COOPERATIVE PROXY CACHING FOR
PEER-TO-PEER TRAFFIC

by

Behrooz Noorizadeh Firouzabadi
B.Sc., Sharif University of Technology, Tehran, Iran, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

© Behrooz Noorizadeh Firouzabadi 2007
SIMON FRASER UNIVERSITY
2007

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.
APPROVAL

Name: Behrooz Noorizadeh Firouzabadi
Degree: Master of Science
Title of thesis: Cooperative Proxy Caching for Peer-to-Peer Traffic

Examinining Committee: Dr. Robert D. Cameron
Chair

Dr. Mohamed Hefeeda, Senior Supervisor

Dr. Ramesh Krishnamurti, Supervisor

Dr. Qianping Gu, Examiner

Date Approved: 31 Dec 07
Declaration of Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada

Revised: Fall 2007
Abstract

Peer-to-peer (P2P) systems currently generate a major fraction of the total Internet traffic, accounting for as much as 60-70% of the traffic in some ISPs. We analyze the potential of cooperative proxy caching for P2P traffic as a means to ease the burden imposed by P2P traffic on ISPs. We perform trace-based simulations to demonstrate: (i) significant improvement in byte hit rate can be achieved using cooperative caching, (ii) simple object replacement policies are sufficient to achieve that gain, and (iii) the overhead imposed by cooperative caching is indeed negligible. In addition, we develop an analytical model to confirm substantial gains from cooperative caching are attainable under wide ranges of traffic and network characteristics. Furthermore, we implement several object replacement policies in a real prototype cache, and we compare their performance using P2P traffic traces collected from a widely-deployed P2P system.

Keywords: Peer-to-Peer, Transparent Proxy Cache, Cooperative Caching, Network Measurement
To my family with love.
“The Internet is the most important single development in the history of human communication since the invention of call waiting.”

— Dave Barry
Acknowledgments

This thesis could not have been written without Dr. Mohamed Hefeeda who not only served as my supervisor but also encouraged me and guided me throughout my academic work in SFU. He patiently show me the way of doing research, never accepting less than my best efforts. Mohamed provided a lot of help during my graduate career, came on weekends before the deadlines and made numerous editing and proof-reading of the material presented in this thesis. I thank him a lot.

I would like to thank my supervisor Dr. Ramesh Krishnamurti and my thesis examiner Dr. Qianping Gu for being on my committee and reviewing this thesis. I would like to thank Dr. Robert Cameron for taking the time to chair my thesis defense. I would also like to extend my gratitude to the faculty and staff in the school of computing science at SFU.

I would like to thank all my friends for their support and help throughout my academic work in SFU, specially: Cheng-Hsin Hsu, who whenever I got stuck with a problem, he was the one knows the solution; and Rahele Salari, who has always encouraged me with her energy and innovation.

Last but certainly not least, I would like to thank my family who inspired me to strive beyond the ordinary, encouraged me, and gave me the confidence I needed to make it through life. This thesis is dedicated to them.
Contents

Approval ii
Abstract iii
Dedication iv
Quotation v
Acknowledgments vi
Contents vii
List of Tables x
List of Figures xi

1 Introduction and Background 1
  1.1 Introduction 1
  1.2 Previous Work 3
  1.3 Thesis Contributions 5
  1.4 Thesis Organization 5

2 Models for Caching P2P Traffic 7
  2.1 Independent Proxy Caches 7
  2.2 Cooperative Proxy Caches in Different ASes 8
  2.3 Cooperative Proxy Caches within the same AS 10
3 Measurement Study and Trace Collection
  3.1 Measurement Study ............................................. 12
  3.2 Trace Collection .............................................. 14
  3.3 Object Popularity ............................................. 15

4 The Potential of Cooperative Caching for P2P Traffic ............................................. 17
  4.1 The Need for Cooperation ....................................... 17
  4.2 Gain from Cooperation among Caches in Different ASes ............................................. 20
  4.3 Gain from Cooperation among Caches within the same AS ............................................. 24
  4.4 Relative Gain from Cooperation by Different ASes ............................................. 25

5 Replacement Policies and Cooperation Overhead ............................................. 27
  5.1 Replacement Policies for Cooperative Caching ............................................. 27
  5.2 Overhead Analysis in Cooperative Caching ............................................. 30

6 Analytical Study ............................................. 32
  6.1 Cost in Independent Caching ............................................. 33
  6.2 Cost in Cooperative Caching ............................................. 33
  6.3 Numerical Analysis ............................................. 35
  6.4 Analysis Validation using Simulation ............................................. 37

7 Implementation in a Cache Prototype ............................................. 39
  7.1 Overview of pCache ............................................. 39
  7.2 Implementation of Replacement Policies ............................................. 43
    7.2.1 Implementation of LRU Policy ............................................. 45
    7.2.2 Implementation of LFU Policy ............................................. 46
    7.2.3 Implementation of LSB Policy ............................................. 46
    7.2.4 Implementation of P2P Policy ............................................. 46
  7.3 Evaluation ............................................. 47
    7.3.1 Setup ............................................. 47
    7.3.2 Results ............................................. 50

