount of angel capacity, as the number of clients increases, which results in an increase in the overall depth of our previously defined GT depth. Naturally, as the number of clients increases the effectiveness of CLOUDANGELS (with a fixed set of angels) in reducing MDT becomes increasingly limited.



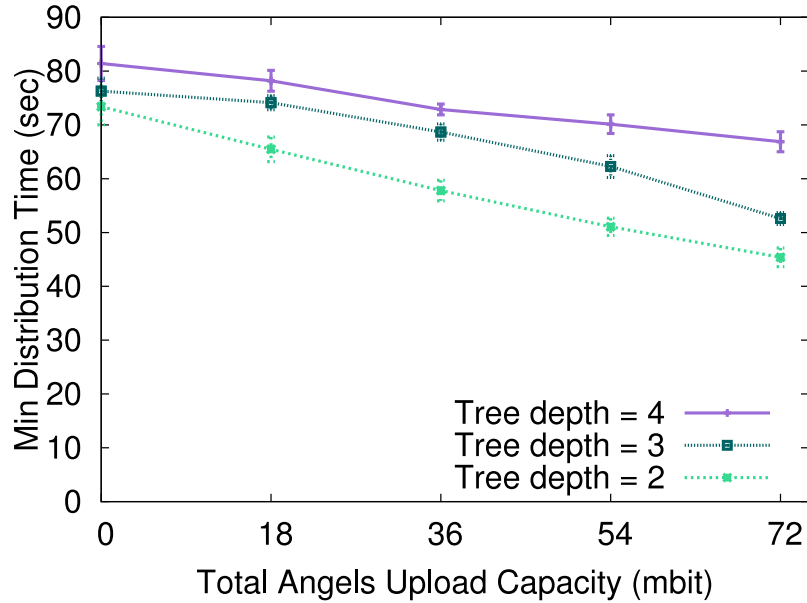
**Figure 3-12:** The impact of angel’s upstream capacity on MDT

### 3.7 Related Work

While not studied before, the bulk-synchronous content distribution problem and solutions we discuss in this chapter build on and leverage a vast body of prior work in a number of areas. We examine the most relevant of these next,<sup>8</sup> noting that we have referred to other related works throughout the chapter.

**MDT bounds and implications:** Kumar and Ross (Kumar and Ross, 2006) established MDT

<sup>8</sup>We note that we do not consider content delivery acceleration schemes – such as caching and replication – that may indirectly improve MDT to be relevant related work.



**Figure 3-13:** MDT as a function of number of clients for fixed angel capacity.

bounds for file dissemination from a seed to a set of clients, and identified three limiting conditions for MDT – namely the seeder, client, and network (swarm) bottlenecks. Based on the observation that the best operating regime for a swarm is when the seeder is not the bottleneck, the Antfarm system (Peterson and Sirer, 2009) directs its resources in support of swarms for which the seeder is the bottleneck. For swarm-assisted content distribution systems (such as those offered by major providers (aka, 2012; oct, 2012)), we argue that the seeder (provider) is likely to be well provisioned, making the network the likely MDT limiter, and providing the motivation for all our constructions and for the CLOUDANGELS prototype we presented in this chapter. While our theoretical MDT bounds (in the presence of angels) are similar to those independently derived by Kumar and Ross (Kumar and Ross, 2007), our fluid construction (to achieve MDT) is different. The fluid construction in (Kumar and Ross, 2007) separates clients into two tiers, with the MDT bound achieved only for the top tier. In our fluid construction, which is simpler, the MDT bound is achieved for *all* clients, making our construction more appropriate for bulk-synchronous content delivery.

**Overlays for optimized bulk-synchronous delivery:** Smaragdakis *et al.* (Smaragdakis et al., 2008) considered an alternative bulk-synchronous delivery framework, in which the content does not originate from a single source (the seed) but rather from the clients themselves. Thus, each bulk-synchronous delivery involves an exchange of content from every client to all other clients in the swarm. In that work, the focus is not on a coordinated swarming construction, but rather on the construction of an optimal topology for minimizing MDT.

**Cooperative swarming:** An inherent feature of bulk-synchronous swarms is that all clients share a common objective – that of minimizing MDT. Thus, unlike the “selfish” behaviors expected in file-sharing swarms (for example), clients participating in a bulk-synchronous content delivery swarm must be cooperative. There has been a relatively small number of studies that considered cooperative swarming (Smaragdakis et al., 2008), the most prominent of which is perhaps Bit-Tyrant (Piatek et al., 2007; Carra et al., 2008), which can be seen as a variant of BT that abandons the more rational tit-for-tat mechanism in favor of a mechanism that pushes the swarm to a more “socially optimal” operation, thus indirectly assuming that clients are cooperative.

**End-system multicast:** In essence, our GT strategy can be seen as an (asynchronous) end-system multicast (Chu et al., 2002; Banerjee et al., 2002; Castro et al., 2003) strategy. In that respect, the dHCPS system (Guo et al., 2008a) resembles GT in that it relies on a two-level hierarchical P2P dissemination mechanism for streaming, whereby each cluster has a Head (the source for this cluster) which is a member of a superNode swarm fed by the source. dHCPS solves the optimization problem for deciding how much upload bandwidth each head should dedicate to the superNode swarm versus its own group, whereas the main aspect of the GT strategy is the assignment of clients to trees and cliques and the determination of the segment sizes for each. Another end-system multicast system that resembles GT’s distribution strategy is ChunkySpread (Venkataraman et al., 2006), which divides a file into appropriately-sized chunks which are disseminated through different trees to avoid the overhead due to routing/ordering of individual chunks.

### 3.8 Summary of Contributions

In settings where bulk-synchronous content distribution is necessary, minimizing the maximum time it takes any client in a set to download content becomes the overarching goal. In this chapter, we have developed a provably optimal fluid construction that enables a content source (provider) to leverage the resources of helper nodes (angels) to meet specific minimum distribution time (MDT) objectives. Armed with insights gained from this fluid model, we proposed the *Group Tree* coordinated swarming strategy that forms the basis of a cloud service, CLOUDANGELS enabling content providers to leverage elastic cloud resources, on the fly, to meet their MDT objectives. We presented performance evaluation results that confirm the potential of our paradigm and system.

Our current work is focused on making the GT strategy more agile, by allowing dynamic adjustments to the assignment of segments to clients as a result of changes in network conditions, and on extending the CLOUDANGELS service to enable pipelined bulk-synchronous distribution of content from a single provider to a set of clients as well as to enable concurrent bulk-synchronous distribution of content from multiple providers.

## Chapter 4

# AngelCast: Peer-Assisted Live Streaming Using Optimized Multi-Tree Construction

Increasingly, commercial content providers (CPs) offer streaming and IPTV solutions that leverage an underlying peer-to-peer (P2P) stream distribution architecture. The use of P2P protocols promises significant scalability and cost savings by leveraging the local resources of clients – specifically, upstream capacity. A major limitation of P2P live streaming is that playout rates are constrained by the upstream capacities of clients, which are typically much lower than downstream capacities, thus limiting the quality of the delivered stream. Thus, to leverage P2P architectures without sacrificing the quality of the delivered stream, CPs must commit additional resources to complement those available through clients. In this chapter, we propose a cloud-based service – *AngelCast* – that enables CPs to elastically complement P2P streaming “as needed”. By subscribing to *AngelCast*, a CP is able to deploy extra resources (“angels”), on-demand from the cloud, to maintain a desirable stream (bit-rate) quality. Angels need not download the whole stream (they are not “leachers”), nor are they in possession of it (they are not “seeders”). Rather, angels only relay (download once and upload as many times as needed) the minimal possible fraction of the stream that is necessary to achieve the desirable stream quality, while maximally utilizing available client resources. We provide a lower bound on the minimum angel capacity needed to maintain a desired bit-rate to all clients, and develop a fluid model construction that achieves this lower bound. Realizing the limitations of the fluid model construction – namely, susceptibility to potentially arbitrary start-up delays and significant degradation due to churn – we present a practical multi-tree construction that captures the spirit of the optimal construction, while avoiding its limitations. In particular, our *AngelCast* protocol achieves near optimal performance (compared to the

fluid-model construction) while ensuring a low startup delay by maintaining a logarithmic-length path between any client and the provider, and while gracefully dealing with churn by adopting a flexible membership management approach. We present the blueprints of a prototype implementation of *AngelCast*, along with experimental results confirming the feasibility and performance potential of our *AngelCast* service when deployed on Emulab and PlanetLab.

#### 4.1 Motivation and Problem Statement

Streaming high-quality (HQ) video content over the Internet is becoming a standard expectation of clients, posing significantly different challenges for today’s content providers (CPs) such as Netflix, Hulu, or IPTV, compared to the challenges associated with the best-effort delivery of low-quality streaming through CPs such as YouTube and Facebook. For example, Netflix reported last year that it is delivering streams at rates between 1.4Mbps and 2.8Mbps (Net, 2010).

**Chapter Contributions:** In this chapter we propose a cloud-based “live stream-acceleration” service, *AngelCast*. By subscribing to *AngelCast*, a CP is assured that its clients would be able to download the stream at the desired rate without interruptions, while maximally leveraging the benefits from P2P delivery. *AngelCast* achieves this by (1) enlisting special servers from the cloud, called *angels* which can supplement the gap between the average client upstream capacity and the desirable stream bit-rate. Angels are more efficient than seeders as they do not download the whole stream, but rather they download only the minimum fraction of the stream that enables them to fully utilize their upstream capacity. In our architecture, the capacity that otherwise would have been wasted in downloading the full live stream to the servers can be channelled to help the clients directly; (2) choreographing the connectivity between nodes (clients and angels) to form optimized end-system multi-trees for peers to exchange stream content; and (3) handling clients dynamic arrival and departure.

We present theoretical results that establish the minimum amount of angel capacity needed to allow all clients to download at a desirable rate, as a function of their downstream/upstream capacities. We show that this lower bound is tight by choreographing the connectivity of nodes in such a way that the optimal bound is achieved under a fluid model.

A good live streaming system would also minimize the start-up delay needed to assure continued service. We prove that the start-up delay is zero under the fluid model, but that the optimal construction leads to a start-up delay that is linear in the number of clients when relaxing the fluid model assumption. Moreover, an optimal construction may require building a full mesh between clients (with no bound on node degrees). These reasons lead us to develop a more practical approach that utilizes almost the minimal amount of angel capacity (predicted under the fluid model), while also ensuring that the start-up delay is logarithmic in the number of clients. Our practical approach relies on dividing the stream into substreams, each of which is disseminated along a separate tree. To download the stream, each client subscribes to all the substreams, whereas angels subscribe only to one substream, allowing them to upload at full capacity while not wasting too much upload bandwidth in downloading the whole live stream. The idea of splitting the stream into substreams was proposed in prior work, most notably in SplitStream (Castro et al., 2003) and (Liu et al., 2008). In the related work section, we discuss what we learned from those proposed techniques, and how we avoided some of their shortcomings.

Another limitation of the fluid (optimal) choreography is that it does not consider issues of churn due to node arrival and departures. We address this limitation by incorporating membership management capabilities that ensure uninterrupted service with minimum startup delay. We achieve this by ensuring that the trees used in our construction are well balanced, and by avoiding degenerate cases. Our cloud-based service provides a *registrar* that collects information about clients, making fast membership management decisions that ensure smooth streaming.

We discuss the architecture of our proposed *AngelCast* system, and evaluate a prototype implementation against SopCast (sop, 2012) – a commonly use P2P streaming client. The experimental results carried out on Emulab and PlanetLab show the utility of angels and the effectiveness of our choreographed live stream distribution.

The remainder of this chapter is organized as follows: In Section 4.2, we present the theoretical model that bounds the minimum amount of angel upstream capacity needed to deliver the stream to all clients with the required bit-rate. We also present an optimal fluid construction that achieves that bound and compute the start-up delay associated with it. We conclude that section by

highlighting the effectiveness of using angels over using seeders for live streaming. In Section 4.3, we present our practical construction that avoids the impracticalities of the optimal construction by relaxing the fluid assumption and bounding the node degree. In Section 4.4, we present our *AngelCast* service architecture including the membership management techniques and the design of our protocol. In Section 4.5, we experimentally evaluate our *AngelCast* prototype against SopCast in Emulab and PlanetLab. In Section 4.6, we review the related work. We conclude in Section 4.7 with a summary of results and directions for future research.

## 4.2 Theoretical Bounds

We adopt the Uplink Sharing Model (USM) presented by Munding in (Munding et al., 2008), wherein each client is defined solely by its upstream and downstream capacities. The client is free to divide its upstream/downstream capacity arbitrarily among the other nodes as long as the aggregate upload/download rates do not exceed the upstream/downstream capacity. Hereafter, we use the term *fluid model* to refer to the use of the Uplink Sharing Model along with the ability to infinitesimally divide link capacities. The provider  $P$  is the originator of the live stream, it has an upstream capacity of  $u(P)$ . The set of clients subscribed to the live stream is  $C$  of size  $c = |C|$ . Each client  $c_i \in C$  has an upstream capacity of  $u(c_i)$  and downstream capacity of  $d(c_i)$ . We denote the clients aggregate upstream capacity by  $u(C) = \sum_{c_i \in C} u(c_i)$ . The aggregate angels' upstream capacity is  $u(A)$ . We assume that the stream playout rate  $r$  is constant.<sup>1</sup>

Each client  $j$  should be able to download fresh live content with a rate  $x_j = \sum_{i \in C \cup A \cup P} x_{ij}$  greater than the playout rate  $r$ , where  $x_{ij}$  is the rate between nodes  $i$  and  $j$ . By definition the provider's upload bandwidth is not less than the playout rate  $u(P) \geq r$ , otherwise the provider cannot upload the live stream. Also, it is fair to assume that the downstream capacity of all clients is greater than the playout rate  $d(c_i) \geq r \quad \forall c_i \in C$ , otherwise these clients will not be able to play the live stream at the desirable playout rate.

---

<sup>1</sup>It is recommended to use CBR for live streaming. But in the case of variable bit-rate encoding (VBR), we can use the optimal smoothing technique to achieve a constant bit-rate (CBR) (Salehi et al., 1996).

### 4.2.1 Optimal Angel Allocation

In this subsection, we derive the minimum amount of angel upstream capacity needed in order for all clients to receive the live stream with rate  $r$ . First, we provide a lower bound on the angel upstream capacity, then find an optimal fluid allocation scheme achieving this bound.

**Theorem 1.** *The minimum angel upstream capacity needed for all clients to receive the stream at a prescribed playout rate  $r$  is:*

$$u(A) \geq \frac{c^2}{c-1} * \left( r - \frac{u(P)+u(C)}{c} \right)$$

*Proof.* For a client to receive the stream live, its download rate should equal the playout rate. Thus, the slowest client should receive content with rate not less than  $r$ ;  $\min_{j \in C} \{x_j\} \geq r$ . Because the average is always greater than the minimum, the average download rate should exceed  $r$  as well; if  $\min_{j \in C} \{x_j\} \geq r$  then  $\frac{\sum_{j \in C} x_j}{c} \geq r$ .

First, let us consider the case of no angels. The uplink sharing model dictates that the aggregate downstream capacity cannot exceed the aggregate upstream capacity in the swarm:  $u(P)+u(C) \geq \sum_{i \in C} x_i$ . To optimally utilize  $u(A)$  of the upstream capacity of the angels, an angel must download fresh data with a rate of at least  $u(A)/c$  then upload it to all  $c$  clients. Thus, in case of using angels, we have:

$$\begin{aligned} u(P) + u(C) + u(A) &\geq \sum_{i \in C} x_i + \frac{u(A)}{c} \\ \frac{u(P) + u(C) + u(A)}{c} - \frac{u(A)}{c^2} &\geq \frac{\sum_{i \in C} x_i}{c} \geq r \end{aligned}$$

Rearranging this inequality allows us to derive the angel upstream capacity needed to achieve the prescribed playout rate  $r$ .

$$u(A) \geq \left( \frac{c^2}{c-1} \right) * \left( r - \frac{u(P) + u(C)}{c} \right)$$

□

Not surprisingly, the bound in Theorem 1 suggests that the capacity of needed angels grows linearly with the number of clients and with the deficit between the playout rate and the client share of the provider and clients upstream capacities.

**Theorem 2.** *All clients can achieve the playout rate  $r$  when:*

$$u(A) = \frac{c^2}{c-1} * (r - \frac{u(P)+u(C)}{c})$$

*Proof.* We prove that the lower bound on the minimum angel upstream capacity is achievable by construction. Using a fluid model, we choreograph the transfer rates between nodes so as to achieve a download rate that equals the playout rate for all clients. The set of Equations 4.1 has these rates. The provider sends data to client  $c_i$  with rate  $x_{Pi}$ . The client  $c_i$  in turn forwards this data to other clients  $j \in C$  with rate  $x_{ij}$ . The provider sends data to the angel with rate  $x_{PA}$ , the angel relays this data to the  $c$  clients immediately.

$$\begin{aligned} x_{Pi} &= \frac{u(c_i)}{c-1} + \delta \quad \forall c_i \in C \\ x_{PA} &= \frac{u(A)}{c} \\ x_{ij} &= \frac{u(c_i)}{c-1} \quad \forall c_i \in C, i \neq j \\ \text{where } \delta &= \frac{u(P) - r}{c-1} \geq 0 \end{aligned} \tag{4.1}$$

These rates guarantee that each client receives data at rate  $r$  without violating the upstream capacity constraint of any node. The aggregate download rate for client  $j$  (from all sources) will be

$$\begin{aligned} x_j &= x_{Pj} + x_{Aj} + \sum_{i \in C, i \neq j} x_{ij} \\ &= \frac{u(c_j)}{c-1} + \delta + \frac{u(A)}{c} + \sum_{i \in C, i \neq j} \frac{u(c_i)}{c-1} \\ &= \frac{u(C)}{c-1} + \frac{u(P) - r}{c-1} + \frac{1}{c} * \frac{c^2}{c-1} * (r - \frac{u(P) + u(C)}{c}) \\ &= r \end{aligned} \tag{4.2}$$

The upload rate of each client,  $i$ , will not exceed its upstream capacity as  $(c-1) * u(c_i) / (c-$

1) =  $u(c_i)$ . The same can be said about the angels:  $c * u(A)/c = u(A)$ . Also, the aggregate upload rate from the provider will not exceed its capacity:

$$\begin{aligned}
& x_{PA} + \sum_{j \in C} x_{Pj} \\
&= \sum_{j \in C} \left( \frac{u(c_j)}{c-1} + \delta \right) + \frac{1}{c} * \frac{c^2}{c-1} * \left( r - \frac{u(P) + u(C)}{c} \right) \\
&= u(P)
\end{aligned} \tag{4.3}$$

To ensure that each client receives non-duplicate data. The provider sends unique data to the angels. As for the clients, each client receives unique data with rate  $u(c_i)/(c-1)$  and the same data with rate  $\delta$  to all clients.  $\square$

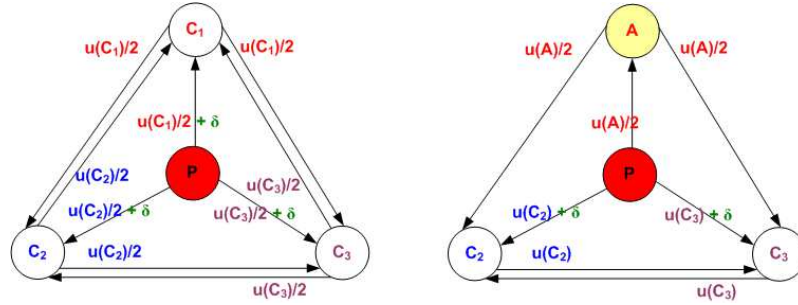
Figure 4-1 illustrates two examples of the optimal construction. The left side of Figure 4-1 is an example with three clients whose upstream capacities are sufficient to achieve a playout rate of  $r$ , thus there is no need for angels. Each client splits its upstream capacity between the other two clients. The provider sends data to clients with a rate that equals half their upstream capacity plus  $\delta$ . The  $\delta$  part of the upload rate is identical, in terms of its content, to all clients, while the other part is unique, ensuring the uniqueness of data disseminated. On the right side of Figure 4-1 is an example where the upstream capacity of the provider and the two clients is not enough to secure a playout rate  $r$ . Thus, an angel is needed. Here, the angel downloads from the provider and uploads to the clients. Each client downloads from the provider and uploads some of what it receives from the provider to the other clients (and angels do not download from clients).