8 Conclusions and Future Work ............................................. 53
  8.1 Conclusions ............................................. 53
  8.2 Future Work ............................................. 54
List of Tables

4.1 Summary statistics for the P2P traffic observed by our monitoring node in 10 ASes in the West Coast. ......................................................... 18
4.2 P2P traffic Statistics for five ASes in LA. ........................................ 20
List of Figures

1.1 The proposed two models for cooperative caching of P2P traffic. 3

3.1 We captured the requests sent by peers all around the world. 14
3.2 Popularity of objects in P2P Systems is better modeled by a Mandelbrot-Zipf distribution. Results shown for AS 9406, other ASes have similar behavior. 15

4.1 Locations of peers in two large ASes in the US. 18
4.2 Comparison between independent and cooperative caching for caches deployed in five ASes in LA. Sample results shown for AS 2161 using: (a) offline optimal replacement policy, and (b) an online replacement policy. 21
4.3 Gain from cooperation using cLFU and cLRU replacement policies for: (a) five ASes in NYC and (b) five ASes in LA. 21
4.4 Comparison between independent and cooperative caching for caches deployed in AS 11715 using: (a) offline optimal replacement policy, and (b) an online replacement policy. 23
4.5 Cooperation among several caches deployed within AS 11715. 23
4.6 Relative gain in byte hit rates when 64 ASes of different sizes cooperate with each other. 25

5.1 Comparison between six replacement policies in terms of achieved group (G-BHR) and local (L-BHR) byte hit rates when cooperation is done among caches in different ASes in LA. 28
5.2 Relative gain in byte hit rates when 64 ASes of different sizes cooperate with each other. 30
5.3 Comparison between six replacement policies in terms of traffic overhead. Data shown for five cooperating ASes in (a) NYC, and (b) LA. 31
6.1 Impact of traffic characteristics on the gain from cooperative caching: (a) skewness parameter \( \alpha \), and (b) plateau factor \( q \). ................................. 35

6.2 Impact of various parameters on the gain from cooperative caching: (a) relative cache size, and (b) relative cost of inter-cache link to external link. . . . 37

6.3 Results from simulation using synthetic traces to validate the analytical results for impact of \( \alpha \) and \( q \). ................................. 37

6.4 Results from simulation using synthetic traces to validate the analytical results for impact of cache size and \( \tau \). ................................. 38

7.1 pCache deployment model. ................................................. 40

7.2 Each received request might be truncated into different parts: Cached, Admitted, and Relayed. .......................................................... 42

7.3 Distribution of object sizes in our traces (Verizon), using a windows size of 20 MB to count them. .......................................................... 47

7.4 Byte hit rate under four common replacement policies for different cache sizes in Verizon during 3-month measurement. ........................................... 49

7.5 Comparing 4 replacement policies, the cache size is fixed at 1 TB during 3-month trace measurement. ......................................................... 50

7.6 Comparing 4 replacement policies as the time passes. The cache capacity is fixed at (a) 250GB and (b) 1 TB. ......................................................... 51

7.7 pCache trace-based measurement using 100GB storage, (a) BHR of 3 large ASes using LRU policy (b) BHR of AS 9406 in cache warm up state. ........ 52

7.8 Average lifetime of the evicted objects in the cache. ................................. 52
Chapter 1

Introduction and Background

In this chapter, we provide an introduction to caching in peer-to-peer systems and its importance as well as its various models. Then, we summarize the previous works in this area. Then, we summarize the main contributions of this thesis. Finally, we describe the organization of the thesis.

1.1 Introduction

Peer-to-peer (P2P) systems currently generate a major fraction of the total Internet traffic [9, 30], accounting for as much as 60–70% of the traffic in some Internet Service Providers (ISPs). Furthermore, it is expected that the amount of P2P traffic will even increase in the future [12]. Previous studies, e.g., [14], have shown that the huge volume of the P2P traffic has negative consequences on ISPs, because it multiplies the load on their backbone links and increases the possibilities of network congestion. Increasing the traffic load on ISP networks escalates the costs incurred by ISPs to provision and run their networks. These costs will, eventually, have to be passed to customers. Several economical studies have already reported that university campuses suffer from thousands of dollars every year of additional costs because of the huge bandwidth consumption of P2P applications. In addition, since the Internet is a shared platform, higher chances of network congestion may indirectly degrade the performance of other Internet applications.

Several approaches have been proposed in the literature to reduce the negative impacts of P2P traffic. These include enhancing traffic locality [1, 14] and traffic caching [38]. More aggressive approaches using devices for traffic blocking and shaping have also been used in
practice [22,23]. These, aggressive approaches however may not always be feasible for some ISPs, because many of their clients like to participate in P2P systems and might switch to other ISPs if they were blocked. We believe that multiple approaches will likely be required to mitigate the problems created by the enormous amount of P2P traffic. For example, caching can be used in conjunction with locality-aware neighbor selection algorithms [1] to further reduce the amount of traffic downloaded from sources outside of the local network domain.

We focus on exploring the full potential of P2P traffic caching. In particular, we study the models, benefits and costs of cooperative proxy caching for P2P traffic, where multiple proxy caches cooperate with each other to serve P2P traffic. Caching is a promising approach because objects in P2P systems are mostly immutable [9] and the traffic is highly repetitive [18]. In addition, caching does not require changing P2P protocols and can be deployed transparently from clients. Therefore, ISPs can readily deploy caching systems to reduce their costs. In fact, several commercial P2P caching products have already made it to the market, including CacheLogic [3], PeerCache [24], and Sandvine [28]. Efficient caching algorithms have also been proposed in the literature [11,38]. However, all of these works and products are designed only for independent caches, i.e., caches that are neither aware nor cooperate with each other. Despite its great potential, as will be shown in this thesis, no previous works have proposed or analyzed cooperative caching schemes for P2P traffic, to the best of our knowledge. This is in contrast to the significant research attention that has been paid to the cooperative caching of web traffic, although its gain is only achieved under certain conditions [4,17], and even in these cases the gain may not be significant [39].

In this thesis, we propose two models for cooperative caching of P2P traffic. Fig. 1.1 depicts these two models. The first model enables cooperation among caches that belong to different autonomous systems (ASes) but are located in close geographical proximity, while the second considers cooperation among caches deployed within the same AS. An example of the first model is ASes operating in the same city, where the bandwidth on links within the city is typically more abundant than bandwidth on long-haul, inter-city, links. Cooperation will benefit the cooperating ASes, because it will reduce the cost of using the expensive links. The second model is typical for large ISPs with multiple access/exit points for its clients who are distributed over several locations. The goal is to reduce the load on external, inter-ISP, links, which are the most expensive for ISPs. We analyze the potential gain of cooperative caching for P2P traffic in the above two models. To perform this analysis, we
Chapter 1. Introduction and Background

Conduct an eight-month measurement study on a popular P2P system to collect actual traffic traces for multiple caches. Then, we perform an extensive trace-based simulation study to analyze different cooperative caching schemes. Our results demonstrate that: (i) significant improvement in byte hit rate can be achieved using cooperative caching, (ii) simple object replacement policies are sufficient to achieve that gain, and (iii) the overhead imposed by cooperative caching is indeed negligible.

We develop a simple analytic model to assess the gain from cooperative caching in different settings. Our model accounts for the number of caches, salient P2P traffic features, and network characteristics. Using our model, we confirm that substantial gains from cooperative caching are attainable under wide ranges of traffic and network characteristics.

1.2 Previous Work

Caching of P2P traffic has recently been studied in a number of papers. The benefits of caching P2P traffic have been shown in [18] and [14]. The authors of [18] show that P2P traffic is highly repetitive and therefore amenable to caching. The study in [14] suggests deploying caches or making P2P protocols locality-aware to reduce the load on ISP networks. No caching algorithms were proposed in [18] [14]. Caching algorithms designed for P2P traffic have been proposed in [38] and [11]. In [38] two object replacement algorithms are
suggested: Minimum Relative Size (MINRS) and Least Sent Byte (LSB). The first algorithm evicts the object which has the least cached fraction, and the second evicts the object which served the least number of bytes from the cache. The more recent caching algorithm in [11] is based on segmentation and partial caching of objects, and it was shown to outperform previous algorithms by wide margins.

The authors of [31] propose using already-deployed web caches to serve P2P traffic. This, however, requires modifying the P2P protocols to wrap their messages in HTTP format and to discover the location of the nearest web cache. Given the distributed and autonomous nature of the communities developing P2P client software, incorporating these modifications into actual clients may not be practical. Furthermore, previous works have shown that the object replacement algorithms designed for web traffic may not yield good performance [11]. All the of above works target independent caches and do not consider cooperation among caches to further enhance the byte hit rate.

Cooperative caching for web traffic has been extensively studied, see for example [4,17,39] and the references therein. Using trace-based simulation and analytical analysis, the authors of [39] argue that cooperation yields marginal benefits for web caching. The authors of [17] show that cooperation may be beneficial only in certain environments. The gain from cooperation in web caching is debatable because: (i) web objects are fairly dynamic, (ii) a web proxy cache may be able to store most of the popular objects locally, (iii) the overhead imposed is high relative to object sizes, and (iv) latency could be increased due to looking up and downloading objects from other caches. None of the above reasons exists in the case of cooperative caching of P2P traffic. First, most objects in P2P systems are immutable [9]. Second, because P2P objects are several orders of magnitudes larger than web objects [9,27], it is unlikely that a single cache can hold a reasonable fraction of popular P2P objects to achieve high byte rate. The large object sizes in P2P systems also make the overhead paid to find and retrieve a requested object from other caches negligible. Finally, adding a few hundreds milliseconds of latency to a P2P download is not a critical concern, because many sessions take long periods (minutes and even hours) and they usually run in the background. This is unlike web sessions in which latency is crucial. Therefore, we believe that cooperative caching has a stronger case in P2P systems than it had in the web. We are not aware of previous works that studied the potential of cooperative caching for P2P traffic.

Finally, although several measurement studies have been conducted to analyze various aspects of P2P systems, e.g., [9,15,30], they were not explicitly designed to study caching
of P2P traffic. They also did not provide traces that can be used with different caches. Therefore, we had to conduct our own measurement study. We make the traces available to the research community.