#### 4.2.2 Startup Delay

In this section, we study the effect of packetization on startup delay. Assume that the unit of transfer is of size  $\psi$ . In the fluid model,  $\psi$  approaches zero. In a practical setting, each node cannot forward a packet to another node before it finishes receiving it.

**Theorem 3.** *Using the optimal construction in Theorem 2, the startup delay,  $D$ , is:*

$$D = \frac{\psi}{r} + \frac{\psi * (c-1)}{\min_{c_i \in C} u(c_i)}$$



**Figure 4-1:** Illustrative examples of the optimal construction: 3 clients not in need of any angels (left) and 2 clients in need of one angel (right).

*Proof.* The proof is by construction. The delay until all clients receive a certain packet consists of two parts; the first is the delay until a client receives this packet from the provider  $\psi/r$  and the second is the delay until it forwards the packet to all the other  $c - 1$  clients. The client that is forwarding the packet to all the other clients can be the one with the smallest upstream capacity, thus we divide the amount of data needed to be sent,  $\psi * (c - 1)$ , by the minimum upstream capacity  $\min_{c_i \in C} u(c_i)$ .  $\square$

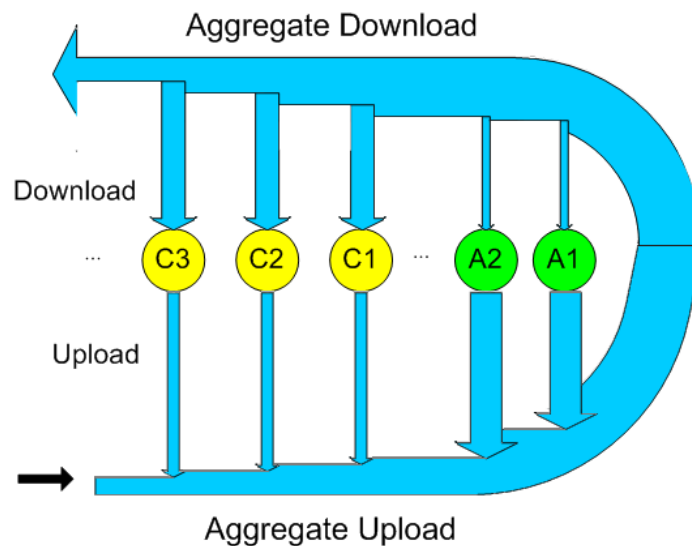
Therefore, if the goal of the system is for all clients to enjoy an uninterrupted service, each client should fill a buffer of size  $B$  before starting the playout, where:

$$B = r * D = \psi + r * \frac{\psi * (c - 1)}{\min_{c_i \in C} u(c_i)}$$

This result suggests that the startup delay grows linearly with the packet size  $\psi$  and with the number of clients  $c$ . Under the fluid model assumption, this is not consequential because  $\psi = 0$ , resulting in a startup delay of zero as,  $\lim_{\psi \rightarrow 0} D = 0$ . In practical settings however,  $\psi \neq 0$  – e.g., it could be the MTU of a TCP packet  $\simeq 1,400$  Bytes. In such cases, the optimal construction may result in significant startup delays for large number of clients. In Section 4.3, we propose a dissemination strategy that achieves the desired download rate using minimum angels capacity while keeping the startup delay under a reasonable (logarithmic) bound.

### 4.2.3 Implications on the Role of Angels

The premise behind this work is that there is a significant gap between the clients' upstream and downstream capacities offered by ISPs. The promised higher downstream capacity encourages content providers to stream at an ever-increasing rate. Pure P2P technology helps alleviate the cost on content providers by utilizing the clients upstream capacity. To bridge this gap, many peer-assisted file-based streaming constructions were proposed. Ours is the first to realize that it is more efficient for the added resources to download a fraction of the live stream instead of its entirety. Figure 4-2 illustrates the ecosystem of our angel-based content acceleration architecture. In this ecosystem, the aggregate downloaded data equals the aggregate uploaded data. The client upstream capacity cannot match the download rate of many streams. Thus, we need to add agents to this ecosystem that produce more than they consume. Angels provide that by downloading a fraction of the stream instead of consuming the already scarce resource of upstream capacity by downloading unnecessary content. In live streaming, content providers do not have the luxury



**Figure 4-2:** Our proposed ecosystem; clients download more than they can upload, and angels download as little as possible.

of uploading the content to many servers beforehand. Thus, those servers will compete over the upstream capacity with clients. In a peer-assisted streaming mechanism, like the one presented

in (Liu et al., 2008), the system relies on more server capacity to stream at a higher stream rate not supported by the clients upstream capacity. Figure 4-3 shows a numerical analysis of the performance of angels against servers in a hypothetical setting. It compares the number of angels, each with upstream capacity ten, versus the number of servers, each with upstream capacity ten too, (on the y-axis) needed to achieve a streaming rate (on the x-axis). The results are shown for a family of curves with varying clients' ratios of upstream-to-playout rates – ranging from 3:10 to 9:10. The streaming rate (x-axis) varies between 1 and 9. There are 100 subscribed clients and the number of clients an angel can connect with concurrently is 40. The formula to compute how many servers we need is

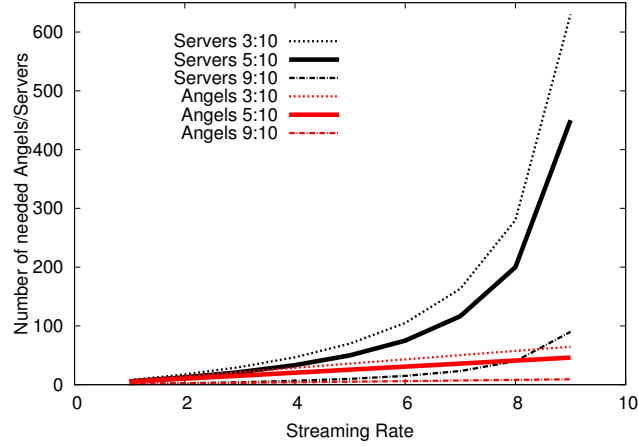
$$\frac{c * r}{u(a) - r} * (1 - ratio)$$

and the formula for the number of angels is

$$\frac{c * r * k}{u(a)(k - 1)} * (1 - ratio)$$

. The growth in the number of angels is linear with the stream rate. However, if we use servers, the growth is super-linear. For example, when the streaming rate is 9 and the server upstream capacity is 10 we need, nine-tenth of a server capacity is wasted in uploading the stream to another server (90% overhead). The graphs compare different values for the ratio between the upstream capacity to the playout rate. The greater the gap, the more angels/servers the system will need. Clearly, angels are significantly more efficient than servers, especially when the stream playout rate is large.

Figure 4-4 shows the maximum number of clients that can download at a desired rate given the number of available angels/servers. We fix the ratio between the upstream rate and stream playout rate to 1:2. At low stream rates, a few high-capacity angels/servers can support many clients. As the stream rate increases, the number of clients satisfied by the service decreases. Again, the angels are more efficient as they are able to serve more clients consuming the same amount or resources.



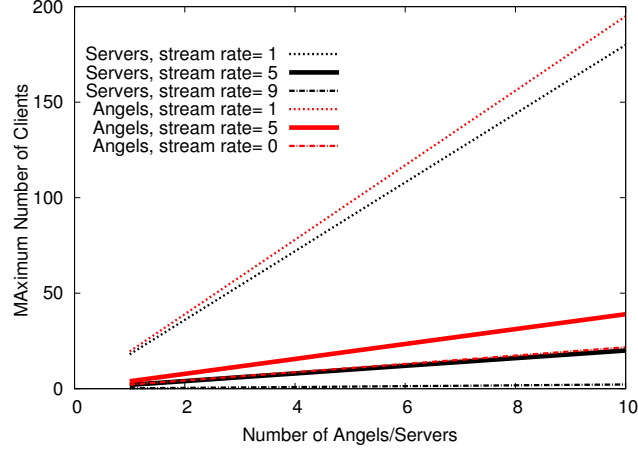
**Figure 4-3:** The number of angels versus the number of servers needed to achieve a desired playout rate for 100 clients.

### 4.3 A Practical Construction

The optimal construction in Theorem 2 requires each client to connect to all other clients. This leads to operational challenges and a start-up delay that grows linearly with the number of clients. To mitigate these problems we developed a new dissemination mechanism that constrains the node degree of each client to  $k$ , *i.e.*, each clients can communicate with at most  $k$  clients at any point in time. We call this construction *AngelCast*.

#### 4.3.1 Bounding the Angel Upstream Capacity

The limit on the node degree influences angel effectiveness inversely. Ideally, an angel would download a small fraction of the stream and would upload it to *all* clients. Under a bounded-degree assumption, it is intuitive that we would need more angel capacity.



**Figure 4-4:** The number of clients that can be supported by a fixed number of angels or servers.

### Lower Bound on Angel Capacity

**Theorem 4.** *The minimum angel upstream capacity needed for all the clients to download at the playout rate, when each node is constrained to connect to only  $k$  other nodes, is:*

$$u(A) \geq \frac{k*c}{k-1} * (r - \frac{u(P)+u(C)}{c})$$

*Proof.* Similar to the proof of Theorem 4.1, in order to optimally utilize  $u(A)$  of angel upstream capacity, an angel must download data at a rate of at least  $\frac{u(A)}{k}$  then upload it to a maximum of  $k$  clients, given that  $k \leq c$ . On the one hand, forwarding the same data to more than  $k$  clients *simultaneously* would violate the out-degree constraint, and on the other hand, forwarding the same data to more than  $k$  clients *on stages* would result in the reception of stale and out of stream data to some clients. Also, tearing down connections and building new ones frequently and systematically would result in performance degradation due to the nature of the transport protocols.

To ensure that the aggregate upstream capacity is more than the aggregate download rate:

$$\begin{aligned}
u(P) + u(C) + u(A) &\geq \sum_{\forall i \in C} x_i + \frac{u(A)}{k} \\
\frac{u(P) + u(C) + u(A)}{c} - \frac{u(A)}{k * c} &\geq \frac{\sum_{\forall i \in C} x_i}{c} \geq r \\
\text{Thus : } u(A) &\geq \left(\frac{k * c}{k - 1}\right) * \left(r - \frac{u(P) + u(C)}{c}\right)
\end{aligned}$$

□

For small swarms, when  $c \leq k$ , the bounded-degree constraint is never reached, thus the bound on the angels upstream capacity does not change. Even when  $c > k$  and  $k$  is relatively large, we would not need significantly larger angel capacity as  $c/(c - 1) \simeq k/(k - 1) \simeq 1$ . For example if  $k = 100$  the overhead due to constraining the out-degree is around 1% even when the number of clients is extremely large.

### Construction Under a Bounded-Degree Assumption

The optimal construction used in Theorem 2 assumes the ability of each client to connect to all other clients simultaneously. In the remainder of this section, we develop a practical construction under the constraint of limited node degree,  $k$ , where each node can communicate with only  $k$  other nodes at any point in time.

Our construction divides the stream into  $m$  substreams. We disseminate each substream using a multicast tree and each client subscribes to all the  $m$  trees. The rate of dissemination of all the substreams,  $r_t$  is equal, such that the sum of the substreams equals the playout rate, *i.e.*,  $r_t = r/m$ . This construction is similar to what is done in Splitstream (Castro et al., 2003) and CoopNet (Padmanabhan et al., 2003).

The number of trees,  $m$ , depends on the allowed degree of any node,  $k$ . For  $d$ -ary trees, each node is connected to at most  $d + 1$  other nodes (one parent and  $d$  children). Thus, the number of trees is bounded by:  $m \leq \lfloor k/(d + 1) \rfloor$ . Choosing the arity of the trees is not trivial. On the one hand, choosing a small arity allows for a greater number of trees, each with a small substream rate. This will minimize the unassigned client capacities  $\sum_{i \in C} (u(c_i) \bmod r_t)$  and utilizes angels

more efficiently. As the theoretical results showed, angels are best utilized when they download the smallest substream which allows them to upload with their maximum bandwidth. On the other hand, choosing a larger arity would result in a smaller start-up delay, as we prove in Section 4.3.2. Also, choosing a large arity enables the utilization of the client with high upstream capacity in full because when a client subscribes to all trees as a parent of  $d$  children, it can forward data with maximum rate  $d * r_t * m = d * r$ . Thus, any client upstream capacity above  $d * r$  will be wasted. Consequently, choosing the right arity has to balance utilizing the resources and providing a small start-up delay.

Beside deciding on the number of trees and their arity, we need to decide on the placement of each client in each tree. First, let us consider the case in which there are no angels. Adding angels to this construction is straightforward and will be explained shortly. In our construction, each client calculates how many children it can adopt across all trees, which equals the upstream capacity of the node divided by the rate of a substream,  $r_t$ . Let us call this parameter  $l_i$ , where  $l_i = \lfloor u(c_i)/r_t \rfloor$ . Our construction dictates that these children be allocated in the minimum number of trees. This is necessary to avoid degenerate trees, where parents have only one child. Thus, the number of trees where this client can have  $d$  children is  $g_i = \lfloor l_i/d \rfloor$ . Client  $i$  can have the remaining children assigned to one more tree and it will be a leaf in all the other trees. Whenever a new client arrives, it will join all trees. To ensure that no tree is starving for bandwidth while another one has an abundance of it. The new client will be a parent in the trees where there are fewer places to adopt more children. We use *vacantSpots* to denote the number of places in a tree where it is possible to adopt more children.

The position of clients in each tree is equally important. To ensure a small start-up delay, the depth of the trees should be minimized. In a bounded-degree setting, the path from any node to the root should be logarithmic in the size of the swarm. Subsection 4.3.2 shows that full trees with large arity achieve that. Therefore, nodes that can adopt more children should be higher in the tree. Subsection 4.4.1 shows how we add/remove clients and change connections while maintaining low-depth trees.

Adding angels to this construction is straightforward. The minimum upstream capacity of an-

gels needed,  $u(A)$ , is given by the lower bound in Equation 4.4. This upstream capacity would be divided equally between a number of angels  $n_a = u(A)/(k * r_t)$ , each of which will be assigned to a different tree. Each angel will have  $k$  children in that tree. Whenever the number of vacantSpots in a tree falls below a certain threshold, we know that this tree is in poor health, hence we add an angel to that tree. When a tree has too many vacantSpots, we eliminate an angel, if any exists in this tree. Even though when an angel is added, it is added to a single tree, the health of the other trees will also improve. This is because newer clients will not need to become parents in this tree and will allocate their resources to other needy trees.

As for the provider, it will be a parent to the roots of the trees. The excess capacity of the provider can be utilized fully as well, as the provider can adopt more children. To avoid adding angels in all trees at the start, the provider focuses its extra capacity in fewer trees.

To conclude, this construction allows each client to download with rate  $r$  and achieves near optimal utilization of the clients upstream capacity. The remaining upstream capacity that is not enough to adopt a child equals  $\sum_{i \in C} (u(c_i) \bmod r_t)$ , which can be seen as insurance in the case of bandwidth oscillations. The gap between the clients' upstream capacities and the playout (and hence download) rate can be supplemented by angels, and each node will not have to connect to more than  $k$  other nodes. In the following subsection, we show that our construction has, on average, a logarithmic startup delay in the number of clients,  $c$ .

### 4.3.2 Bounding the Startup Delay

In this section, we compute the startup delay given our construction in Subsection 4.3.1. A node with  $d$  children dedicates  $r_t$  of its upstream capacity to each one of them. Therefore, if we serialize the dissemination by sending a packet to a child at a time, the time to send a packet of size  $\psi$  to the first child will be  $\psi/(r_t * d)$ . The time to disseminate a packet of size  $\psi$  to all the  $d$  children, is  $d * \psi/(r_t * d) = \psi/r_t$  seconds. Each tree has  $c$  children, thus, it has  $\log_d(c)$  levels. Therefore, the time it takes for the last client in the last level of the tree to download a packet is:

$$D = \log_d(c) * \frac{\psi}{r_t} = \log_d(c) * \frac{\psi}{r * (d + 1)/k}$$

This startup delay is the minimum when  $\frac{\partial D}{\partial d} = 0$

$$\frac{\partial D}{\partial d} = \frac{\psi * k * \ln(c)}{r} * \left( \frac{-(\frac{d+1}{d} + \log(d) * 1)}{\ln^2(d)} \right)$$

$$\ln(d^*) = -\frac{d^* + 1}{d^*} \quad \text{At } \frac{\partial D}{\partial d} = 0$$

The above means that in order to minimize the start-up delay, the degree of the tree should be maximized – *i.e.*,  $k - 1$ . This result illustrates the trade-off between minimizing the start-up delay and minimizing the needed angel capacity: The more trees we have, the better we are utilizing the angels/clients at the expense of increasing the start-up delay.

#### 4.4 AngelCast Architecture

Figure 1-1 shows the agents in our system. While the registrar does not disseminate data, it choreographs the connectivity of clients and angels in the system. The registrar is the main agent in our cloud service. Content providers contact the registrar to enroll their streams. The registrar uses the profiler to estimate the upstream capacity of clients. The accountant uses the estimated gap between the clients' upstream capacity and the stream playout bit-rate to give the content provider an estimate of how many angels it will need.

##### 4.4.1 Membership Management: The Registrar

Live streaming swarms are dynamic in nature. Clients arrive and depart the stream constantly. Also, some bilateral connections between clients can degrade arbitrarily. Thus, it is absolutely essential to incorporate resilience in the design of swarms to ensure that some minimal quality of service is maintained. In this section, we explain how *AngelCast* handles membership management, *i.e.*, handling client arrival and departure, and replacing degraded connections with new ones. Our system relies on a special server to achieve that, the registrar. The registrar is a special node in the system that orchestrates the swarm and choreographs the connectivity of the clients. When a new node joins the stream, it contacts the registrar and informs it of its available upstream capacity. The registrar uses a data-structure representing the streaming trees and assigns the new

client to a parent node in each tree. The registrar also decides how many future children the new node can adopt in each tree. Clients can also “complain” to the registrar about their parents. The registrar would pick a new parent for a complaining client, informs the client of its new parent, and also probes the under-performing parent to ensure it is still alive. If not, it pro-actively informs other children to disconnect from it and provides them with new parents.

These decisions are crucial in guaranteeing a continued service with low start-up delay and little disruption. In order to ensure fast response to clients’ requests, the registrar maintains a data-structure containing the state of each node in the system. The state of a node contains: (1) the depth of the subtree rooted at that node, (2) a pointer to the closest descendent with a vacant spot that can adopt one new child, which we call the *closestAdopter*, and (3) the distance to the *closestAdopter*. When a new client joins the swarm, the registrar adds it to the root’s closest adopter in each tree. The arrival and departure of clients changes this state. Algorithm 1 shows the function that updates the *closestAdopter* as well as the distance to it. The value of the *closestAdopter* can change for the new/old parent as well as for predecessors, recursively all the way to the root (by construction, a logarithmic process at worst). If the node has *vacantSpots*, then it is its own *closestAdopter* with distance zero. Otherwise, it picks the *closestAdopter* of one of its children, the one with the minimum distance to its *closestAdopter*. The update will propagate recursively along the path towards the root until the *closestAdopter* of a node along the path does not change.

Figure 4-5 illustrates how the value of the *closestAdopter* changes when a new client arrives. Before the addition, nodes C and D had *vacantSpots*. The root, node A, has node C as the *closestAdopter*. Thus node C will adopt the new node, F. This will change the value of the *closestAdopter* up the path to the root. Nodes C, F will have no *closestAdopter*. Thus, node A’s *closestAdopter* will become node D, its other child, node B’s, *closestAdopter*.

As we alluded before in Subsection 4.3.2, in order to minimize the average start-up delay, we need to minimize the depth of the tree. We note that the degeneration of a tree could be caused by these few nodes that cannot have the maximum number of children. Our goal is to push these nodes down the tree to avoid this condition. In order to do so, we allow new nodes to *intercept* certain connections. By intercepting a connection we mean severing a connection between a parent and a

---

**Algorithm 1** UpdateClosestAdopter()

---

```

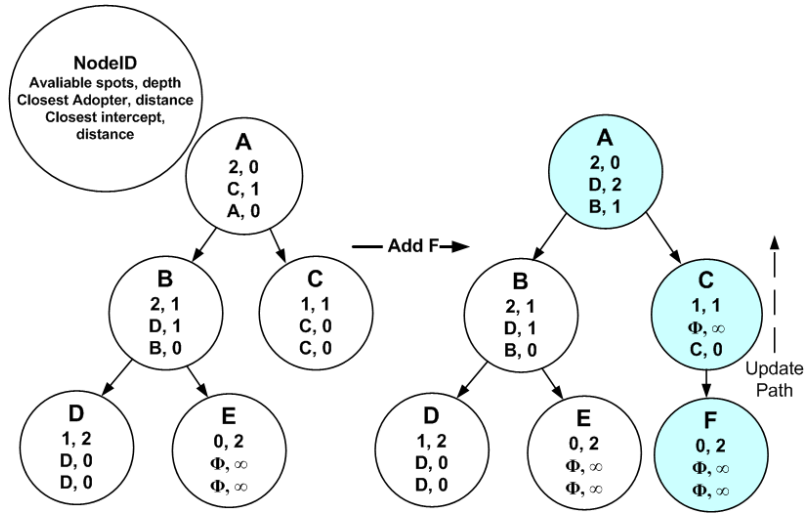
oldAdopter = self.closestAdopter
if This node can adopt more children then
    self.closestAdopter = self
    self.closestAdopterDistance = 0
5: else
    minDistance = Infinity
    adopter = NULL
    for all child in ChildList do
        if child.closestAdopterDistance < minDistance then
10:             minDistance = child.closestAdopterDistance;
                adopter = child.closestAdopter
        end if
    end for
    self.closestAdopter = adopter
15: self.closestAdopterDistance = minDistance + 1
end if
if oldAdopter != self.closestAdopter then
    parent.UpdateClosestAdopter()
end if

```

---

*leaf* child node, making the parent adopt the new node, and making the new node adopt the child node. We prefer interception over adoption if the distance to the *closestIntercept* is smaller than the distance to the *closestAdopter*. We maintain and update the information about the *closestIntercept* and its distance for each node in the tree in a way similar to the *closestAdopter*.

The registrar receives complaints from nodes about their parents when they are not downloading at an adequate bit-rate. The registrar sends a probe to the parent, if the parent is alive and replies, the problem is with the link not with the parent. The registrar severs the connection to that parent and attaches the complaining child, and the subtree below it, to the *closestAdopter* in the tree. We need to ensure that the complaining node is not re-attached to the same parent, or worse to a descendent of its own. We ensure that by setting the *closestAdopter* distance of the parent to a very big number and propagating the update upward the tree, forcing the root to choose a *closestAdopter* away from the complaining node. We then attach the severed subtree to the new *closestAdopter* then set the *closestAdopter* distance of the old parent to zero and update up the tree. By the end of this process, the complaining node gets allocated, with its subtree, to a different part of the



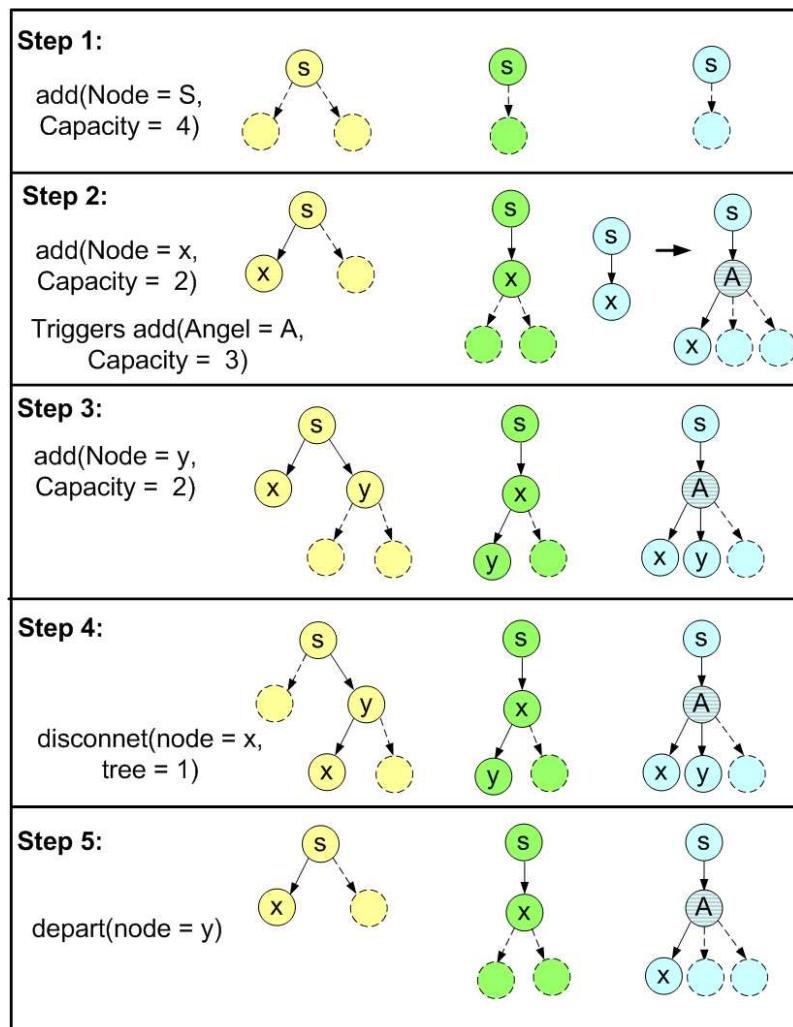
**Figure 4-5:** Node F joins the tree, the registrar updates the data structure accordingly.

tree and the values of closestAdopter of all the nodes are adjusted.

If the registrar's probe to a node results in no response, the registrar concludes that the client unsubscribed from the stream. Thus, it will actively remove it from all trees. When a node is removed from a tree, each of its orphaned children will be added, one by one, to the closestAdopter in the tree. To maintain balance in the tree, the children with smaller distance to their closestAdopter are added before the children with larger distance to their closestAdopter. Our service enables a graceful departure of clients by allowing them to declare their intention to leave the stream.

Figure 4-6 illustrates how our membership management techniques work through an example. Step 1 shows the initial state in which the provider is the root of the three trees, with an upstream capacity of 4 and a stream playout rate of 3. Thus, the root,  $s$ , has 4 vacantSpots, two of which are assigned to the first tree and one vacantSpot for each of the second tree and the third tree. Client  $x$  joins in Step 2. It has two vacant spots, both of them will be assigned to the second tree. As we discussed before we assign a client as a parent in the minimum number of trees to maintain low depth trees. As a result of client's  $x$  arrival, the number of vacantSpots in the third tree is reduced to zero. Thus a new angel,  $A$ , is added automatically to the third tree. The angel's upstream capacity equals 3, allowing it to have three children in one tree. If we had only one tree, the angel would

have been useless as it will have had only one child and downloading as much as it is uploading. Because there are no vacantSpots in the third tree, the added angel will intercept the connection between the provider and client  $x$ . Step 3 illustrates the arrival of client  $y$ . Both of its vacantSpots will be assigned to the tree with the minimum number of vacantSpots, which is tree 1. In Step 4, client  $x$  complains about its connection to the provider in the first tree. The registrar disconnects it from the provider and instructs it to connect to client  $y$ . Step 5 illustrates the departure of client  $y$ . It will be removed from all trees and its children, if any, will be added one by one.



**Figure 4-6:** A hypothetical scenario illustrating the formation of *AngelCast* trees when clients join, leave or change parents.

#### 4.4.2 The AngelCast Protocol

The registrar decides on the number of substreams, the fan-out of the trees and adds the provider as root to all trees. It also initializes the data structure in which the state of the system is kept. The registrar then starts a listener process to receive join requests and complaints from clients. It uses the membership management techniques, described in Subsection 4.4.1, to respond to such request. Whenever the number of vacantSpots in a tree falls below a certain threshold, the registrar instantiates a new machine from the cloud as an angel. When a tree has too many vacantSpots, an angel is released from this tree, if any exists. A full implemented system, serving many live streams concurrently, can instantiate a couple of machines and leave them on standby at all time. Therefore, in the case when a stream is in need of help, an angel would be ready to help and there would be no need to wait for the typical delay associated with acquiring a machine from the cloud.

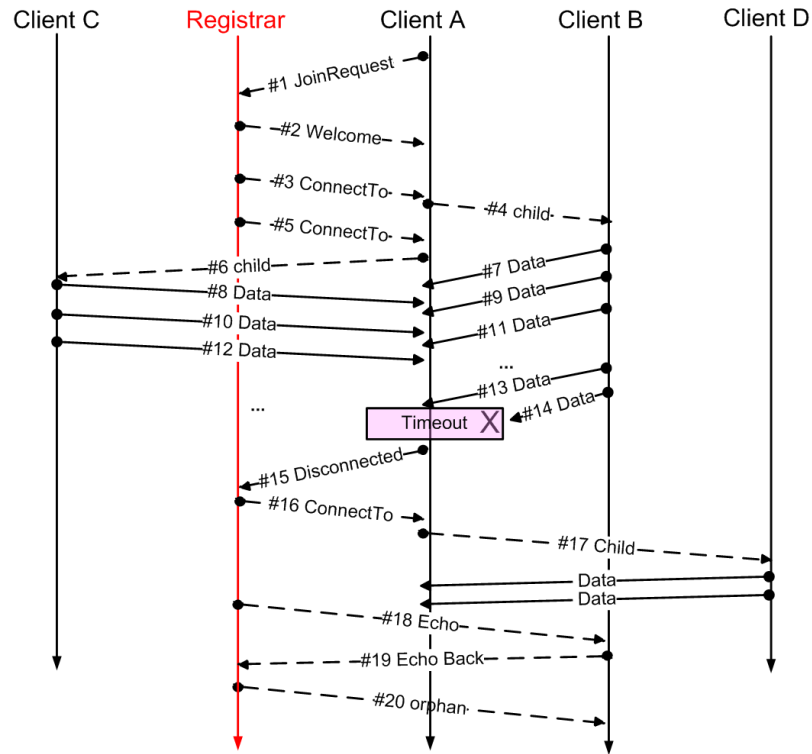
In our implementation, the provider and clients need to download plug-ins (software) to enable them to interact with the *AngelCast* system. On the provider side, the software divides live content as it arrives into substreams, which are maintained in separate substream buffers. The content of each buffer is divided into chunks of fixed size. The provider also starts a listener process, which instantiates threads in response to join requests. These threads read from a substream buffer and sends chunks of data to the client. On the client side, the software starts by contacting the registrar asking to join a stream. The registrar replies with information about the stream as well as the identity of a parent capable of serving the (sub)stream in each tree. Upon contacting these parents, a newly-arriving client is able to start downloading the substreams into different buffers. The software on the client side includes a thread that reads from these buffers in a round robin fashion and writes to a local port using an HTTP server. Any media players with the ability to play streams over HTTP (*e.g.*, `mplayer`) can play out the stream from this local port. Similar to the software on the provider side, the software at the client also starts a listener process, which instantiates threads in response to join request from children. When such a request is received for a specific substream, the software sends data from the buffer associated with that substream. The angels' software is similar to the software at the clients, except that angels need only subscribe to one substream and then listen and serve incoming client requests for that one substream.

Figure 4-7 illustrates an example of the interaction between a newly arriving client, client *A*, the registrar and other clients chosen by the registrar as parent to client *A* in two trees. Client *A* joins the system by sending a “Join” message (message #1) to the registrar, the “Join” message contains the ID of the requested stream and the upstream capacity it is willing to contribute. The registrar replies with a “Welcome” message (message #2) informing the client with the stream playout rate, the number of trees (substreams), the chunk size, and the identity of the chunk it should download first. The registrar also sends to the client the IP/port# of each parent that the client should contact for each substream (Messages #3 and #5). When contacting these parents (messages #4, #6), the client specifies the tree ID and the chunk it is expecting to start downloading from. Messages #7 to #13 represent data messages from the parents to client *A*. The data messages has the chunkID and the associated streaming data. In this example, the streaming continues smoothly (message #13), after that the connection is lost or degraded (message #14 is lost). The client realizes that and contacts the registrar before it depletes its buffer. It sends message #15 to the registrar informing it that it was disconnected from its parent in the first tree, client *B*. The registrar replies to client *A* with a new parent to connect to, client *D* (Message #16). The client sends message #17 to the new parent, client *D*, requesting chunks starting at where it stopped, the new parent starts streaming this substream. In the meanwhile, the registrar probes old parent, client *B*, to check if it is still alive or not (Message #18). In this case it receives an “EchoBack” message (#19), and the registrar sends the old parent message #20 informing it to disconnect that child. If the registrar had not received an “EchoBack” message, it would have assumed the client is disconnected and would have sent a proactive message to all its children in all trees to connect to a different parent. For security reasons, the registrar sends messages to the parents informing them which clients to accept as children. The client can ignore any download request from any unauthorized client. We omitted those messages from this example for simplicity.

We implemented a prototype of our *AngelCast* protocol in python.<sup>2</sup> Our prototype includes the code for the registrar, provider, clients and angels. Our prototype does not include the profiler, thus clients report how much upstream capacity they are willing to contribute to each swarm.

---

<sup>2</sup>The code is available at: <http://csr.bu.edu/angelcast/>



**Figure 4-7:** An interaction diagram showing the exchange of messages between a new client, the registrar and the parents.

## 4.5 Experimental Evaluation

To evaluate the performance of our *AngelCast* prototype, we deployed it in the widely used research platforms: Emulab(Emu, 2010) and PlanetLab(pla, 2010). On the one hand, the Emulab experiments give us accurate insights by isolating our protocol from other experiments, and also making it possible for our results to be repeatable. On the other hand, the PlanetLab experiments are meant to validate that *AngelCast* performs well “in the wild” on the Internet.

Our main motivation in performing these experiments is three-fold: (1) establish confidence in our implementation by comparing its performance to that of widely used streaming solutions, (2) establish the effectiveness of deploying angels from the cloud for the purpose of guaranteeing the desired streaming rates, and (3) measure the performance of our system under churn. We deploy the registrar on a machine of its own. The angels were also deployed on Emulab/PlanetLab

machines instead of the cloud.

The first set of experiments aims at validating our *AngelCast* prototype by comparing its performance to that of SopCast(sop, 2012). SopCast is a popular P2P streaming client used widely on the Internet. We ran SopCast and *AngelCast* protocols on the same set of machines at the same time to neutralize unpredictabilities related to host/network processes (e.g., cross-traffic).

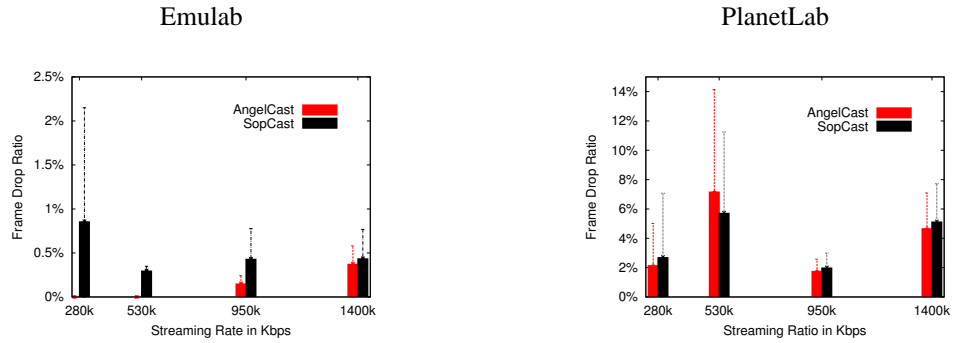
We performed experiments to compare the frame drop-ratio of *AngelCast* vs Sopcast for streams of varying rates (280Kpbs to 1.4Mbps) on Emulab. The results<sup>3</sup> are shown in Figure 4-8a. Emulab machines tend to have significantly higher upstream bandwidth than the average household, thus we use traffic shapers to limit the upstream capacity of the nodes. We limit the upstream capacity of the provider to twice the stream rate,  $2 * r$ , and the 32 clients upstream capacity to  $(4/3) * r$ . This assignment guarantees that there is enough upstream capacity for all clients, thus, there is no need for angels. In this baseline experiment, the stream is split into ten substreams each is disseminated through a trinary tree. The start-up buffer between the time a client requests to join a stream and the start of the playout is four seconds. The result shows that the frame drop-ratios of *AngelCast* and SopCast are comparable when the upstream capacity is plentiful.

Figure 4-8b shows the result of the same previous experiment on PlanetLab. The frame drop-ratio is significantly higher than in Emulab but the performance of *AngelCast* and Sopcast is still comparable. Because Emulab allow us to perform repeatable experiments and to isolate the performance from other experiments running on the same machine, we decided to run the rest of the experiments on Emulab.

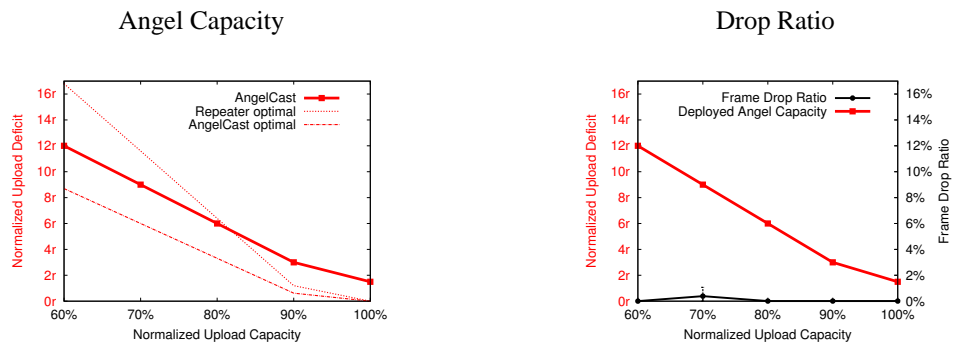
The second set of experiments aims to characterize the effectiveness of angels. Our system deploys angels when a tree has no vacantSpots. We secured 26 clients for downloading a live stream at  $r=950\text{Kbps}$  playout rate. We vary the client upstream capacity between 60% to 100% of the stream rate,  $r$ . *AngelCast* splits the stream into ten trees each with a fan-out of three. The provider's upstream capacity is double the stream rate, ensuring that the provider is not the bottleneck. An angel upstream capacity is 1.5 times the stream rate, ensuring that it is larger than the stream rate, but also that it does not have too many children in one tree (maximum=15).

---

<sup>3</sup>All results are shown within 95% confidence interval.



**Figure 4-8:** The frame drop ratios of *AngelCast* vs *SopCast*.



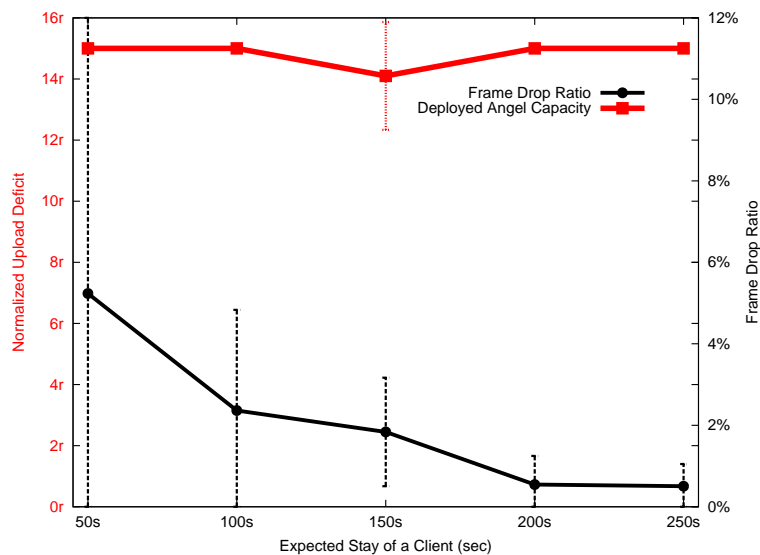
**Figure 4-9:** Minimal amount of angel capacity is sufficient to achieve good stream rates.

On the x-axis of Figures 4-9a and 4-9b, we vary the client upstream capacity, shown as the ratio between the client upstream capacity and the stream rate. On the y-axis of Figure 4-9b we plot the capacity of angels deployed by *AngelCast* against the theoretical bound for the minimum angel capacity (*AngelCast* theoretical). Also, we compare it against the minimum server capacity when the server downloads the whole live stream (*ServerCast*). The results confirm that *AngelCast* utilizes near minimal capacity, and that it achieves significant savings when compared to *ServerCast*. On the left-hand-side of Figure 4-9a (in red), we plot the angel capacity being used and on the right-hand-side scale (in black), we plot the associated frame drop ratio. This experiment verifies that our system achieves reliable streaming utilizing near minimal resources.