1.3 Thesis Contributions

In this thesis, we analyze the potential gain of cooperative caching in the two cooperation models described in Chapter 2. We also implement our ideas in a real proxy cache prototype for P2P traffic. In particular, the contributions of this thesis can be summarized as follows.

- We perform trace-based simulations to demonstrate that significant improvement in byte hit rate can be achieved using cooperative caching. We propose two cooperation models for proxy caches, cooperation among the ISPs within a small geographic region, and cooperation among caches within a large ISP.

- We demonstrate that simple object replacement policies are sufficient to achieve that gain, and also the overhead imposed by cooperative caching is indeed negligible.

- We develop a simple analytic model to assess the gain from cooperative caching in different settings. Our model accounts for the number of caches, salient P2P traffic features, and network characteristics. Using our model, we confirm that substantial gains from cooperative caching are attainable under wide ranges of traffic and network characteristics. We also validate our analytical model using simulation.

- We implement several object replacement policies in a real prototype cache, and we compare their performance using P2P traffic traces collected from a widely-deployed P2P system.

1.4 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we describe different models for caching P2P traffic. Chapter 3 presents our measurement study and the methods we use to construct traces for different caches. In Chapter 4, we present several trace-based simulation experiments to show the potential of cooperative caching. In Chapter 5, we propose and analyze several object replacement policies for cooperative caching. We also analyze the
overhead introduced because of cooperation among caches. Mathematical models for the cooperative caching of P2P traffic is shown in Chapter 6 and we validate our models using simulation. Chapter 7 presents our implementation of a proxy cache server for P2P traffic. Finally, we conclude the thesis in Chapter 8 and discuss the possibilities for extending this work.
Chapter 2

Models for Caching P2P Traffic

In this chapter, we first elaborate on caching P2P traffic using independent caches. Then, we present the two proposed models for cooperation among caches. These two models will be evaluated throughout the thesis.

2.1 Independent Proxy Caches

As argued in [11, 38], the primary goal of caching P2P traffic is to reduce the load on backbone links, and hence reduce the operational costs of ISPs. To reflect this goal, we choose the byte hit rate as the main performance metric for evaluating caching systems for P2P traffic. Byte hit rate is defined as the number of bytes served from cached content divided by the total number of bytes requested. In addition, unlike the case of caching web traffic, the hit rate may not be well defined in the P2P case [38]. This is because requests in P2P systems are typically issued for segments of objects, not for entire objects. Thus, byte hit rate is more suitable than the hit rate for studying caching of P2P traffic.

In independent caching, a cache is to be deployed near the gateway routers of ASes that choose to employ caching to reduce the burden of P2P traffic. See Fig. 1.1(a), but notice that caches in different ASes work independently from each other. In order to take full advantage of a deployed cache and to avoid modifying the source code of P2P client software, the cache should work in a transparent mode. This is similar to web caching systems such as Squid [34], where the gateway router detects HTTP requests and forwards them to the web cache. Detecting P2P traffic, however, is a bit more involved because many P2P systems use dynamic ports and some of them even encrypt control packets. Nonetheless, there have been
several works on identifying P2P traffic based on application signatures [29] and connection patterns [13]. Many commercially-available P2P traffic shaping and blocking products, e.g. Packeteer [23] and P-Cube [22], already identify most of the packets belonging to P2P systems.

We briefly describe how the cache works based on our running prototype cache (called pCache) [10]. pCache is a transparent cache implemented on top of Linux. It is currently serving real BitTorrent and Gnutella clients running in our lab, with well-defined interfaces to easily add other P2P systems. It implements several replacement policies and new secondary storage management methods optimized for P2P traffic. Requests issued from clients are identified using a P2P detection module located at the gateway router. Identified P2P connections are handed over to pCache. Upon receiving a request, pCache does one of the following. First, if the entire requested byte range is found in the cache, it is served to the client. Second, if the byte range is not in cache and the caching algorithm decides that it is not worth storing locally, the cache forwards the request to the P2P network. In this case, the requested byte range will be sent directly to the requesting peer, not to the cache. Direct transmission of bytes to the requesting peer reduces the load on the cache. Third, if a part of the requested range is found in the cache, this part is served to the client. Then, the caching algorithm decides whether to store more bytes of the requested object or not. In the former case, the cache constructs a request with the missed part of the byte range—with the cache itself as the source—and sends it to the P2P network. While receiving the missed part of the requested byte range, the cache serves it to the client and stores it locally. In the latter case, the cache constructs a request for the missed part with the requesting peer as the source and sends it to the P2P network.

2.2 Cooperative Proxy Caches in Different ASes

The first model for cooperation considered in this thesis is depicted in Fig. 1.1(a). In this model, caches deployed in different ASes cooperate with each other to serve requests from clients in their networks. The ASes are assumed to be located within a reasonable geographical area such as a city or county. Since the cost of links within the city is typically less than the cost of long-haul, inter-city, links, cooperation will benefit the cooperating ASes, because it will reduce the cost of using the expensive links.