The third set of experiments aims to demonstrate the performance of *AngelCast* under churn. In live streaming, churn is due to client arrivals and departures. This is different from how churn is typically modeled in VoD where playback functionality, such as pause, seek and fast-forward must

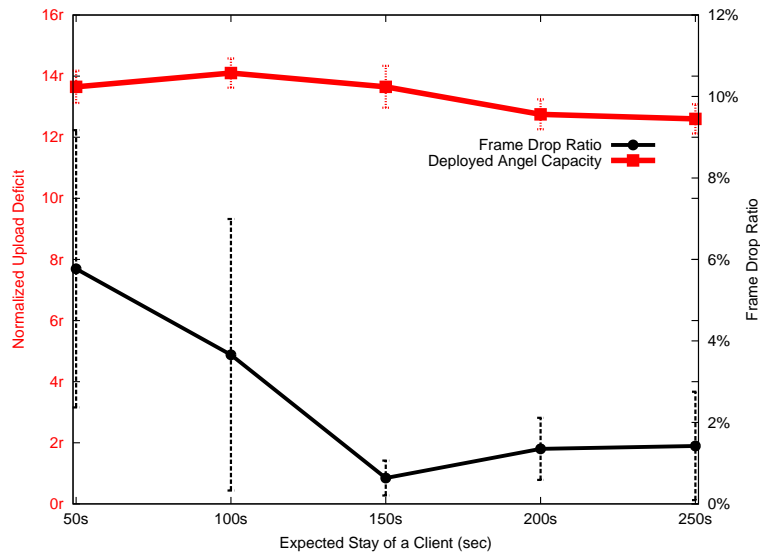
be considered as well (Guo et al., 2008b). Therefore in this experiment, we observe a live stream over a period of 210 seconds, whereby clients join the stream after an exponentially distributed waiting period with a mean of ten seconds. Clients watch the stream for an exponential amount of time of mean 50, 100, 150, 200 or 250 seconds then leave.

We set the upstream capacity of the provider to be twice the playout rate; we set the upstream capacity of angels to be 150% the playout rate; and we set the clients' upstream capacity to be 70% the playout rate. The stream is divided over ten trinary trees and the stream start-up buffer is 4 seconds. The first set of results are in Figure 4-10, on the x-axis we vary the expected duration of the client's stay. On the right y-axis we show the frame drop-ratio. This experiment shows that, as expected, higher churn results in poor performance, (e.g., when clients stay less than a minute on average, the frame drop ratio is as bad as 5% but as clients stay longer, the frame drop ratio drops to 0.5%). On the left y-axis, we plot the aggregate capacity of the deployed angels. When there is no churn, six angels are needed to fill the capacity gap. In the presence of churn, the number of deployed angels is almost fixed to ten, the number of trees. The reason for that is the lack of vacantSpots in each tree at different points during the experiment, due to churn.



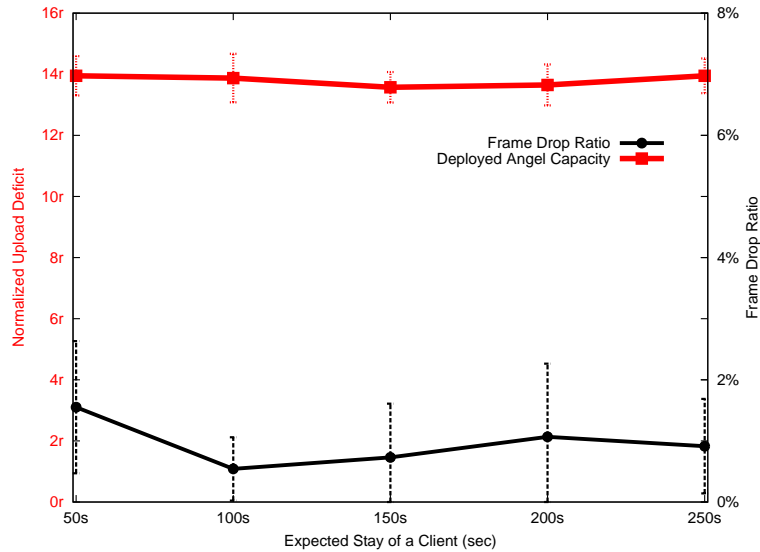
**Figure 4-10:** The performance of *AngelCast* under churn.

In the aforementioned experiment clients leave the system and do not return. Figure 4-11 shows the result for the same experiment except that departing clients take a break for 10 seconds on average then rejoin the stream, maintaining the same multiprogramming level (MLP) in the system. The results are similar to the ones shown before, indicating that whether the number of clients drops with time or stays the same, the performance is similar. For the following experiments, we use the first departure model where clients leave and do not return.



**Figure 4-11:** The performance of *AngelCast* under churn where clients rejoin the stream after they leave.

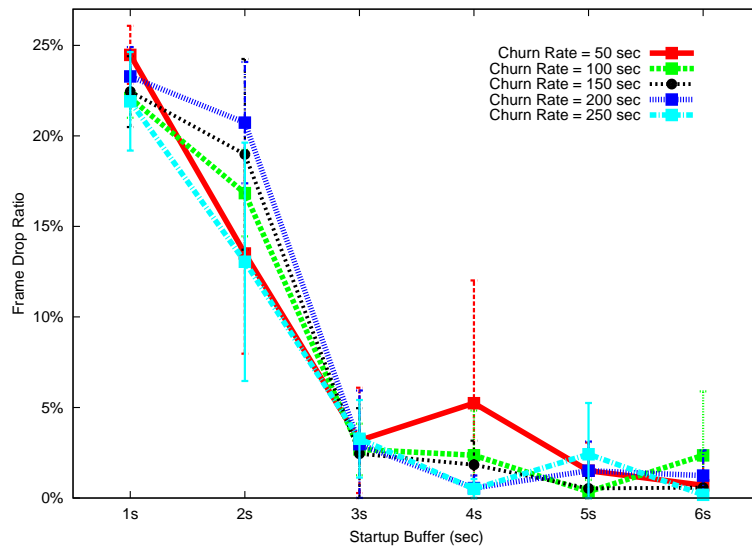
The aforementioned experiment also is for the case when all the clients are churning. But one important question to answer is the effect of churning clients on non-churning clients (*i.e.*, clients who stay for the whole duration of the experiment). On the x-axis of Figure 4-12 we vary the expected duration of a churning client's stay. On the right y-axis we show the frame drop-ratio for *non-churning* clients. The results shows that the performance of non-churning clients is independent of the churn level of other clients. This results illustrates that not only our system tolerates churn, but that the resulting modest degradation is confined largely to churning clients. The next experiment examines the effect of the start-up buffer on the performance of clients. The x-axis denotes the buffering duration before the client starts the playback. The x-axis of Figure



**Figure 4-12:** The frame drop ratio for non-churning clients against varying churning rate for other clients.

4-13 denotes the start-up delay in seconds and the y-axis denotes the frame drop ratio. We plot the results for a family of varying churn levels, where clients stay for a duration of 50, 100, 150, 200 or 250 seconds (no churn). Regardless of the churn level it seems that three to four seconds are enough to achieve smooth streaming. Longer buffers will result in stale content without achieving meaningful performance gains.

Another parameter of the system is the number of trees we construct per stream. The x-axis of Figure 4-14 denotes the number of such trees, the right y-axis denotes the frame drop ratio and the left y-axis denotes the normalized angel capacity utilized. We limit the number of available angels to ten. When the number of trees is under ten, the frame drop ratio is low. The optimal value for this specific configuration is six trees, as the frame drop ratio is lowest and we use the least number of angels. When the number of trees is above ten, some trees will not be assigned angels. Thus in the event of churn there will be significant frame loss in the angel-less trees despite having enough overall angel capacity to deliver the content. We conclude that we should limit the number of trees to a handful to avoid wasting angel capacity or having high frame drop ratio in the angel-less trees.

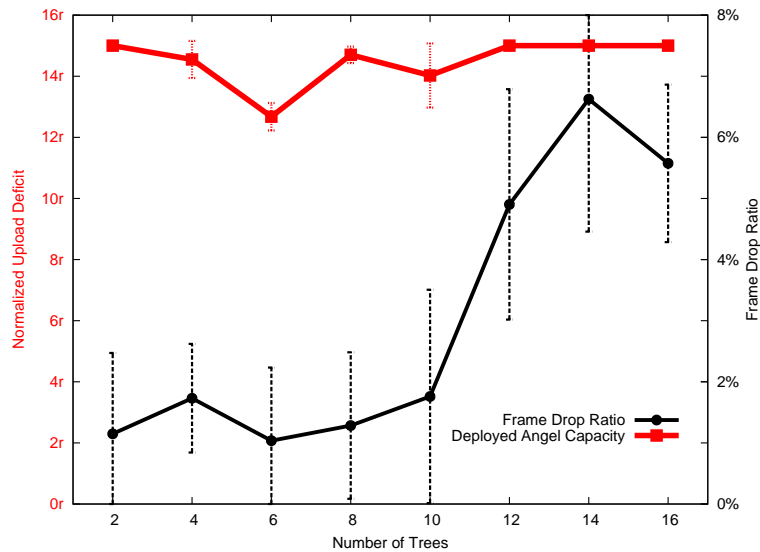


**Figure 4-13:** The frame drop ratio of clients against an increasing start-up buffer. Different curves denote different churning levels.

## 4.6 Related Work

This chapter builds on a rich body of work that approach the problem of content delivery in general and live streaming in particular from a number of perspectives:

**Pull-based Mesh Protocols:** There is a large number of papers/systems that utilize pull-based mesh streaming, such as SopCast, PPLive, UUsee, Joost, and CoolStreaming. Our push-based approach differs from these in that it choreographs the connectivity among nodes to guarantee the quality of streaming for *every* client. We choose to compare *AngelCast* against SopCast (sop, 2012) as it is extensively used, allows users to stream their own channels and works with `mplayer` over Linux. An example of such pull-mesh protocols is CoolStreaming (Zhang et al., 2005), the streaming version of Bittorrent. The difference is that the deadline of a chunk playtime is a factor in the piece selection algorithm. Feng et. al. (Feng et al., ) illustrated the inherent shortcomings of pull-based mesh networks as well as providing a glossary of such protocols.



**Figure 4-14:** The frame drop ratio of clients against an increasing number of substreams (trees).

**Peer-Assisted Content Distribution:** The research community is aware of the promise of P2P in alleviating the load of content distribution on servers. Nonetheless, it is also aware of its limitations, especially in providing sufficient upstream capacity. For file sharing, Wang et al (Wang et al., 2007) proposed adding powerful peers to BitTorrent swarms to accelerate the download rate. This is not optimal, because these “helpers” will unnecessarily consume the scarce upstream capacity in the swarm. In Antfarm (Peterson and Siler, 2009), the authors propose a protocol for seeders to measure the vital signs for multiple swarms and allocate more seeder bandwidth to struggling swarms. Jin et al. (Jin et al., 2002) introduced edge caching to help original servers stream to clients at the prescribed bit-rate. The cache contains the objects where the ratio between the client request rate and the deficit between download rate and playback rate is maximal. This work is different as it requires in-network caching, but it highlights the need for infrastructure help to ensure smooth streaming.

**Modeling P2P Performance:** Qiu and Srikant’s seminal work (Qiu and Srikant, 2004) is the first to characterize the average download time in a swarm. Das et al (Das et al., 2006) extended Qiu’s

model (Qiu and Srikant, 2004) of swarm download rate to incorporate seeders. The result points out that the average download time is inversely proportional to the seeders' aggregate upload rate. However, the increase in the number of peers requires a linear increase in the number of seeders to maintain the same average download time. Although Qiu's model is fundamentally different, these results are in line with our findings. Parvez et al. (Parvez et al., 2008) extended Qiu and Srikant's model (Qiu and Srikant, 2004) to the case of stored VoD, studying the need for sequential progress instead of random chunk download. This is inline with our design concept of choreographing the connectivity of the swarm instead of relying on random chunk download.

Kumar et al. (Kumar et al., 2007) used the uplink sharing model (Mundinger et al., 2008) to find the bound on the highest streaming rate a swarm can handle given the upstream/downstream capacities of the clients. They proposed a fluid construction that achieve their bound. We extend their model to compute the required angel capacity to achieve a desired streaming rate. Likewise, we built an optimal fluid construction that achieves the bound, incorporating angels. They highlighted the gap between the upstream capacity and download rate, stating that most channels in PPLive are running at rate 2-4 times the upstream capacity of many residential broadband peers. Angels are the answer to this problem. Liu et al. (Liu et al., 2008) extended Kumar's model in the case of bounded node degree as well. They proposed creating spanning trees with varying streaming capacities. Given their construction, they provided bounds on the depth of the tree, the maximal upload rate and the minimum server capacity. Their constructions assumes the provider can connect to all clients, that the number of spanning trees can reach the number of clients and the depth of some trees can be linear in the size of the swarm. Our *AngelCast* technique avoids these three problems.

**Multicast Multi-Tree Construction:** Many papers leverage the idea of constructing multiple multicast trees (a forest) for file distribution as well as for streaming. SplitStream (Castro et al., 2003) constructs a forest of multicast trees, one for each stripe, all with the same rate. This approach is different from ours in that the client can choose the number of stripes it wants to subscribe to, thus receiving a subset of the broadcasted data. In contrast to tree-based multicast,

the load on the nodes is balanced as each node is an internal node in one tree but a leaf node on the others. When the internal nodes of one tree cannot adopt any more children, the child must search for some node in the excess capacity tree to download this strip. This leads to inefficiencies as a node could perform a distributed linear search until it finds a suitable parent. More importantly a client with significantly large capacity can adopt many children each in a different tree resulting in degenerate trees and more than logarithmic depth of trees. Our technique makes sure that any parent has at least two children except for the nodes in the second to last level, insuring a logarithmic depth of all trees.

P2PCast (Nicolosi and Annapureddy, 2003) is a modification of SplitStream where all nodes subscribe to all trees to download the same content. They require from each client to participate an upstream capacity equal to the download rate. This could be unrealistic given the diversity of user's upstream capacities and the shielding of some clients behind NATs. CoopNet(Padmanabhan et al., 2003; Padmanabhan et al., 2005) creates multiple trees, each streaming a substream of an MDC encoded video. Their contribution is in providing a mechanism to cope with the fluctuation in the available bandwidth.

**Coding for Adaptive Streaming:** Multiple Description Coding (MDC)(Chou et al., 2003) and Scalable Video Coding (SVC) (Abboud et al., 2011) were introduced to enable clients to download the same video on different rates/qualities. MDC tends to be more theoretical, where any number of descriptions are enough to decode the movie at a rate proportional to the number of received descriptions. SVC codes the video in layers, the base layer is essential for decoding while the subsequent layers are increasingly less important. Such techniques are increasingly deployed to overcome the uncertainty of the available bandwidth and its fluctuation, in particular Dynamic Adaptive Streaming over HTTP (DASH) (Stockhammer, 2011). We consider these techniques complementary to our angels approach. Providers who prefer better than *best effort* delivery to their clients can deploy angels to offer better download rate. MDC and SVC can be deployed in conjuncture with angels in this case, where clients can attempt subscribing to  $m$  trees downloading  $m$  descriptions/layers. If a client is not able to download all the layers, it is still able to decode

the video with lower quality. Our current prototype implementation works over HTTP, we are planning on extending our protocol to conform more closely to DASH.

#### **4.7 Summary of Contributions**

In this chapter we considered the potential of peer-assisted content distribution for affordable high-quality live streaming. As the deficit between clients' upstream and downstream capacities limits the use of pure P2P architecture in such a setting, we introduced the notion of angels – servers who do not have a feed of the live stream and are not interested in downloading it in full. We computed the minimum amount of angel capacity needed in a swarm to achieve a desired bit-rate to all clients and provided a fluid model construction that achieves that bound. We introduced practical techniques that handle limited node degree constraints and churn. We built *AngelCast*, a cloud-based service that assists content providers in delivering quality streams to their customers, while allowing the content providers to leverage the customers' resources to the fullest. We deployed *AngelCast* onto two research platforms: Emulab and Planetlab. We showed that the performance of *AngelCast* is comparable to that of SopCast, a widely used streaming protocol, and that it is capable of supplementing the bandwidth deficit with near minimal capacity, while being able to handle churn. We are currently developing a version of *AngelCast* for wide-spread deployment. We are planning on conducting extensive evaluation and on collecting measurements of its performance in delivering real-world live streams. Our future work will explore ways in which under-utilized angels are managed. This includes the possibility of using angels across multiple swarms. Security is another focus of our future work, especially securing the registrar control messages.

## Chapter 5

# TorAssist: Enhancing Tor Performance For Bandwidth-Intensive Applications

When it was first introduced a decade ago, Tor, the anonymous onion routing protocol, aimed at providing anonymity for latency-sensitive applications, such as web-browsing, as opposed to bandwidth-intensive applications, such as on-demand or live video streaming. This emphasis on latency-sensitive applications is evident from proposed Tor circuit-scheduling techniques (Tang and Goldberg, 2010; Dingedine and Murdoch, 2009) that throttle bandwidth-intensive applications in favor of bursty, latency-sensitive applications. In this chapter, we deviate from this traditional view by identifying key attributes and design decisions that negatively impact Tor’s performance in general and its ability to cater to bandwidth-intensive applications in particular, and by proposing new capabilities that aim to enhance Tor’s performance as it relates to anonymizing bandwidth-intensive traffic. We present results from in-vivo measurement studies that shed light on Tor’s approach to manage load across relays, which manifests itself in the way source-based routing at the end-systems (clients) is handled. We present an analytical model that captures the key attributes of the feedback control inherent in Tor’s approach to load management – namely, probing and circuit selection. We show that changing some of these key attributes yields measurable improvement in terms of overall network utilization as well as better load balancing of relays, resulting in better predictability of individual circuit performance. To boost the performance of bandwidth-intensive circuits, we propose the use of on-demand relays (angels) to not only increase the capacity in the Tor network, but also to implement special bandwidth-boosting functionality using multi-path routing. Our conclusions are backed up with results from simulation experiments.

## 5.1 Introduction

Tor, a second generation onion routing system (tor, 2012a), is based on a source routing protocol that provides anonymity to its clients by routing traffic through a randomly-constructed circuit of intermediate relays. A client does not have to trust any node except the directory server from which it gets the list of relays that are presumed as independent and non-colluding. Typically, the client chooses three relays from such a list to construct a circuit (an overlay path) for traffic to/from any Internet host (server). Through proper encapsulation and encryption, traffic flows from one relay to the next in such a way that any node (relay) in the network can only glean a partial view of the overlay path between the client and the server, thus breaking any association between the source and destination of flows traversing the Tor network (and also hiding the identity of the client from the server). Although Tor is the most widely used anonymity network, it still suffers from major performance issues (Dingledine and Murdoch, 2009). Tor developers and the research community are aware of these performance limitations, as evidenced by a plethora of studies and proposals aiming to enhance the performance of Tor (Reardon and Goldberg, 2009; Tang and Goldberg, 2010; Jansen et al., 2010; He et al., 2007). The main performance issue considered in these works is the highly unpredictable performance of circuits established through the Tor network, *e.g.*, resulting in poor HTTP response time. There have been two main hypotheses as to the culprit: (1) delay through a relay – namely, the amount of time a Tor packet (cell) spends in a Tor relay from the moment it is received till the moment it is sent to the next relay, and (2) latency of an overlay link: the time it takes the cell to traverse an overlay link, from the moment it is sent from one relay till the time it appears at the other end. For example, the authors of *Tunable Tor* (Snader and Borisov, 2010) assumed relay delays are dominant, thus choosing relays based on their estimated throughput, whereas Sherr *et al.* assumed overlay link latencies are dominant, thus choosing relays to minimize the sum of overlay link latencies.