Caches cooperating with each other form what we call a cache group. The cooperation
in the cache group works as follows. When a cache receives a request for an object that it
does not store locally, it first finds out whether another cache in the cache group has the
requested object. If any of them does have the object, the object is directly served to the
requesting client. If otherwise, the request is forwarded to external sources as described
in the previous subsection. Communication and object look up inside the cache group
can be done in several ways. For example, a centralized directory can be used, similar to
the CRISP protocol [5,6] proposed for cooperative web caching. The look up process is
straightforward in this case, and it requires only two messages. However, the directory is a
single-point of failure and it does require frequent updates from participating caches. We
adopt distributed look up methods. One distributed look up method is using the Internet
Cache Protocol (ICP) [36,37], which is implemented on top of the open-source Squid web
cache [34]. We note that minor modifications to ICP will need to be made to support the
P2P traffic case. For example, two fields should be added to the query messages of ICP
to indicate the start and end of the requested byte range, because clients in P2P systems
request segments of objects, not full objects at once.

The look up process of ICP adds communication overhead because of the messages
exchanged by the protocol, and it may increase the latency of serving objects. In Sec. 5.2,
we show through simulating the ICP protocol that the communication overhead is negligible.
The added latency (in the order of milliseconds) of the ICP protocol is also negligible. This is
because, unlike web traffic, P2P traffic is less sensitive to latency, since most P2P download
sessions run in the background and they tend to take long periods (in the order of minutes
and sometimes hours).

The above mentioned overhead and latency, although very small, can further be reduced
by using more efficient object look up methods. For example, caches can construct a dis-
tributed hash table (DHT) using CAN [25], Chord [35], Pastry [26], or similar algorithms.
Entries in the DHT will be object IDs and pointers to object locations in the cache group.
Since caches are fairly stable (unlike the clients in P2P systems), DHT protocols will not
suffer from high churn overhead and will, therefore, be preferred for large cooperative cache
groups.

Upon a local miss, a cache using ICP in the cooperation sends a query message to
every other cache in the group [36,37], assuming a flat structure of the cache group. When
receiving a query, ICP requires that a cache replies with hit, miss, or other messages (e.g.,
denied). Hence, the number of messages exchanged to look up an object is $2m$, where $m$ is
the number of caches in the group. Every ICP messages has a 20-byte header and variable-size payload. The payload contains the requested object ID and the requested byte range. Object IDs in P2P systems are typically obtained using hash functions. For example, the Gnutella protocol uses the SHA-1 hash function to generate a 160-bit ID. Thus, the average ICP message size will be around 50 bytes, which is similar to the estimates computed from real traces for cooperative web caching given in [16]. Therefore, the overhead to look up an object is approximately 100m bytes, which is really negligible for large P2P objects. To illustrate, there is about 2 KB overhead to look up an object in a cache group of size $m = 20$.

### 2.3 Cooperative Proxy Caches within the same AS

The second model for cooperation proposed in this thesis is for caches deployed within the same AS. For example, Fig. 1.1(b) shows this model for a large ISP with multiple access/exit points. The network of such ISPs is composed of multiple points of presence (POPs) interconnected with high-speed optical links. ISPs provide Internet access to their customers at POPs. The links inside an ISP are usually over provisioned. ISPs are attached to the Internet through inter-ISP links. Inter-ISP links are usually the bottlenecks of the Internet and where congestion occurs. In addition, the inter-ISP links are expensive because an ISP either pays another ISP for carrying its traffic (in a customer-provider relationship) or it needs to mutually carry the same amount of traffic from the other ISP (in a peer-to-peer relationship) [20]. Deploying cooperative caches in such large ISP would save a huge amount of P2P traffic from going on the inter-ISP links, and thus would reduce the costs incurred by ISPs, because the cost of the internal links (between caches) is much smaller than the cost of inter-ISP links [20]. Caching would also benefit clients because their traffic will traverse fewer inter-ISP links, which are more susceptible to overload and congestion.

As concrete examples for this model of cooperative caching, we show in Fig. 4.1 the distribution of clients in two large ASes in the US: AS 11715 (Cox Communications) and AS 1859 (AT&T-Comecast). We discuss how we created this map in Section 4. Since ISPs provide Internet access to their customers at POPs, they are the natural locations for deploying caches. Therefore, caches would be near client clusters, somewhere inside the rectangles in Fig. 4.1. The cooperation among these caches employs protocols similar to the ones described in the previous subsection.
We note that cooperation among caches within the same AS would be easier to implement in practice than cooperation among caches in different ASes. This is because in the former case all caches are owned and managed by a single entity, while in the latter multiple parties are involved. Moreover, political issues between different parties might affect the decision of enabling or disabling cooperative caching. Nonetheless, we hope that the significant potential gains shown in this thesis will motivate ASes to enable cooperative caching. Finally, we mention that caching of P2P traffic might raise some legal issues, similar to (or probably more than) those raised for caching of web traffic about two decades ago. Addressing these legal issues are outside the scope of this thesis.
Chapter 3
Measurement Study and Trace Collection

We are interested in studying the potential collaboration among caches to reduce the WAN traffic imposed by P2P systems. Ideally, we would like to have a trace showing information about requested objects from each cache. Although several P2P caching products have already been introduced and deployed [3, 24, 28], we are not aware of any public traces that can be used to study caching of P2P traffic. Therefore, we conducted a measurement study to collect and analyze the characteristics of P2P traffic that would be observed by many individual caches. We make these traces available to the research community. We start this section by providing some details on our measurement study. Then, we describe how we created traces for individual caches. Finally, we briefly present a model for object popularity in P2P systems, which will be used in the analysis conducted in this thesis. This measurement study was done by other members of the Network Systems Laboratory at Simon Fraser University [27]. We processed the collected traces to make them suitable for cooperative caching. We present a brief description of the measurement study for the sake of completeness.

3.1 Measurement Study

We conducted a passive measurement study of the Gnutella file-sharing network [8], which is one of the top-three most popular P2P systems [40]. Gnutella has two kinds of peers:
ultra peers, characterized by high bandwidth and long connection periods, and leaf peers which are ordinary peers that only connect to ultra peers. Peers exchange several types of messages including PING, PONG, QUERY and QUERYHIT. A QUERY message contains search keywords, a TTL field and the address of the immediate neighbor which forwarded the message to the current peer. Query messages are propagated to all neighbors in the overlay up to a hop distance specified by the TTL field. If a peer has one or more of the requested files, it replies with a QUERYHIT message. A QUERYHIT message is routed on the reverse path of the QUERY message it is responding to, and it contains the name and the URN (uniform resource name) of the file, the IP address of the responding peer, and the file size.

We modified a popular Gnutella client, called Limewire [19], to run as a monitoring node. We ran our monitoring node in the ultra-peer mode. It passively recorded the contents of all QUERY and QUERYHIT messages passing through it without injecting any traffic into the network. Although we deployed only one ultra peer, we configured it to reach most of the Gnutella network as follows. We increased the number of concurrent connections that it could maintain to be up to 500. A regular ultra peer allows up to 16 connections to other ultra peers and a maximum of 30 to leaf peers. Effectively, our peer is worth more than 20-30 regular ultra peers. There were many time instances when our peer was concurrently connected to more than 350 other ultra peers. Let us assume that each of these 350 ultra peers connect to 10 other ultra peers on average, each of them connect to 10 other, and so on. Given that queries in Gnutella are forwarded up to several (typically 7) hops among ultra peers, our monitoring node was able to capture traffic from a huge number of peers. In addition, our monitoring node ran continuously for eight months, while other peers joined and left the network. This means that the 200—400 other peers connected to our node were continuously changing, which allowed it to reach different and larger portions of the Gnutella network. It is important to emphasize that our monitoring node captured traffic from numerous ASes, not only our local AS. This is because of its high connectivity to many other ultra-peers.

The measurement study lasted for a long period, from 16 January 2006 to 16 September 2006, and collected a huge amount of traffic. During the eight months of the study, we recorded more than 288 million QUERY messages and 134 million QUERYHIT messages issued from approximately 38 million peers distributed over more than 17 thousand different ASes. The total amount of traffic observed was more than 6,000 tera bytes. Fig. 3.1
CHAPTER 3. MEASUREMENT STUDY AND TRACE COLLECTION

Figure 3.1: We captured the requests sent by peers all around the world. demonstrates the wide scale of our measurement.

3.2 Trace Collection

We construct the traces for individual ASes as follows. For a given AS, we use unique QUERYHIT messages to count the number of replicas of each object found in that AS. We divide QUERYHIT messages among ASes based on the source IP addresses contained in them. We use the GeoIP database [7] in mapping IPs to ASes. Most object replicas found in an AS were downloaded sometime in the past, and a cache would have seen a sequence of requests for these objects if it had been deployed in that AS. This assumes that most of the downloads were supplied by peers from outside the AS, which is actually the case because peers in most current P2P networks have no sense of network proximity and thus do not favor local peers over non-local peers. The measurement study in [9] has shown that up to 86% of the locally-available objects were downloaded from external peers. Thus, we construct the sequence of requests from the unique QUERYHIT messages, i.e., the sequence has one request for each replica downloaded by a peer. Peers who replied earlier with QUERYHITs for an object are assumed to have downloaded the object earlier. Notice
that, from the cache perspective, the exact time when the object was downloaded is not important. It is the relative popularity of objects and the distance between similar requests in the trace that matter. These two issues are captured by our sequences. In addition, as discussed below, various P2P traffic characteristics extracted from these sequences in different ASes are similar to those reported in previous studies. Therefore, because of the long duration, huge amount of traffic recorded, and matching results with previous works, we believe that our traces are representative of what would be observed by caches deployed in different ASes.

### 3.3 Object Popularity

An important aspect in caching P2P traffic is object popularity. Several previous works indicate that object popularity in P2P systems deviates from the common Zipf-like distribution, which is usually used to model popularity of web objects [2]. For example, the authors of [9] show that object popularity of P2P objects in one AS domain (campus network) has a flattened head near the low-ranked objects. Objects are ranked based on their popularity, lower the rank higher the popularity. Our own work [11] confirms this flattened-head nature in several ASes, and presents an analytic model that captures this behavior well. We illustrate this model in Fig. 3.2, which shows object popularity in a sample AS from our traces. The figure shows that a Zipf-like distribution will over-estimate the popularity of objects near the head of the curve, which are the most important for the cache. The figure also
shows that a generalized form of Zipf-like distributions, called Mandelbrot-Zipf [32], is a better model for object popularity in P2P systems. Similar results were obtained for other ASes [11]. We will refer to Fig. 3.2 and the Mandelbrot-Zipf model later in the thesis.