**Motivation and Scope:** When it was first introduced a decade ago, Tor aimed at providing anonymity for latency-sensitive applications, such as web-browsing, as opposed to bandwidth-intensive applications, such as on-demand or live video streaming – hence the emphasis in the

aforementioned studies on delays as the primary gauge of performance. In light of the increasing prevalence of streaming, in this chapter we deviate from this traditional view by focusing on Tor’s ability to deliver higher bit rates, consistently. At a very basic level, this is a matter of provisioning: adding relay “capacity” to the Tor network to allow it to move more bits-per-second. In prior projects of ours (Sweha et al., 2012b; Sweha et al., 2011), we addressed similar provisioning problems in peer-to-peer (P2P) overlays through the use of *angels* – special resources acquired on demand (e.g., from the cloud) for the sole purpose of improving the fidelity of a service. Adding raw capacity to make up a deficit in a particular resource is not a panacea. In addition to deploying angels (if needed), it is necessary to ensure that the underlying P2P system is able to optimally capitalize on this added capacity. To do so, it is often necessary to make judicious decisions, not only as it relates to the number, capacity, and placement of angels, but also as it relates to altering the way angels participate in the underlying P2P protocol.

**Chapter Outline:** In this chapter, we evaluate the extent to which angels may be used in support of bandwidth-intensive applications in a Tor network: we examine the key attributes and design decisions that negatively impact Tor’s performance in general and its ability to cater to bandwidth-intensive applications in particular, and we propose new capabilities, supported through the use of angels, to enhance Tor’s performance relating to anonymizing bandwidth-intensive traffic.

We start in Section 5.2 with some background on Tor’s operation as it relates to how clients construct circuits in a way that is meant to achieve high utilization of Tor resources. We also review prior work aiming to improve Tor’s performance. In Section 5.3 we present results from in-vivo measurement studies that we conducted to shed light on the effectiveness of the various mechanisms used for load distribution in Tor. Our first finding is that the primary culprit for Tor’s poor performance is the uneven throughput of the relays that make up a circuit. This finding implies that better and more predictable performance would result if Tor’s relays are load balanced, *i.e.*, the attainable throughput from individual relays is more even. Indeed, our second finding is that as currently implemented and configured, Tor results in an inadequately-balanced load distribution over relays, resulting in lower overall utilization and higher variance in circuit throughput – both of which are undesirable attributes.

Load distribution in the Tor network is achieved through a set of interacting mechanisms: a *probing mechanism* to estimate the available capacity of relays, a *feedback mechanism* to compute a normalized value that corresponds to the total capacity of the relay, and a *circuit construction mechanism* to preferentially assign load to under-utilized relays. To gain insights into how the interaction between these processes affects load balancing, in Section 5.4, we present an analytical model that captures the key probing and circuit construction attributes of the feedback control inherent in Tor’s load distribution mechanisms. Using this model, we show that changing some of these key attributes yields measurable improvement in terms of overall network utilization as well as lower variability of individual circuit performance. In Section 5.5, we show that these improvements are tangible by presenting results from simulation experiments, in which assumptions are relaxed, and various variant mechanisms are evaluated.

While adding angels to a well-balanced and highly utilized Tor network ensures that the added capacity will be utilized judiciously, it does not provide us with a mechanism via which bandwidth-intensive applications are properly provisioned. In Section 5.6, we introduce a simple angel functionality that boosts the performance of bandwidth-intensive circuits using multi-path routing.

## 5.2 Background and Related Work

**Tor Protocol Basics:** Tor (tor, 2012a) is a source routing protocol that provides anonymity to its clients by means of routing their traffic through an encrypted circuit of intermediate relays (muxes). Tor relies on a core set of around 2,900 persistent relays, typically operated by research institutes or privacy activists. Around 400,000 clients make use of the Tor network, daily, with around 120,000 active at any point in time (tor, 2012b). Tor leverages the (processing and communication) resources of clients by allowing them to function as relays, boosting the overall capacity of the Tor network, which can be seen as a P2P overlay. A Tor client contacts a *directory server* to get a list of Tor relays, from which it constructs circuits (overlay paths). A relay in a circuit cannot identify other nodes in the circuit except for the relay that precedes it and the one that follows it. When a client wants to send a message (*e.g.*, an HTTP request), it recursively encrypts the message with symmetric keys shared with the relays in the circuit, starting with the exit relay (the relay that sends

the HTTP request to, and receives the response from the web server). In the forward direction, each relay peels a layer of encryption (a layer of the onion), sending the resulting message along the path towards the exit relay. In the reverse direction, each relay, starting with the exit relay, adds an onion layer of encryption, sending the resulting message along the reverse path towards the client. Upon receiving such a message, the client recursively peels all layers and consumes the content (displays a web page). By construction, each byte of data sent or received by a client needs to traverse at least three relays, consuming communication and processing capacities at each.

**Acquisition and Management of Relay Capacities in Tor:** Given documented, chronic performance issues, an important aspect of the Tor protocol (and the subject of much research) relates to how Tor manages the capacity and utilization of various relays. In this work, we use *relay capacity* to refer to the number of bytes that a relay can forward per second on all circuits going through it; we use *relay utilization* to mean the fraction of the relay capacity that is in use at any point in time; we use *normalized relay capacity* to refer to the relative overall capacity of a relay; and we use *relay available capacity* to mean the throughput that a circuit is able to command from the relay at any point in time, which is a function of the (raw) relay capacity as well as the number of circuits using that relay in a max-min fair fashion.

As we alluded before, Tor’s load/resource management is done through three interacting mechanisms: a *probing mechanism*, a *feedback mechanism*, and a *circuit construction mechanism*. Tor’s circuit construction mechanism seeks to improve the overall system utilization by increasing the selection probability of relays deemed to have higher capacities. In earlier versions of Tor, each relay self-reported its capacity. To protect against untruthful clients, the current implementation of Tor uses directory authorities (DAs) to estimate the normalized relay capacities (Loesing et al., 2011). Every hour, based on previous estimates, each DA partitions the list of relays into groups of 50, and constructs two-relay circuits using relays in the same group to download a file for the purpose of estimating the available capacity of the constituent relays. These available capacity estimates are fed to a PID controller (Loesing et al., 2011) that implements the *feedback mechanism* for estimating the normalized relay capacities. Let  $F_i$  be the average throughput of the DA’s relay-pair probes involving relay  $i$ .  $F_i \forall i$  is fed into a PID controller (Loesing et al., 2011), which

produces an estimate of the normalized capacity for each relay at time  $t$  as follows:

$$EST_i^t = EST_i^{t-1} * \frac{F_i}{\sum_{\forall j \in R} \{F_j\} / |R|} \quad \forall i \in R$$

The controller increases (decreases) its estimate of a relay’s normalized capacity when the probe throughput through that relay is above (below) average, thus encouraging (discouraging) clients to (from) using this higher-capacity (lower-capacity) relay in circuits they construct in the future.

When a client contacts a set of DAs, it gets from each DA the list of relays, as well as estimates of their normalized capacities,  $EST_i$ , according to that DA. A client takes the average of these estimates, also called the *consensus capacity*. When a client constructs a new circuit, it chooses the constituent relays with probability proportional to the consensus capacity of the relays. Tor clients construct up to twelve circuits and alternate their usage.

**Link and Relay Delays in Tor:** Concerned with the performance of latency-sensitive applications, Dhungel *et al.* (Dhungel et al., 2010) conducted a measurement study to dissect the delay characteristics of Tor circuits. They showed that the delay of a circuit depends on two components: the delay through the relays, and the delay attributed to the latency of traversed overlay links. They concluded that while relay delays are the principal contributor of circuit delays, a sizable minority of circuits are dominated by overlay link delays. More importantly, they found no correlation between the delay introduced by a relay and its advertised or consensus capacity. Moreover, they showed that relay delay fluctuates over time, except for relays with very high capacity. They suggested that Tor’s token-bucket scheduler – for multiplexing the use of a relay among all circuits going through it – needed more frequent replenishment, as circuits using less congested relays end up exhausting their quota of tokens, resulting in under-utilization of these relays. Dhungel *et al.* stopped short of proposing circuit construction algorithms.

**Relay-based vs Link-based Circuit Construction:** In an attempt to address Tor’s performance issues, two classes of circuit construction algorithms emerged: (1) relay-based circuit construction, and (2) link-based circuit construction.

Under the hypothesis that selecting powerful relays results in better performance, the authors of *Tunable Tor* (Snader and Borisov, 2010) propose having relays (as opposed to DAs) estimate

the relay available capacity, and having the consensus capacity be calculated by one authority that distills the normalized capacity estimates of all relays using *EigenSpeed*, a principle component analysis estimator. To minimize overhead, a relay does not actively probe other relays, but rather uses the circuits routed through these relays to passively estimate their available capacities. Tunable Tor allows Tor clients to trade-off anonymity for enhanced performance. A parameter  $\alpha$  governs this tradeoff by decreasing the probability of choosing a relay exponentially with the product of  $\alpha$  times the rank of the relay with respect to available capacity. If  $\alpha$  equals zero, all relays are chosen with equal probability, achieving maximum anonymity at the expense of performance; as  $\alpha$  grows, the more powerful relays are chosen with an increasingly higher probability.

Under the hypothesis that circuit construction should avoid overlay links that are congested or suffer from long delays due BGP routing, Sherr *et al.* (Sherr et al., 2009; Sherr et al., 2007) proposed using Vivaldi (Dabek et al., 2004), a decentralized coordinate system, to measure the distances between different relays. The virtual coordinate of each relay is adjusted every time a new RTT measurement to another relay is available. Any client can measure the overall delay of a circuit by summing the delay of each link in the circuit. A circuit is chosen with probability exponentially proportional to a tuning parameter  $\alpha$  times the rank of the circuit in the sorted list of all possible circuits based on the estimated circuit delay. Again, the parameter  $\alpha$  tunes the tradeoff between performance and anonymity. Panchenko *et al.* (Panchenko and Renner, 2009) advocate link-based measurement to enhance path selection in Tor. They propose two link-based measurements: (1) actively measuring RTTs of circuits, (2) passively estimating link delay using the circuits passing through it. They acknowledge that estimating link statistics is quadratic in the number of relays and suggest selecting relays based on average link-distance to all other relays. They also mention a hybrid approach that combines relay capacity and link delay; they suggest ranking circuits with probability inversely proportional to their aggregate link delay and directly proportional to the minimum relay capacity.

### 5.3 A Measurement Study on Live Tor

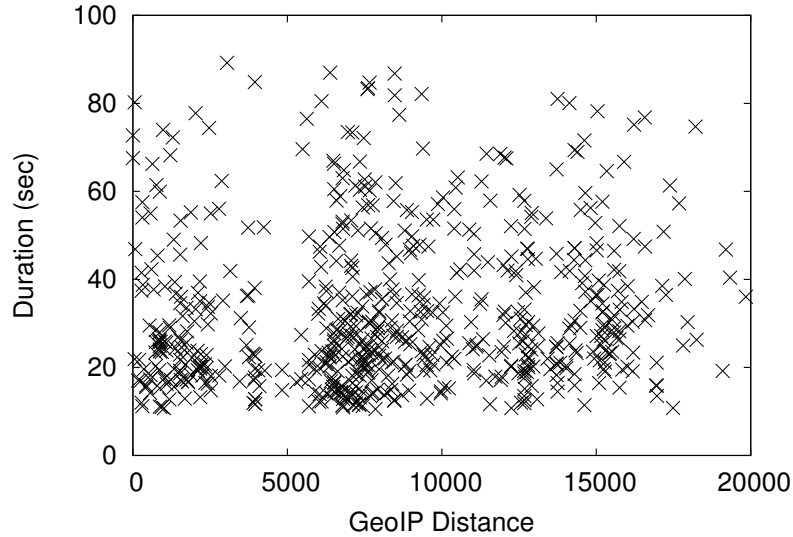
**Experimental Setup:** We used a client machine at Boston University (BU) to create 1,000 circuits, one after the other, over the course of a few days. The client uses Tor to download files from an FTP server, located at BU as well. Both the client and the FTP server are machines with powerful CPUs and 1Gbps connections to the internet, ensuring that any bandwidth bottleneck is in the Tor network, not in the client, the server, or their links. The three relays – entry, intermediate, and exit relays – vary from a circuit to another, in accordance with Tor’s circuit construction mechanism. Tor maintains multiple circuits (typically twelve) at a time and chooses the one to use at random. In our experiment, we limit the number of established circuits to one, by setting the parameter `__DisablePredictedCircuits 1`. We mine the log files produced by Tor to get the ID and IP address of the constituent circuit relays. Using a GeoIP service (fre, 2012), we find the longitude and latitude coordinates of the relay, from which we are able to calculate the geographical distance between adjacent relays in a circuit. Although the geo-distance between two relays does not directly correspond to the propagation delay between these relays, it could serve as a meaningful first-order approximation (Huffaker et al., 2002; Pastor-Satorras and Vespignani, 2007). In addition, we mine the consensus file when each circuit is established (hourly) to collect the consensus capacity estimate for each constituent relay. We use each of the constructed circuits to measure the average download time of a 3MB file and that of a one-byte file (both averaged over eight trials).

**Results:** The first question to consider is whether the delays introduced by circuit links (as estimated by the sum of the distances between adjacent relays in a circuit) is correlated with the circuit throughput. If strong correlation exists, link-based circuit selection would be justified. Figure 5-1 shows the correlation between the sum of link-distances in the circuit (x-axis) and the average duration to download a 3MB file (y-axis). The Pearson product-moment correlation coefficient is found to be  $r = 0.0063$ .<sup>1</sup> This result suggests that there is a very weak (negligible) correlation

---

<sup>1</sup>The distance is computed using the great circle Haversine formula and the average duration is the file size (3MB) divided by the throughput. We plot duration against distance since any correlation between the two is likely to be linear, and hence possible to gauge using the  $r$  statistic.

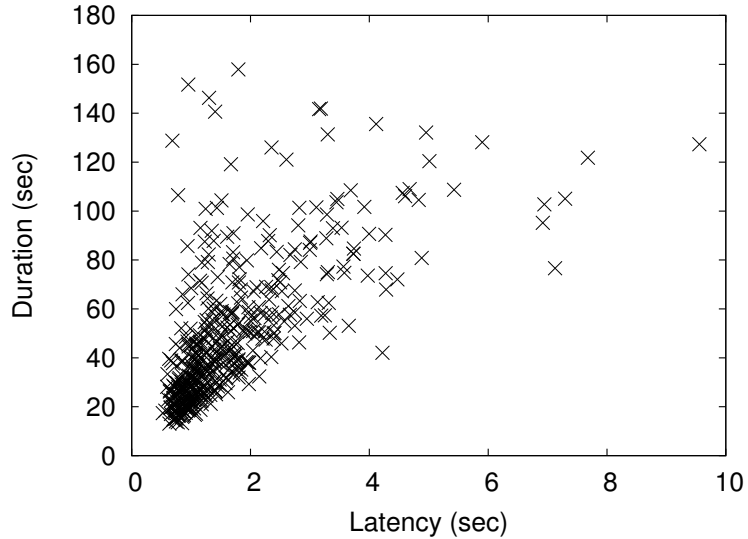
between the throughput of a circuit and the aggregate link distances (delays) between its relays.



**Figure 5-1:** The correlation between the duration to download a file (y-axis) and the sum of the geo-distances between the relays in the circuit (x-axis).

The second question to consider is whether the latency of a circuit is correlated to its throughput. We define *circuit latency* to be the time it takes to download a minimal-size (one-byte) file using that circuit. Clearly, latency captures the delays due to links (propagation delay) as well as relays (packet processing delays). Figure 5.2 shows the relationship between 3MB download times (as proxy for circuit throughput) on the y-axis and latencies on the x-axis. The correlation is evident as  $r$  equals 0.6553, suggesting that using latency as a first-order approximation of download duration is justifiable.

The third question is whether the throughput of a circuit is correlated with the normalized capacities of the constituent relays. Figure 5.3 shows the relationship between the throughput of a circuit in Mbps (x-axis) and the minimum of the consensus capacity estimates of the relays that make up the circuit (y-axis). The correlation is weak, as  $r = 0.1496$ , suggesting that Tor’s three-pronged approach (using the probing, feedback control, and circuit construction mechanisms) to distribute load results in a good utilization across relays. In (Wang et al., 2012), the authors considered this lack of correlation as an evidence of failure. On the contrary, we argue that the

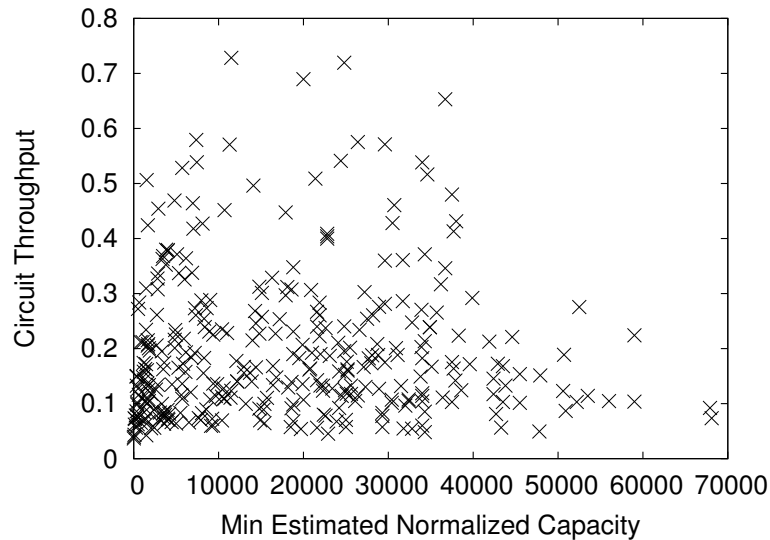


**Figure 5-2:** The correlation between the duration to download a file (y-axis) and the delay to download just one byte using the same circuit (x-axis).

lack of correlation is indicative of evenly utilized relays, which is one of the goals of the system.

The fourth question we consider is whether the clients' experience (the throughput observed by different clients or by the same client over time) is consistent. This is a fairness question, corresponding to a recurring complaint from Tor clients that sometimes they are stuck with low-throughput circuits (Dingledine and Murdoch, 2009). Here we note that even though relays may be evenly utilized (as suggested by the results in Figure 5-3), it might be the case that the constructed circuits exhibit highly uneven performance. For the circuits we constructed, we noticed that the coefficient of variation (the standard deviation divided by the mean) of their achieved throughput is large –  $CV = 0.62576$  – confirming that the fidelity (measured in terms of achievable bit rates) of Tor circuits is highly unpredictable.

The last question we consider is whether Tor's three-pronged approach to load distribution yields any particular probability distribution function (PDF) of relay available capacities. Recall that the available capacity of a Tor relay is the expected throughput of a circuit traversing that relay. For a fully-utilized relay, the available capacity is nothing other than the max-min fair-share for a circuit going through that relay. For a relay that is not fully utilized, the available capacity is

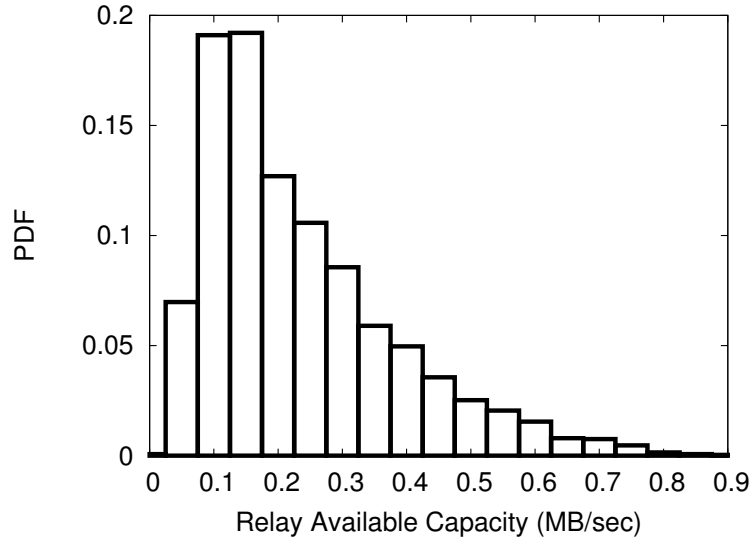


**Figure 5-3:** The correlation between the throughput of a circuit (y-axis) and the raw capacity of its constituent relays (x-axis).

the expected throughput of a newly created circuit going through that relay. To characterize relay available capacities, we construct a special two-relay circuit where the exit node is our local BU Tor relay `BostonUCompSci`. The circuit downloads three files of size 3MB each from a powerful BU server to a powerful BU client. This set-up is our best attempt to make sure that the circuit bottleneck is the entry relay. The average throughput of the circuit is recorded for each of the 2,917 relays that were present in the Tor network at the time of the experiment. Figure 5-4 shows the PDF of relay available capacities. It suggests that most relays have small available capacities, and that the number of relays with higher available capacity decays at an exponential rate.<sup>2</sup>

**Summary:** Our live measurements on Tor lead us to the following conclusions: (1) the throughput of a Tor circuit can be assumed to be largely independent from the geographical link-distance between relays in that circuit; (2) the latency of a circuit and its throughput are inversely, strongly correlated; (3) the throughput of a circuit is largely independent of the overall (normalized) capacity of its relays, suggesting that Tor is relatively efficient in its use of resources by keeping an

<sup>2</sup>This is important as it allows us to assume that relay available capacities follow an exponential distribution in Section 5.4.1.



**Figure 5-4:** PDF of relay available capacities, decaying exponentially.

even utilization of relays; (4) the throughput of different circuits vary greatly, suggesting that some clients suffer significant performance degradation; and (5) the distribution of the relay available capacity can be approximated using an exponential distribution.

#### 5.4 Tor Circuit Throughput: Analytical Model

Assuming that relay available capacities follow a certain probability distribution,<sup>3</sup> can we characterize the probability mass function of the throughput of a circuit constructed as a result of Tor (current or proposed) circuit construction mechanisms? In this section, we develop an analytical model that allows us to obtain such characterization, and consequently quantify the performance implications from variants of Tor circuit construction mechanisms, as well as variants of Tor probing mechanism.

---

<sup>3</sup>We derive two models: the first is under the assumption (verified through measurement) that relay available capacities are exponentially distributed, whereas the second is under the assumption that they are uniformly distributed.

#### 5.4.1 Variability of Tor Circuit Throughput

As previously defined, the available capacity of a relay is the expected throughput of a circuit traversing that relay. In this subsection, we assume that the available capacity of a relay follows an exponential distribution with parameter  $\lambda$ :  $f(x) = \exp(\lambda) = \lambda e^{-\lambda x}$ . The throughput of a circuit is the minimum throughput of its three constituent relays. From the literature of order statistics (David and Nagaraja, 1970), the minimum of three random variables is a random variable of density function:

$$\begin{aligned} f_{1:3}(x) &= 3(1 - F(x))^2 f(x) = 3(1 - (1 - e^{-\lambda x}))^2 * (\lambda e^{-\lambda x}) \\ &= 3\lambda e^{-3\lambda x} = \exp(3\lambda) \end{aligned}$$

which is itself an exponentially distributed variable with a parameter three times the one for the relay available capacity. Knowing that the mean and standard deviation equal  $\frac{1}{3\lambda}$ , the coefficient of variation (CV) would equal one. This is a surprising result, CV is independent from  $\lambda$ , which suggests that if we manage to decrease the skewness in the distribution of the relay available capacity, the variability experienced by clients (due to different circuit capacities) would stay the same. We attribute this result to the memoryless nature of the exponential distribution.

#### 5.4.2 Effect of “Best-Of-K” Circuit Selection

Tor clients typically create between two and twelve circuits and choose the one to use at random. In this section, we ask the question what if each client uses the best (as opposed to a random) circuit from the set of circuits it created? Doing that would eliminate slow circuits, such as those observed by Dhungel in (Dhungel et al., 2010). Since circuit throughput is an exponentially distributed random variable with parameter  $3\lambda$ , order statistics tells us that the maximum of  $k$  such circuits is also a random variable with PDF:

$$f_{k:k}(x) = k(F(x))^{k-1} f(x) = k(1 - e^{-3\lambda x})^{k-1} 3\lambda e^{-3\lambda x}$$

This random variable has an expected value:

$$\mu_k = \int_0^{\infty} x f_{k:k}(x) dx = \int_0^{\infty} x k (1 - e^{-3\lambda x})^{k-1} 3\lambda e^{-3\lambda x} dx = \frac{H_k}{3\lambda}$$

where  $H_k = \sum_{i=1}^k \frac{1}{i}$ . The detailed derivation is in Appendix 7.1.

Similarly, we can compute the variance of the random variable governing the circuit throughput:

$$\begin{aligned} var_k(x) &= \int_0^{\infty} (x - \mu_k)^2 f_{k:k}(x) dx \\ &= \int_0^{\infty} \left(x - \frac{H_k}{3\lambda}\right)^2 k (1 - e^{-3\lambda x})^{k-1} 3\lambda e^{-3\lambda x} dx = \frac{b_k}{\lambda^2} \end{aligned}$$

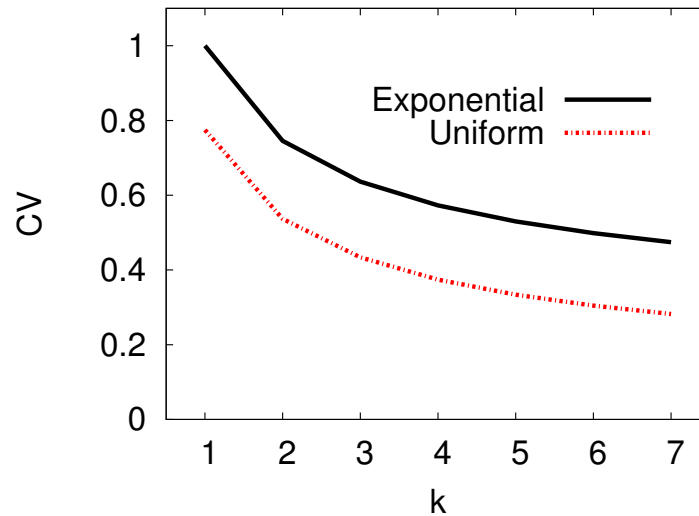
where  $b_k = \frac{2}{9} \sum_{i=1}^k \frac{1}{i} H_i - \frac{H_k^2}{9}$ . The detailed derivation is in Appendix 7.2.

While the coefficient of variance  $CV_k = \frac{\sqrt{var_k(x)}}{\mu_k} = \frac{\sqrt{b_k}}{H_k/3}$  is independent of  $\lambda$ , the skewness of the relay available capacity distribution, it does depend on  $k$ , the number of circuits from which the best is chosen. Figure 5-5 shows CV as a function of  $k$ . CV decreases with the increase in the number of circuits,  $k$ .

Figure 5-5 also shows CV as a function of  $k$ , under the assumption that the relay available capacities follow a uniform distribution  $U(a, b)$ . In this case,

$$\begin{aligned} \mu_k &= b + (b - a) \sum_{i=0}^k C_i^k \frac{(-1)^{i+1}}{3i + 1} \\ var_k(x) &= (b - a)^2 \left( 2 \sum_{i=0}^k \left[ C_i^k \frac{(-1)^i}{3i + 2} \right] - \left[ \sum_{i=0}^k C_i^k \frac{(-1)^{i+1}}{3i + 1} \right]^2 \right) \end{aligned}$$

. See Appendix 7.3 for the detailed derivation. Again, the CV of circuit throughput decreases with  $k$ . This result suggests that choosing the best-of- $k$  circuit is not a mere byproduct of assuming an exponential distribution for the relay available capacities, thus it is a recommended practice. We verify this result through simulation in the next section.



**Figure 5-5:** The CV of the throughput of the best-of- $k$  circuit.

### 5.4.3 Effect of Filtering out Probes with Below-Average Throughput on the Performance of the PID Controller

As we discussed earlier, the Directory Authority (DA) probes all relays regularly and feeds the PID controller with the average throughput of (typically five) probes as an estimate for the relay available capacity. The designers of the PID controller were concerned that some bad relay pairings would result in probes with very low throughput, lowering the estimate of available capacity, thus skewing the feedback signal to the PID controller. To mitigate this, the average of the five probes is calculated, and probes with below-average throughput are filtered out. The PID controller is fed with the *filtered* average probe throughput as the estimate for available capacity. This probe filtering mechanism turns out to have the inverse effect of what it was intended to achieve, resulting in an average estimate that is heavily skewed towards the maximum throughput (indeed, mostly equal to it).<sup>4</sup>

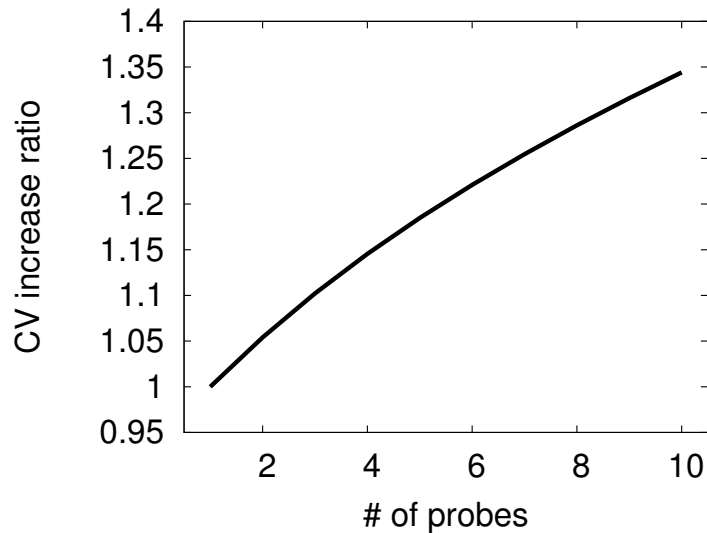
<sup>4</sup>Using our simulator, described in Section 5.5, we noticed that typically, one of the five probes has a throughput that is significantly higher than the other probes. As a result, this maximum probe is the only one with throughput above the average probe throughput before filtering, effectively equating the filtered average with the maximum of the probes throughput.

To quantify the impact of probe filtering, we analytically derive the CV of the signal sent to PID controller with and without probe filtering. The estimate of the relay normalized capacity is  $EST_i^t = EST_i^{t-1} * \frac{F_i}{\sum_{j \in R} \{F_j\} / |R|} \quad \forall i \in R$ , where  $F_i$  is a random variable representing the throughput of the average-of- $l$  (no filtering) or max-of- $l$  probe (filtering). A good controller would have the feedback signal:  $s = \frac{F_i}{\sum_{j \in R} \{F_j\} / |R|}$  approach one quickly. For a large set of relays, it is safe to assume that  $\sum_{j \in R} \{F_j\} / |R| \simeq E(F_i)$ . The feedback signal is a random variable  $s = \frac{F_i}{E(F_i)}$  with a mean of one and a standard deviation of  $\frac{\sigma(F_i)}{E(F_i)}$ . Hence,  $CV = \frac{\sigma(F_i)}{E(F_i)}$ . In the case of filtering,  $F_i$  is the max-of- $l$ , which is mathematically identical to the derivation of best-of- $k$  circuit selection in the previous section. In this case  $CV = \sqrt{\sum_{i=1}^l \frac{2}{i} H_i - H_l^2} * \frac{1}{H_l}$ , where  $H_l = \sum_{i=1}^l \frac{1}{i}$ . In the case of no filtering,  $F_i$  is the average-of- $l$  probes, each is exponentially distributed, which is an Erlang- $l$  distribution divided by  $l$  with a mean of  $\frac{1}{2\lambda}$  and standard deviation of  $\frac{\sqrt{l}}{l * 2\lambda}$ . Hence  $CV = \frac{1}{\sqrt{l}}$ .

Figure 5-6 plots the ratio between the CV of the signal sent to the PID controller with and without filtering. It suggests that the current version of Tor, with  $l = 5$ , feeds the PID controller a feedback signal that is 18.5% noisier (with higher variability) than our proposed version without filtering. Such a noisy feedback signal would negatively impact the normalized relay capacity estimates produced by the PID controller and, consequently, compromising the efficiency of the whole system.

## 5.5 Performance Evaluation

To confirm the conclusions from the analytical models in Section 5.4 under relaxed assumptions (and additional parameters), to capture the feedback dynamics between Tor's probing, feedback control, and circuit construction mechanisms, and to evaluate our proposed improvements to these mechanisms, we built our own Tor simulator. In this section, after describing our simulator, we show a representative set of the experiments we conducted that give more insights into the behavior of Tor, and confirm our findings.



**Figure 5-6:** The ratio between the CV of the signal sent the PID controller with filtering and without filtering as a function in the number of probes.

### 5.5.1 The Simulator

Our simulation model consists of three modules. The first module mimics the behavior of a large number of clients. Each client constructs a circuit, using three relays out of the available relays. The second module assigns throughput to constructed circuits given max-min fairness<sup>5</sup> and estimates relay available capacities, which determines the throughput available to a new circuit (probe). The third module models Tor’s measurement infrastructure, which regularly probes all the relays, and uses PID control to update the normalized capacity estimates, hourly.

These three modules greatly influence each other as constructed circuits affect the throughput of the measurement probes, which in turn are used by Tor’s PID controller to estimate each relay’s normalized capacity, to be used by clients to bias their relay selection for circuits constructed in the next hour. Thus, over time, the simulator captures the change in relay capacity estimates and its effect on the achieved throughput of the constructed circuits. In particular, we adopt a quasi-static discrete-time simulation model wherein probing is performed at each time step (representing, say,

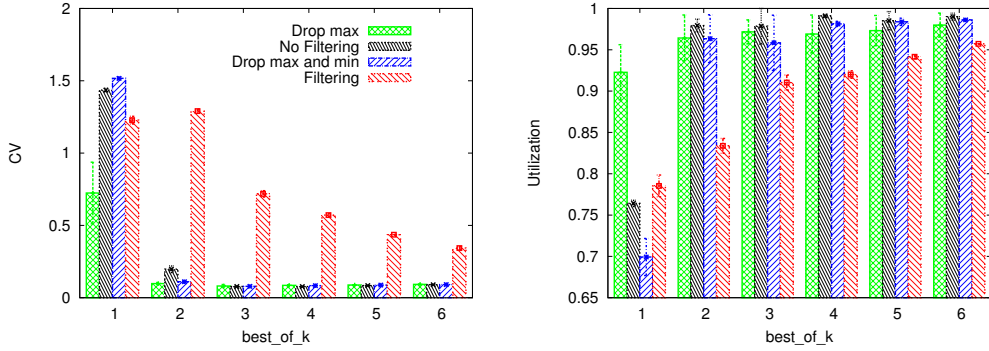
<sup>5</sup>Max-min fairness is typical of how resources are shared on the Internet.

12 minutes), and weights for circuit selection are updated every  $n > 1$  steps. We take  $n = 5$  (representing an hour), and we assume that the system reaches steady state between time steps so circuits achieve their steady-state max-min fair share.

**Circuits Construction:** To model the circuits selected by Tor clients at a certain point in time, we create a certain number of circuits  $M$  at each time step, with a default of  $M = 120,000$ . We choose a circuit’s constituent relays proportionally to their PID-provided capacity estimates as described in Section 5.2. To simulate our modification of choosing the best-of- $k$  circuit, a client builds  $k$  candidate circuits, compares their anticipated throughput, and chooses the best, with  $k$  between one and six. Because the selection of the best circuit depends on observed performance of already established circuits, while constructing the  $M$  circuits, we re-estimate the relay available capacities, and hence throughput of established circuits, every  $m < M$  circuits, with a default of  $m = M/10$ . As noted earlier, the throughput of circuits is calculated assuming a max-min fair capacity allocation.

**The PID Controller:** As described in Section 5.2, every hour, Tor probes each relay a number of times, at least five by default. These probes provide the PID controller with an estimate of each relay’s available capacity, *i.e.*, throughput of a circuit passing through the relay. If the average probe throughput for a relay is higher (lower) than the average for all relays, this indicates that this relay is relatively under-utilized (over-utilized), and the PID controller increases (decreases) its estimate of the relay normalized capacity and hence the relay’s selection weight, enticing more circuits to (not) use this relay.

As noted earlier, we simulate probing at each time step. The probes are conducted over two-relay circuits. A probe throughput is computed in the same way as a circuit throughput, based on max-min capacity allocation. After probing all relays, we apply a filtering technique to drop some (or none) of each relay’s probes. Tor’s default behavior is to drop probes that have throughput that is below the average of all probes. Besides Tor’s default behavior, our simulator supports: (1) *no filtering*, *i.e.*, use all probes; (2) drop the probe with the maximum throughput for each relay; or (3) drop the minimum and the maximum throughput probes for each relay. The PID updates its estimate of each relay normalized capacity (weight) and publishes it to clients (cf. Section 5.2).



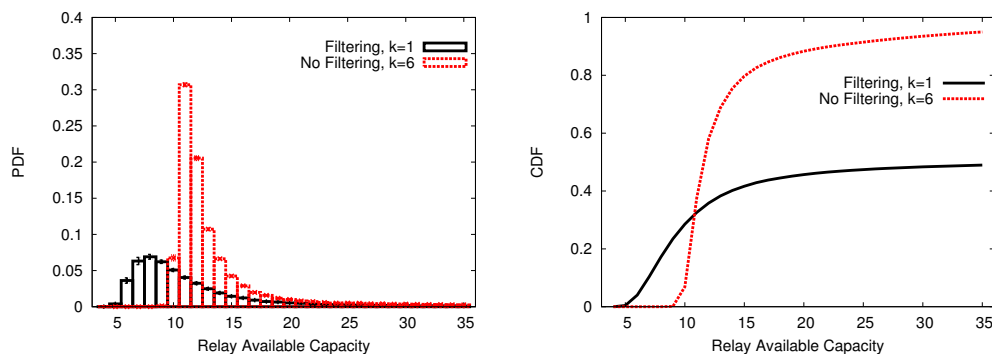
**Figure 5-7:** The performance of different probe filtering techniques against an increasing number of circuits from which the best is chosen.

To obtain performance metrics, each simulation runs for a duration of  $25 \times n$  time steps, equivalent to updating circuit selection weights 25 times. Our simulation results follow.

### 5.5.2 Simulation Results

The base model for the following experiments mimics Tor’s current default behavior. In it, a client constructs only one circuit and all the below-average probe values are filtered out. The number of probes is five. The relay capacities follows a bounded-Pareto distribution between 1,000 Kbps and 1,000,000 Kbps and a shape parameter  $\alpha = 2$ . We vary the number of relays (around 2,000 relays) while keeping the sum of the relay capacities (the aggregate capacity of the Tor network) equal to 3,900,000 Kbps. The number of circuits constructed is 120,000. Ideally, the capacity of relays would be assigned to the circuits allowing each one of them to have a throughput of around 35 Kbps. We set a limit on the throughput of each circuit that is uniformly distributed between 70 and 100 Kbps. This limit is introduced to mimic Tor’s flow-control window mechanism (AlSabah et al., 2011b), which limits the throughput of circuits to protect the Tor network from bandwidth-intensive clients like P2P applications.