The Mandelbrot-Zipf distribution defines the probability of accessing an object at rank $i$ out of $N$ available objects as:

$$p(i) = \frac{K}{(i+q)^\alpha},$$  \hspace{1cm} (3.1)$$

where $K = 1/(\sum_{i=1}^{N} 1/(i+q)^\alpha)$, $\alpha$ is the skewness factor, and $q \geq 0$ is a parameter which we call the plateau factor. $q$ is so called because it is the reason behind the plateau shape near to the left-most part of the distribution. When $q = 0$, Mandelbrot-Zipf distribution degenerates to a Zipf-like distribution with a skewness factor $\alpha$. We will use the Mandelbrot-Zipf popularity distribution in our analysis throughout the thesis.
Chapter 4

The Potential of Cooperative Caching for P2P Traffic

In this chapter, we use our traces to study various aspects of cooperative caching. We start by showing that cooperative caching is needed to achieve high byte hit rates and to save bandwidth on expensive links. Then, we demonstrate the potential gain from cooperation in the two models proposed in this thesis (described in Chapter 2). We study the gain under an optimal offline object replacement policies to show the upper bounds on the gain as well as under realistic online replacement policies. We also analyze the relative gain from cooperation achieved by different ASes.

4.1 The Need for Cooperation

We start our analysis by making the case for cooperative caching of P2P traffic. We choose 10 different ASes from our traces to see what would happen if each deployed a cache to serve P2P traffic originated from a given geographical region. As an example region, we select the West Coast of North America. In this region, we choose the 10 ASes with the largest amount of traffic seen in our traces to make our results statistically significant. For each of these 10 ASes, we find all requests issued from that AS. We use the IP addresses of requests to map a request to an AS using the GeoIP database [7]. We refer to this process as IP-to-AS mapping. Since an AS can span multiple geographical regions, we need to remove
CHAPTER 4. THE POTENTIAL OF COOPERATIVE CACHING FOR P2P TRAFFIC

(a) Cox Communications (AS 11715)  
(b) AT&T-Comcast (AS 1859)

Figure 4.1: Locations of peers in two large ASes in the US.

Table 4.1: Summary statistics for the P2P traffic observed by our monitoring node in 10 ASes in the West Coast.

<table>
<thead>
<tr>
<th>AS#</th>
<th>AS Name</th>
<th>Unique objects (TB)</th>
<th>BHR, 0.5TB cache (%)</th>
<th>BHR, 1TB cache (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2161</td>
<td>AT&amp;T</td>
<td>30.41</td>
<td>11.0</td>
<td>15.1</td>
</tr>
<tr>
<td>9548</td>
<td>Road Runner</td>
<td>14.37</td>
<td>9.3</td>
<td>15.4</td>
</tr>
<tr>
<td>9406</td>
<td>Verizon</td>
<td>12.05</td>
<td>13.4</td>
<td>20.4</td>
</tr>
<tr>
<td>11715</td>
<td>Cox</td>
<td>12.43</td>
<td>12.3</td>
<td>19.1</td>
</tr>
<tr>
<td>1859</td>
<td>AT&amp;T-Comcast</td>
<td>59.67</td>
<td>3.8</td>
<td>6.5</td>
</tr>
<tr>
<td>1782</td>
<td>Shaw</td>
<td>27.96</td>
<td>6.7</td>
<td>11.0</td>
</tr>
<tr>
<td>233</td>
<td>Telsus</td>
<td>19.02</td>
<td>8.4</td>
<td>13.7</td>
</tr>
<tr>
<td>18952</td>
<td>Comcast</td>
<td>17.51</td>
<td>9.3</td>
<td>14.2</td>
</tr>
<tr>
<td>105</td>
<td>Qwest</td>
<td>14.02</td>
<td>8.9</td>
<td>15.1</td>
</tr>
</tbody>
</table>
CHAPTER 4. THE POTENTIAL OF COOPERATIVE CACHING FOR P2P TRAFFIC

requests from outside the West Coast. We do this by finding the geographical locations of the IP addresses of requests again using the GeoIP database (this is referred to as IP-to-geolocation mapping). Then, we remove all requests that are not issued from clients in the West Coast. Table 4.1 lists the names and summary statistics for these 10 ASes.

As Table 4.1 shows, the total size of unique objects observed in each AS is too large to fit in a single cache. Note that the total amount of traffic in each AS is much larger than the total size of unique objects, because the former accounts for the number of times each object is requested. Notice also that these statistics are for the data our monitoring node was able to capture from only one P2P system (Gnutella). Therefore, the actual amount of P2P data for each AS is much larger and indeed the unique objects cannot fit into one cache. In addition, as indicated by the flattened-head of the object popularity distribution in Fig. 3.2, the probability of accessing objects is not concentrated in a few objects as in the Zipf-like distribution case. Rather, it is spread across a much larger number of objects. This means that a single cache may not be able to store enough popular objects to achieve a high byte hit rate.