The first experiment compares different probe filtering techniques against an increasing number of circuits  $k$  from which the client chooses the best. In Figure 5-7-left, the y-axis is the coefficient-of-variation (CV) of the constructed circuits, while the x-axis is the number of circuits from which the client chooses the best. When  $k = 1$ , the current Tor filtering technique (*i.e.*, drop



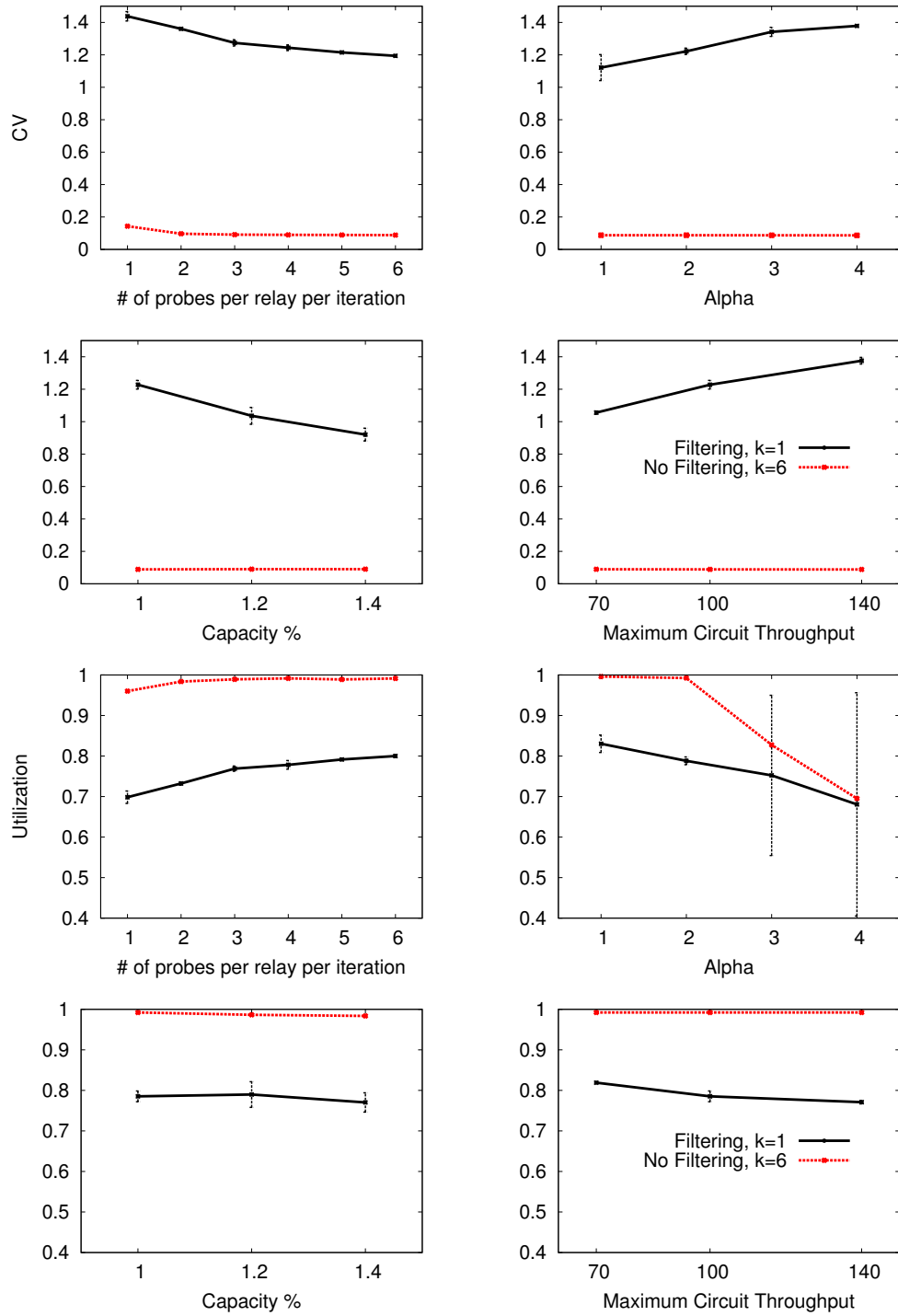
**Figure 5-8:** The PDF and CDF of relay available capacities under two configurations: (1) Filtering,  $k = 1$ , and (2) No filtering,  $k = 6$ .

probes that are below average) is outperformed by “drop max” but still beats the “no filtering” technique. For  $k > 1$ , “no filtering” outperforms other filtering techniques. Figure 5-7-right shows the aggregate utilization of relays (on the y-axis) with similar conclusions. Henceforth, the results are shown in a pair of graphs: one CV on the y-axis, and another utilization. All results show 95% confidence intervals.

For the same experiment, in Figure 5-8, we plot the PDF and CDF of the relay available capacities for two configurations: (1) vanilla Tor configuration with filtering deployed and  $k = 1$ , and (2) our proposed configuration with no filtering and  $k = 6$ . This result validates the model in Section 5.4 in that the available relay capacity decays exponentially. It is worth noting though that vanilla Tor has a heavier tail, indicating that a few relays are left with a lot of available capacity, indicating inefficiency (bad load balancing).

The following set of experiments examine the effect of changing certain parameters on Tor performance. The first is increasing the minimum number of probes that Tor conducts per relay. Figure 5-9 shows that as the number of probes increases, the utilization increases and the variation in client experience (circuit throughput) decreases. The improvement is limited though.

The next experiment studies the effect of changing the shape parameter  $\alpha$  used to generate the relay capacities. It is important to note that we kept the aggregate capacity the same. Thus, a higher  $\alpha$  results in fewer relays with more variable capacities. The higher variability of relay capacities results in a reduction in utilization (with wider confidence intervals) and also in more



**Figure 5-9:** The effects of various parameters on CV of circuit throughput (top) and utilization of relays (bottom).

variability in the throughput of constructed circuits.

The next experiment considers the effect of increasing the aggregate capacity of the Tor network. The x-axis shows the factor by which the aggregate capacity of all relays is increased, *e.g.*, by a factor of 1, 1.2 and 1.4. The increased aggregate capacity translates into an increase in the number of relays, while the number of circuits remains the same. This does not seem to affect the utilization of relays but it lowers the CV of circuits, due to the decreased load on each relay (less circuits per relay).

Lastly, we study the effect of increasing the bound on the allowable circuit throughput. The average throughput is 35 Kbps, and we set the upper limit to be uniformly distributed  $U(70,70)$ ,  $U(70,100)$ ,  $U(70,140)$ . As expected, allowing circuits to have higher throughput results in higher circuit throughput CV without a significant effect on the utilization of the relays.

## 5.6 Bandwidth Boosting using Multi-Path Routing

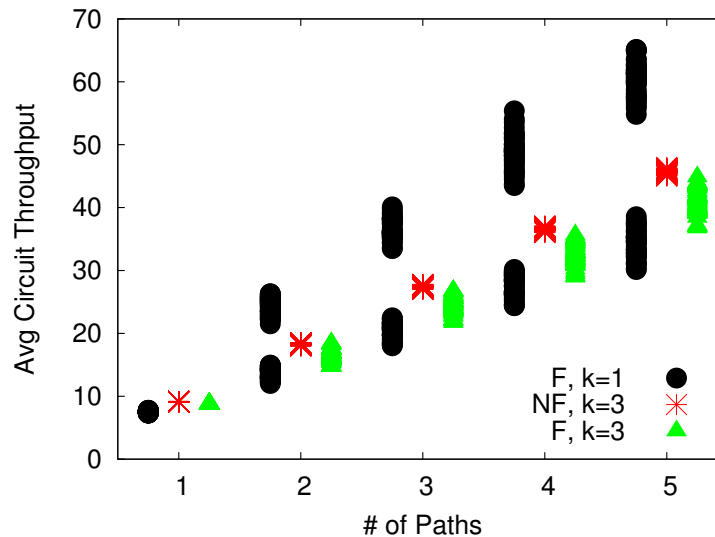
So far, our effort has been to increase the utilization of the relays in the Tor network while limiting the unpredictability of clients' circuit throughput. In this section we propose enhancing the performance of certain clients using angels. Angels are special exit nodes with a lot of capacity. However, the throughput of a circuit is governed by the minimum available capacity of its constituent relays. To mitigate this problem, we adopt an approach originally proposed by AlSabah *et al.* (AlSabah et al., 2011a). This approach uses multi-path forwarding to enhance the performance of constructed circuits. Figure 1.1 illustrates the idea. The angel (the exit node), is responsible for fetching content from, say, a web server and routing it back to the client. The angel can split the reply into segments, routing each through a different reverse path. The client receiving these segments reorders the received cells to reconstruct the original content (before being split by angels). For this to work, the client needs to actively construct these different paths and to inform the angel that these paths belong to the same circuit. This can be achieved through a nonce. When a path is extended to the angel, it will have a random nonce, if the angel has another concurrent circuit with the same nonce, it knows that both are paths of the same circuit. It is important to note that this capability does not require any modification to regular Tor relays (*i.e.*, relays that are

not angels). Indeed, regular relays cannot even discern whether or not a circuit going through it is part of a multi-routed circuit. More importantly, angels performing the split functionality cannot glean any information pertaining to the identity of the client – only that some anonymous client wants to use a multi-path circuit. In this section, we show analytically and verify experimentally that angel-assisted multi-path circuit construction increases the throughput for clients and reduces the variance of constructed circuits. Analytically, we show that if all clients use multi-paths, the variance in their throughput will decrease.

We consider a multi-path enabled Tor network. Let there be  $n$  circuits at a certain point in time where each circuit has  $m$ -paths. Also, let the aggregate capacity of angels be  $C_a$  and the throughput of path  $i$  of circuit  $j$  be  $t_i^j$ . Consequently, the sum of the throughput of all paths of all circuits must not exceed the angel capacity, *i.e.*,  $\sum_{i=1}^m \sum_{j=1}^n t_i^j \leq C_a$ . Assuming  $t_i^j$  follows an exponential distribution, then its parameter must be  $\lambda = \frac{m*n}{C_a}$ . Thus the throughput of a circuit is the sum of the throughput of its paths which is an Erlang distribution (sum of exponentials) with variance  $= \frac{m}{\lambda^2} = \frac{C_a^2}{m*n^2}$ . For vanilla Tor, *i.e.*, no multi-path routing, the sum of the throughput of all circuits must not exceed the angel capacity, *i.e.*,  $\sum_{j=1}^n t^j \leq C_a$ . Assuming  $t^j$  follows an exponential distribution, then its parameter must be  $\lambda = \frac{n}{C_a}$ , and its variance  $\frac{1}{\lambda^2} = \frac{C_a^2}{n^2}$ . This suggests that the variance in circuit throughput in case of multi-path construction is  $\frac{1}{m}$  that of the vanilla Tor protocol.

The above analysis assumes that all clients use multi-path routing with the same number of paths. To evaluate the effect of deploying a limited number of angels, catering to a fraction of the traffic, we revert to simulation. We set up our simulator in such a way that 10% of the aggregate capacity of the network is provided through angels capable of the splitting functionality. Clients equipped with bandwidth-boosting capabilities inform angels acting as exit relays of the number of paths they require using special bit marking when setting up a circuit. In our simulation, this is set up as a uniformly random integer between one and five. We evaluate multi-path routing under three configurations: (1) “F, k=1” is Tor’s default behavior, using probe filtering and best-of-1 circuit selection, (2) “NF, k=3” uses no filtering and best-of-3, and (3) “F, k=3” uses filtering and best-of-3.

Figure 5-10 is a scatter plot showing the results. On the x-axis is the number of paths a circuit is requesting and the y-axis is the average circuit throughput. The “NF, k=3” setting provides the highest utilization and the most predictable results. For example the relay utilization is 99% while the average circuit throughput when the number of paths is five is 4.99 (almost five) times the throughput of circuits with one path. The CV of constructed circuits is between 0.079 and 0.045 (near zero). These results should be contrasted to the “F, k=1” setting, where the relay utilization is 0.78 and the CV is between 1.2 and 0.6. The third “F, k=3” setting corresponds to the case when we have no influence on the directory authority (which uses filtering), with clients choosing the best-of-3 circuits, with multi-path forwarding enabled through angels. In this case the relay utilization is 0.91 and the CV is between 0.78 and 0.38. These results highlight the premise of multi-path angel-enabled routing as explained at the outset.



**Figure 5-10:** The effects of splitting a circuit into multi-paths.

## 5.7 Summary of Contributions

In this chapter, we set out to understand the behavior of Tor and how it could be possibly influenced to yield better, more predictable performance. We started with a characterization of throughput

and delay performance for small and large data transfers, which we correlated to a number of relay/circuit attributes. We concluded that the mechanisms that Tor employs to distribute load across relays achieve the basic goal of high relay utilization, but that they leave much to be desired in terms of predictability due to the high variability of circuit throughput.

Armed with these observations, we used order statistics to analyze the coefficient-of-variation (CV) of throughput achieved by circuits, under Tor's default random circuit selection versus our proposed best-of- $k$  strategy, and also under Tor's approach to probe filtering (of relay available capacity measurements) versus other variant probe filtering strategies, including no-filtering. We analytically showed a lower CV under the no-filtering and best-of- $k$  approaches. We then validated this using a simulation model that captures the feedback dynamics between the processes responsible for measuring/estimating relay available/normalized capacities, and the circuit construction mechanism used by clients. To further improve performance, we proposed and evaluated the deployment of special relays (angels) capable of supporting multi-path routing of some client circuits. We show that even without disabling Tor's default filtering, we can achieve lower CV, and scalable higher throughput to multi-path capable, throughput-sensitive clients.

A future research direction is to incorporate these improvements into the Tor system, by modifying only angel relays and clients empowered to use multi-path circuits, and by assessing the effectiveness of these improvements in the presence of legacy relays and clients. Another research direction of ours is to evaluate the effectiveness of angels in mitigating timing attacks, which create congestion to de-anonymize their victims. By alleviating this congestion, angels have the potential to improve the anonymity characteristics of Tor.

## Chapter 6

### Summary

In this thesis, we presented a framework for deploying on-demand cloud resources (angels) into peer-assisted content delivery systems. The P2P paradigm lends itself to applications where a large number of clients are interested in fresh content that cannot be cached, straining the resources of the content delivery network. We noted that most clients suffer from asymmetric Internet connectivity, *i.e.*, their upstream capacity is but a fraction of their downstream capacity. In aggregate, this capacity gap results in limited quality for clients especially the ones that want to obfuscate their identity. Our solution, as presented in this thesis, was to study such systems and develop a coordinated mechanism to most efficiently utilized the clients' resources. When the clients' resources are not enough to maintain the clients' desired level of service, we deploy resources, on-demand from the cloud (angels), in the same judicious manner we utilize clients' resources. We addressed three applications; (1) delivering bulk-synchronous content: where a large number of clients need to wait for each other to finish download before anyone can take advantage of the downloaded content, (2) live video streaming, and (3) bandwidth-intensive applications over Tor, the anonymous routing network.

In Chapter 3 we introduced the first application, bulk-synchronous transfer. We extended the analytical model in (Kumar and Ross, 2006) to accommodate added angels. We developed a lower-bound for the maximum time by which all clients finish downloading the content. We developed an optimal dissemination strategy that achieves this bound. Realizing the impracticality of the optimal construction and the need to centrally decide on the clients' connectivity to achieve near-optimal performance, we developed GT – our practical tree-based dissemination strategy. Our simulation shows that our centralized GT strategy outperform well-established distributed peer

and piece selection strategies. CLOUDANGELS is the blueprint of cloud-based service that helps content providers minimize the MDT of content dissemination to a set of clients with the help of angels, we deployed a prototype in Emulab. Our experimental results showed that we achieve better MDT than BitTorrent as well as client-server architectures. It showed also that we are able to efficiently utilize the angel capacity.

In Chapter 4, we introduced our second application, live video streaming. We extended the analytical model in (Kumar et al., 2007) to accommodate angels and find the minimum amount of angel capacity needed for all clients to achieve a desired streaming bit-rate. An optimal construction was developed to achieve this lower-bound. We generalized this optimal construction in *AngelCast*, our multi-tree practical construction. *AngelCast* deploys angels efficiently to maintain the streaming rate at the required level, limits the needed start-up buffer to a logarithmic factor, as a function of the number of clients, and handles churn efficiently through a proactive set of membership management algorithms. We proposed a blueprint of a cloud-based service to help content providers with their live streams through the deployment of angels from the cloud. Our proof-of-concept implementation managed to disseminate and playback a VC-1 encoded live video stream to tens of clients with the help of angels. Our experimental evaluation showed that: (1) our performance is slightly better than SopCast, the widely used P2P live streaming protocol, (2) there is a trade-off between the start-up buffer and frame-drop ratio, and that (3) when the clients' upstream capacities are not enough, *AngelCast* dynamically invokes a minimal number of angels, helping clients maintain their play-out rate. To mitigate skepticism towards churn-handling abilities of centralized peer-assisted content distribution systems, we conducted extensive experimental evaluation. Our findings confirmed that our proactive membership management techniques manage churn with minimal effect on clients' performance.

Chapter 5 studied improving the throughput of bandwidth-intensive application over Tor. We conducted in-vivo measurement study to identify the cause behind the performance uncertainty seen by Tor clients. Our findings suggested that the current three-pronged load-balancing system

in Tor leaves more to be desired. We proposed modifications for the circuit-selection mechanism used by Tor clients as well as the probing mechanism used by Tor's Directory Authority. We used an analytical model based on order statistics to validate these modifications, measuring both fairness as well as efficiency. We used the coefficient of variation (CV) of the throughput achieved by Tor clients as a measure of fairness. We used the unassigned capacity of relays as a measure of the network inefficiency. We built a simulator that accounts for the DA, the set of relays, the set of clients as well as the feedback loop between these three groups. Our simulation results showed that our modifications can enhance both the network efficiency as well as provide fairness to clients. Increasing Tor's efficiency and fairness is needed but not enough for bandwidth-intensive applications over Tor. Consequently, we proposed an angel-based, multi-path routing functionality to help clients with bandwidth-intensive applications. Our analytical model as well as simulations showed that we can dedicate more of the added resources towards bandwidth-intensive applications without having to change anything in the Tor network except the angels and the demanding clients.

Our findings in this thesis confirmed our hypothesis that a coordinated approach to optimize the resource deployment of cloud-resources for peer-assisted fresh content delivery systems is attainable.

## Chapter 7

### Appendices

#### 7.1 Calculate $\mu_k$ for Exponential Distribution

$$\mu_k = \frac{1}{3\lambda} \int_0^1 -\ln(1-z) dz^k$$

$$\text{Where : } z = (1 - e^{-3\lambda x})$$

$$\begin{aligned}\mu_k &= \frac{-1}{3\lambda} [\ln(1-z)z^k]_0^1 - \int_0^1 z^k * \frac{-1}{1-z} dz \\ &= \frac{-1}{3\lambda} [\ln(1-z)z^k]_0^1 - \int_0^1 \sum_{i=0}^{k-1} z^i dz + \int_0^1 \frac{1}{1-z} dz \\ &= \frac{-1}{3\lambda} [\ln(1-z)z^k - \sum_{i=0}^{k-1} \frac{z^{i+1}}{i+1} - \ln(1-z)]_0^1 \\ &= \frac{1}{3\lambda} \sum_{i=1}^k \frac{1}{i} + \frac{1}{3\lambda} \lim_{z \rightarrow 1} (1-z^k) \ln(1-z) \\ &= \frac{1}{3\lambda} \sum_{i=1}^k \frac{1}{i} + \frac{1}{3\lambda} \lim_{z \rightarrow 1} \left( \sum_{i=0}^{k-1} z^i \right) \frac{\ln(1-z)}{1/(1-z)}\end{aligned}$$

Using l'Hopital rule by differentiating the nominator and the denominator:

$$\begin{aligned}\mu_k &= \frac{1}{3\lambda} \sum_{i=1}^k \frac{1}{i} + \frac{k}{3\lambda} \lim_{z \rightarrow 1} \frac{1/(1-z)}{-1/(1-z)^2} \\ \mu_k &= \frac{1}{3\lambda} \sum_{i=1}^k \frac{1}{i} = \frac{H_k}{3\lambda}\end{aligned}$$

## 7.2 Calculate $var_k(x)$ for Exponential Distribution

$$var_k(x) = \frac{1}{9\lambda^2} \int_0^1 (-\ln(1-z) - H_k)^2 dz^k$$

$$\text{Where : } z = (1 - e^{-3\lambda x})$$

$$var_k(x) = \frac{1}{9\lambda^2} [(\ln(1-z) + H_k)^2 z^k]_0^1$$

$$+ \frac{-1}{9\lambda^2} \int_0^1 z^k * 2(\ln(1-z) + H_k) * \frac{-1}{1-z} dz$$

$$var_k(x) = \frac{1}{9\lambda^2} [(\ln(1-z) + H_k)^2 z^k]_0^1$$

$$+ \frac{-1}{9\lambda^2} \int_0^1 2(\ln(1-z) + H_k) * \frac{-1}{1-z} dz$$

$$+ \frac{-1}{9\lambda^2} \int_0^1 2(\ln(1-z) + H_k) \sum_{i=0}^{k-1} z^i dz$$

$$var_k(x) = \frac{1}{9\lambda^2} [(\ln(1-z) + H_k)^2 z^k]_0^1$$

$$+ \frac{-1}{9\lambda^2} \int_0^1 2(\ln(1-z) + H_k) * d\ln(1-z)$$

$$+ \frac{-1}{9\lambda^2} \int_0^1 2\ln(1-z) \sum_{i=0}^{k-1} z^i dz + \frac{-1}{9\lambda^2} [2H_k \sum_{i=0}^{k-1} \frac{z^{i+1}}{i+1}]_0^1$$

$$var_k(x) = \frac{1}{9\lambda^2} [(\ln^2(1-z) + 2H_k \ln(1-z))(z^k - 1)]_0^1 + \frac{1}{9\lambda^2} [H_k^2]$$

$$+ \frac{1}{9\lambda^2} \sum_{i=0}^{k-1} \frac{2}{i+1} H_{i+1} + \frac{-2H_k^2}{9\lambda^2}$$

$$var_k(x) = \frac{1}{9\lambda^2} \lim_{z \rightarrow 1} \frac{\ln^2(1-z) + 2H_k \ln(1-z)}{1/(z-1)} \sum_{i=0}^{k-1} z^i$$

$$+ \frac{2}{9\lambda^2} \sum_{i=1}^k \frac{1}{i} H_i + \frac{-H_k^2}{9\lambda^2}$$

$$var_k(x) = \frac{k}{9\lambda^2} \lim_{z \rightarrow 1} \left[ \frac{2\ln(1-z)/-(1-z)}{-1/(z-1)^2} + 0 \right] + \frac{2}{9\lambda^2} \sum_{i=1}^k \frac{1}{i} H_i + \frac{-H_k^2}{9\lambda^2}$$

$$var_k(x) = \frac{2}{9\lambda^2} \sum_{i=1}^k \frac{1}{i} H_i - \frac{H_k^2}{9\lambda^2}$$

### 7.3 The Model Assuming a Uniform Distribution

In this appendix we assume that the relay available capacities follow a uniform distribution:  $f(x) = u(a, b) = \frac{1}{b-a}$ . The throughput of a circuit is the minimum throughput of its three constituent relays. The minimum of three random variables is a random variable of density function:

$$\begin{aligned}
 f_{1:3}(x) &= 3(1 - F(x))^2 f(x) = \frac{3}{(b-a)^3} * (b-x)^2 \\
 F_{1:3}(x) &= \int_a^x f_{1:3}(x) dx = \int_a^x \frac{3}{(b-a)^3} * (x^2 - 2bx + b^2) dx \\
 &= \frac{x^3 - 3bx^2 + 3ba^2 - b^3 + b^3 - 3b^2a + 3ba^2 - a^3}{(b-a)^3} \\
 &= 1 - \left(\frac{b-x}{b-a}\right)^3
 \end{aligned}$$

The expected value of the circuit throughput will be:

$$\begin{aligned}
 \mu &= \int_a^b x f_{1:3}(x) dx = \int_a^b -x d\left(\frac{b-x}{b-a}\right)^3 = \int_1^0 ((b-a)y^{1/3} - b) dy \\
 &= \frac{3(b-a)}{4} [y^{4/3}]_1^0 - b[y]_1^0 = \frac{b}{4} + \frac{3a}{4}
 \end{aligned}$$

The variance:

$$\begin{aligned}
 var(x) &= \int_a^b (x - \mu_k)^2 f_{1:3}(x) dx \\
 &= \frac{3}{(b-a)^3} \int_a^b \left( (x-b) + \frac{3(b-a)}{4} \right)^2 (x-b)^2 d(x-b) \\
 &= \frac{3}{(b-a)^3} \int_{a-b}^0 \left( y + \frac{3(b-a)}{4} \right)^2 * y^2 dy \\
 \text{where } y &= x - b \\
 var(x) &= \frac{3}{(b-a)^3} \int_{a-b}^0 \left( y^4 + \frac{6(b-a)y^3}{4} + \frac{9(b-a)^2 y^2}{16} \right) dy \\
 &= \frac{3}{(b-a)^3} \left[ \frac{y^5}{5} + \frac{6(b-a)y^4}{16} + \frac{9(b-a)^2 y^3}{16 * 3} \right]_{a-b}^0 \\
 &= \frac{3}{80} (b-a)^2
 \end{aligned}$$

The coefficient of variation then equals:

$$CV_k = \frac{\sqrt{var(x)}}{\mu} = \frac{\sqrt{3/80}(b-a)}{b - 3/4(b-a)}$$

Which means that the CV increases with the increase of the width of the support  $[a, b]$ . Thus if angels can be used to decrease this support, the client experience would be more predictable.

In the case of best-of-k circuit selection.

$$f_{k:k}(x) = k(F(x))^{k-1} f(x) = k \left( 1 - \left( \frac{b-x}{b-a} \right)^3 \right)^{k-1} \frac{3(b-x)^2}{(b-a)^3}$$

The mean circuit throughput in this case would be:

$$\begin{aligned}
 \mu_k &= \int_a^b x f_{k:k}(x) dx = \int_a^b x d \left( 1 - \left( \frac{b-x}{b-a} \right)^3 \right)^k \\
 &= \left[ x \left( 1 - \left( \frac{b-x}{b-a} \right)^3 \right)^k \right]_a^b - \int_a^b \left( 1 - \left( \frac{b-x}{b-a} \right)^3 \right)^k dx \\
 &= [b - 0] + \int_1^0 (1 - y^3)^k (b-a) dy
 \end{aligned}$$

Where  $y = \frac{b-x}{b-a}$

$$\mu_k = b + (b-a) \int_1^0 (1-y^3)^k dy = b + (b-a) * a_k$$

Where

$$a_k = \int_1^0 (1-y^3)^k dy = \int_1^0 \sum_{i=0}^k C_i^k (-y^3)^i dy = (-1)^i \sum_{i=0}^k C_i^k \left[ \frac{y^{3i+1}}{3i+1} \right]_1^0$$

$$\mu_k = b + (b-a) \sum_{i=0}^k C_i^k \frac{(-1)^{i+1}}{3i+1}$$

As for the variance of the circuit throughput:

$$\begin{aligned} var_k(x) &= \int_a^b (x - \mu_k)^2 f_{k:k}(x) dx \\ &= \int_a^b (x - \mu_k)^2 d\left(1 - \left(\frac{b-x}{b-a}\right)^3\right)^k \\ &= (b - \mu_k)^2 - \int_1^0 (1-y^3)^k (2(b - (b-a)y - \mu_k))(-b-a) dy \\ &= ((b-a)a_k)^2 - 2(b-a)^2 \int_1^0 (1-y^3)^k (y + a_k) dy \\ &= ((b-a)a_k)^2 - 2(b-a)^2 \int_1^0 y \sum_{i=0}^k C_i^k (-y^3)^i dy - 2(b-a)^2 a_k^2 \\ &= -((b-a)a_k)^2 - 2(b-a)^2 \sum_{i=0}^k (-1)^i \left[ C_i^k \frac{y^{3i+2}}{3i+2} \right]_1^0 \\ &= (b-a)^2 \left( 2 \sum_{i=0}^k \left[ C_i^k \frac{(-1)^i}{3i+2} \right] - a_k^2 \right) = (b-a)^2 * b_k \end{aligned}$$

Where

$$b_k = 2 \sum_{i=0}^k \left[ C_i^k \frac{(-1)^i}{3i+2} \right] - a_k^2$$

The coefficient of variation in this case would be:

$$CV_k = \frac{\sqrt{\text{var}_k(x)}}{\mu_k} = \frac{(b-a)\sqrt{b_k}}{b+a_k(b-a)}$$

If  $a = 0$  then  $CV_k = \frac{\sqrt{b_k}}{(1+a_k)}$ . Figure 5.5 shows that  $CV_k$  decreases in this case as well with the increase of  $k$ , the number of circuits a client chooses from. This results suggests that choosing the best-of- $k$  circuit is a recommended practice, regardless of the skewness of the underlying distribution of the relay available capacities as a uniform distribution is flat compared to the exponential distribution.

## References

- (1995-2010). Instrumented BitTorrent. <https://gforge.inria.fr/projects/bt-instru/>.
- (1998-2012). Akamai Netsession. <http://www.akamai.com/client/>.
- (2001-2012). SopCast: Free P2P Broadcasting. <http://www.sopcast.org/>.
- (2001-2012). Bittorrent DNA Website. <http://www.bittorrent.com/dna/technology>.
- (2002-2012a). The Tor Project, Inc. [www.torproject.org](http://www.torproject.org).
- (2004-2012). Pando Networks Website. <http://www.pandonetworks.com/cdn-peering>.
- (2005-2012). Octoshape. <https://www.octoshape.com/>.
- (2007). BSP Website. [www.bsp-worldwide.org](http://www.bsp-worldwide.org).
- (2008). OpenP4P Website. <http://www.openp4p.net>.
- (2010). Emulab. <https://www.emulab.net/>.
- (2010). PlanetLab. <https://www.planet-lab.org/>.
- (2010). Netflix Tech Blog. <http://techblog.netflix.com/2011/01/netflix-performance-on-top-isp-networks.html>.
- (2012). Global Internet Phenomena Report, 2H 2012.  
[http://www.sandvine.com/downloads/documents/Phenomena\\_2H\\_2012/Sandvine\\_Global\\_Internet\\_Phenomena\\_Report\\_2H\\_2012.pdf](http://www.sandvine.com/downloads/documents/Phenomena_2H_2012/Sandvine_Global_Internet_Phenomena_Report_2H_2012.pdf).
- (2012b). Tor Live Measurements. <https://metrics.torproject.org/>.
- (2012). FreeGeoIP. <http://freegeoip.net/>.
- Abboud, O., Zinner, T., Pussep, K., Sabea, S. A., and Steinmetz, R. (2011). On the impact of quality adaptation in SVC-based P2P video-on-demand systems. In *Proceedings of the second annual ACM conference on Multimedia systems*, MMSys '11, pages 223–232, New York, NY, USA. ACM.
- Aggarwal, V., Feldmann, A., and Scheideler, C. (2007). Can isps and p2p users cooperate for improved performance? *ACM SIGCOMM Computer Communication Review*, 37(3):29–40.
- AlSabah, M., Bauer, K., Elahi, T., and Goldberg, I. (2011a). The path less travelled: Overcoming tors bottlenecks with multipaths. *cacr.uwaterloo.ca/techreports/2011/cacr2011-29.pdf*.

- AlSabah, M., Bauer, K., Goldberg, I., Grunwald, D., McCoy, D., Savage, S., and Voelker, G. (2011b). Defenestrator: Throwing out windows in tor. In *Privacy Enhancing Technologies*. Springer.
- Banerjee, S., Bhattacharjee, B., and Kommareddy, C. (2002). Scalable application layer multicast. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, Pennsylvania, USA.
- Bestavros, A. and Jin, S. (2003). Osmosis: Scalable delivery of real-time streaming media in ad-hoc overlay networks. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 214–219. IEEE.
- Bindal, R., Cao, P., Chan, W., Medved, J., Suwala, G., Bates, T., and Zhang, A. (2006). Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 66, Washington, DC, USA. IEEE Computer Society.
- Carra, D., Neglia, G., and Michiardi, P. (2008). On the impact of greedy strategies in bittorrent networks: The case of bittyrant. In *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*.
- Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., and Singh, A. (2003). Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313. ACM.
- Cataltepe, Z. and Moghe, P. (2003). Characterizing nature and location of congestion on the public internet. In *Computers and Communication, 2003.(ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, pages 741–746. IEEE.
- Chiu, D. M., Yeung, R. W., Huang, J., and Fan, B. (2006). Can Network Coding Help in P2P Networks? In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006 4th International Symposium on*, pages 1–5.
- Choffnes, D. R. and Bustamante, F. E. (2008). Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. *ACM SIGCOMM Computer Communication Review*, 38(4):363–374.
- Chou, P., Wang, H., and Padmanabhan, V. (2003). Layered multiple description coding. In *Proc. Packet Video Workshop*.
- Chu, Y., Rao, S., Seshan, S., and Zhang, H. (2002). A case for end system multicast. *Selected Areas in Communications, IEEE Journal on*, 20(8):1456–1471.
- Cohen, B. (2003). Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer systems*.
- Dabek, F., Cox, R., Kaashoek, F., and Morris, R. (2004). Vivaldi: a decentralized network coordinate system. In *ACM SIGCOMM Computer Communication Review*, pages 15–26, New York, NY, USA. ACM.

- Das, S., Tewari, S., and Kleinrock, L. (2006). The Case for Servers in a Peer-to-Peer World. *Communications, 2006 IEEE International Conference on*, 1:331–336.
- David, H. and Nagaraja, H. (1970). *Order statistics*. Wiley Online Library.
- Dhungal, P., Steiner, M., Rimac, I., Hilt, V., and Ross, K. (2010). Waiting for anonymity: Understanding delays in the tor overlay. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–4. IEEE.
- Dingledine, R. and Murdoch, S. (2009). Performance improvements on tor or, why tor is slow and what were going to do about it. *Online: <http://www.torproject.org/press/presskit/2009-03-11-performance.pdf>*.
- Feng, C., Li, B., and Li, B. Understanding the performance gap between pull-based mesh streaming protocols and fundamental limits. In *INFOCOM '09: IEEE International Conference on Computer Communications*.
- Grube, G., Markison, T., Pendleton, M., and Rybicki, M. (1996). Method and apparatus for allocating carrier channels. US Patent 5,495,483.
- Guo, Y., Liang, C., and Liu, Y. (2008a). dhcps: decentralized hierarchically clustered p2p video streaming. In *CIVR '08: Proceedings of the 2008 international conference on Content-based image and video retrieval*, pages 655–662, New York, NY, USA. ACM.
- Guo, Y., Yu, S., Liu, H., Mathur, S., and Ramaswamy, K. (2008b). Supporting vcr operation in a mesh-based p2p vod system. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 452–457. IEEE.
- He, J., Suchara, M., Bresler, M., Rexford, J., and Chiang, M. (2007). Rethinking internet traffic management: From multiple decompositions to a practical protocol. In *Proceedings of the 2007 ACM CoNEXT conference*.
- Huang, C., Li, J., and Ross, K. W. (2007). Can internet video-on-demand be profitable? In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, Kyoto, Japan.
- Huffaker, B., Fomenkov, M., Plummer, D., Moore, D., and Claffy, K. (2002). Distance metrics in the internet. In *Proc. of IEEE International Telecommunications Symposium (ITS)*.
- Jansen, R., Hopper, N., and Kim, Y. (2010). Recruiting new tor relays with braids. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 319–328. ACM.
- Jin, S., Bestavros, A., and Iyengar, A. (2002). Accelerating internet streaming media delivery using Network-Aware partial caching. *Distributed Computing Systems, International Conference on*, 0:153+.
- Kumar, R., Liu, Y., and Ross, K. (2007). Stochastic fluid theory for P2P streaming systems. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 919–927. IEEE.

- Kumar, R. and Ross, K. W. (2006). Peer-Assisted File Distribution: The Minimum Distribution Time. In *Hot Topics in Web Systems and Technologies, 2006. HOTWEB '06. 1st IEEE Workshop on*, pages 1–11.
- Kumar, R. and Ross, K. W. (2007). Optimal peer-assisted file distribution: Single and multi-class problems. Technical report.  
<http://cis.poly.edu/~ross/papers/MinimumDistributionTime.pdf>.
- Legout, A., Urvoy-Keller, G., and Michiardi, P. (2006). Rarest First and Choke Algorithms are Enough. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*.
- Liu, S., Zhang-Shen, R., Jiang, W., Rexford, J., and Chiang, M. (2008). Performance bounds for peer-assisted live streaming. In *ACM SIGMETRICS Performance Evaluation Review*, volume 36, pages 313–324. ACM.
- Loesing, K., Perry, M., and Gibson, A. (2011). Bandwidth scanner specification.  
[https://gitweb.torproject.org/torflow.git/blob\\_plain/HEAD:/NetworkScanners/BwAuthority/README.spec.txt](https://gitweb.torproject.org/torflow.git/blob_plain/HEAD:/NetworkScanners/BwAuthority/README.spec.txt).
- Michiardi, P., Carra, D., Albanese, F., and Bestavros, A. (2012). Peer-assisted content distribution on a budget. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 56(7):2038–2048.
- Mundinger, J., Weber, R., and Weiss, G. (2008). Optimal scheduling of peer-to-peer file dissemination. *Journal of Scheduling*, 11(2):105–120.
- Nicolosi, A. and Annapureddy, S. (2003). P2pcast: A peer-to-peer multicast scheme for streaming data. In *1st IRIS Student Workshop (ISW03)*. Available at:  
<http://www.cs.nyu.edu/nicolosi/P2PCast.ps>.
- Padmanabhan, V., Wang, H., and Chou, P. (2005). Supporting heterogeneity and congestion control in peer-to-peer multicast streaming. *Peer-to-Peer Systems III*, pages 54–63.
- Padmanabhan, V. N., Wang, H. J., and Chou, P. A. (2003). Resilient peer-to-peer streaming. In *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*. IEEE Comput. Soc.
- Panchenko, A. and Renner, J. (2009). Path selection metrics for performance-improved onion routing. In *Applications and the Internet, 2009. SAINT'09. Ninth Annual International Symposium on*, pages 114–120. IEEE.
- Parvez, N., Williamson, C., Mahanti, A., and Carlsson, N. (2008). Analysis of bittorrent-like protocols for on-demand stored media streaming. In *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '08, pages 301–312, New York, NY, USA. ACM.
- Pastor-Satorras, R. and Vespignani, A. (2007). *Evolution and structure of the Internet: A statistical physics approach. Section 10.2*. Cambridge Univ Pr.

- Peterson, R. S. and Sirer, E. G. (2009). Antfarm: efficient content distribution with managed swarms. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 107–122, Berkeley, CA, USA. USENIX Association.
- Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., and Venkataramani, A. (2007). Do incentives build robustness in bittorrent. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, volume 7.
- Qiu, D. and Srikant, R. (2004). Modeling and performance analysis of BitTorrent-like peer-to-peer networks. *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, 34(4):367–378.
- Reardon, J. and Goldberg, I. (2009). Improving tor using a tcp-over-dtls tunnel. In *Proceedings of the 18th conference on USENIX security symposium*, pages 119–134. USENIX Association.
- Salehi, J., Zhang, Z., Kurose, J., and Towsley, D. (1996). Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. *ACM SIGMETRICS Performance Evaluation Review*, 24(1):222–231.
- Sharma, A., Bestavros, A., and Matta, I. (2005). dpam: a distributed prefetching protocol for scalable asynchronous multicast in p2p systems. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1139–1150. IEEE.
- Sherr, M., Blaze, M., and Loo, B. (2009). Scalable link-based relay selection for anonymous routing. In *Privacy Enhancing Technologies*, pages 73–93. Springer.
- Sherr, M., Loo, B., and Blaze, M. (2007). Towards application-aware anonymous routing. In *Proceedings of the 2nd USENIX workshop on Hot topics in security*, page 4. USENIX Association.
- Smaragdakis, G., Bestavros, A., Laoutaris, N., Byers, J. W., Michiardi, P., and Roussopoulos, M. (2008). Swarming on optimized graphs for n-way broadcast. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 141–145.
- Snader, R. and Borisov, N. (2010). Improving security and performance in the tor network through tunable path selection. *Dependable and Secure Computing, IEEE Transactions on*, (99):1–1.
- Stockhammer, T. (2011). Dynamic adaptive streaming over http-: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144. ACM.
- Sweha, R., Bestavros, A., and Matta, A. (2012a). Enhancing Tor Performance For Bandwidth-Intensive Applications. *Technical Report BUCS-TR-2012-013, CS Department, Boston University*.
- Sweha, R., Ishakian, V., and Bestavros, A. (2011). Angels In The Cloud: A Peer-Assisted Bulk-Synchronous Content Distribution Service. In *IEEE CLOUD'2011: The IEEE International Conference on Cloud Computing*.

- Sweha, R., Ishakian, V., and Bestavros, A. (2012b). Angelcast: Cloud-based peer-assisted live streaming using optimized multi-tree construction. In *Proceedings of the 3rd Multimedia Systems Conference*. ACM.
- Tang, C. and Goldberg, I. (2010). An improved algorithm for tor circuit scheduling. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 329–339. ACM.
- Venkataraman, V., Yoshida, K., and Francis, P. (2006). Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *Network Protocols, 2006. ICNP'06. Proceedings of the 2006 14th IEEE International Conference on*.
- Wang, J., Yeo, C., Prabhakaran, V., and Ramch, K. (2007). On the role of helpers in peer-to-peer file download systems: Design, analysis and simulation. In *In IPTPS'07*.
- Wang, T., Bauer, K., Forero, C., and Goldberg, I. (Feb 2012). Congestion-aware path selection for tor? In *16th International Conference on Financial Cryptography and Data Security*.
- Zhang, X., Liu, J., Li, B., and Yum, T. (2005). Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 13–17.

**Curriculum Vitae**