To confirm the above intuition, we simulate an independent cache for each of the 10 ASes. The cache uses an optimal offline algorithm denoted by iOPT. iOPT looks at the entire trace offline and stores the most popular objects that can fill the cache. This simulation gives us an upper bound on the achievable byte hit rates with independent caches. We run the simulation for two cache sizes: 0.5 TB and 1.0 TB. The achieved byte hit rates are shown in the fourth and fifth columns of Table 4.1. As shown in the table, the optimal byte hit rate achievable with independent caches is less than 10% with a 0.5 TB cache in most cases, and it is slightly improved for a 1 TB cache. This is a fairly small theoretical bound for the byte hit rate, and the practical online replacement algorithms will have even smaller byte hit rates.

Finally, we contrast the need for cooperative caching in the P2P traffic case against that need in the web traffic case. Previous studies, e.g., [39], indicate that an individual cache could store most of the cacheable web objects seen in its trace. This is because web objects have relatively small sizes, and many web caching products limit the size of objects that can be stored locally. The ability of a cache to store most of the requested objects locally diminishes the need for cooperation among web caches. This is not the case for P2P traffic that has very large object sizes [9]. Furthermore, web traffic is very sensitive to latency because it is interactive; clients wait for web pages to be loaded. Cooperative web caching
may increase this latency, since a cache needs to find out whether a requested object is in the cache group and if so where. Increased latency has negative effects on web clients. On the other hand, the P2P traffic usually runs in the background, and therefore is more tolerant for the additional delay that might be added due to cooperative caching.

Therefore, given that individual caches do not have enough capacity to store popular objects to achieve high byte hit rate and that the P2P traffic can easily tolerate the small delay that might result from cooperative caching, we believe that cooperation among caches is needed to enhance the byte hit rate and save more traffic from going on expensive WAN links. It remains to see how much gain we could achieve from cooperative caching, which we study in the following two subsections, and what costs are involved, which we analyze in Sec. 5.2.

4.2 Gain from Cooperation among Caches in Different ASes

We consider cooperation among ASes within the same city. We select two large cities: New York City (NYC) and Los Angeles (LA). In each city, we choose five popular ASes for the analysis. Each AS is assumed to deploy a cache, and the five caches form a cache group. Caches in the same group cooperate with each other to serve P2P traffic originated from the city in which they are located. Traces for caches in different ASes are created using the same IP-to-AS and IP-to-geolocation mapping methods explained in Sec. 4.1. Summary statistics about the five ASes in LA are listed in Table 4.2.

Table 4.2: P2P traffic Statistics for five ASes in LA

<table>
<thead>
<tr>
<th>AS#</th>
<th>AS Name</th>
<th>Unique objects (TB)</th>
<th>BHR, 0.5TB cache (%)</th>
<th>BHR, 1TB cache (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2161</td>
<td>AT&amp;T</td>
<td>18.86</td>
<td>9.6</td>
<td>14.1</td>
</tr>
<tr>
<td>9548</td>
<td>Road Runner</td>
<td>18.29</td>
<td>7.4</td>
<td>12.3</td>
</tr>
<tr>
<td>9406</td>
<td>Verizon</td>
<td>10.91</td>
<td>12.6</td>
<td>20.6</td>
</tr>
<tr>
<td>11394</td>
<td>Charter</td>
<td>7.72</td>
<td>14.1</td>
<td>20.1</td>
</tr>
<tr>
<td>11715</td>
<td>Cox</td>
<td>5.39</td>
<td>14.0</td>
<td>22.8</td>
</tr>
</tbody>
</table>
CHAPTER 4. THE POTENTIAL OF COOPERATIVE CACHING FOR P2P TRAFFIC

Figure 4.2: Comparison between independent and cooperative caching for caches deployed in five ASes in LA. Sample results shown for AS 2161 using: (a) offline optimal replacement policy, and (b) an online replacement policy.

Figure 4.3: Gain from cooperation using cLFU and cLRU replacement policies for: (a) five ASes in NYC and (b) five ASes in LA.
we mean that if an object exists in one of the caches it will definitely be found by any other cache looking for it. We denote this cooperation as cOPT. cOPT looks at the entire traces of all caches and stores the most popular objects in them without exceeding their storage capacities. cOPT does not store an object in more than one cache, because it assumes that the cost of retrieving objects from another cache in the group is smaller than the cost of retrieving objects from outside the group. cOPT represents the upper bound on the byte hit rate in the cooperative caching case.

We conduct several simulation experiments to compare the upper bound on the byte hit rate in the cooperative case (resulted by using cOPT) versus the corresponding upper bound in the independent caching case (resulted by using iOPT). We analyze the performance of the two cache groups in LA and NYC. We vary the size of individual caches from 100 GB to 5 TB and compute the byte hit rate achieved by iOPT and cOPT for each cache in the two cache groups. Fig. 4.2(a) shows a sample of the results for AS 2161 in LA; other ASes achieved similar results. The figure implies that significant gains in byte hit rate can be achieved by cooperation among caches. For example, with a cache of size 1 TB, the byte hit rate of cOPT is about 24%, which is almost double of iOPT. In addition, the gain improves as the cache size increases, which is expected in the future as storage prices keep dropping.

The above results were obtained using the theoretical cOPT and iOPT offline policies to study the upper bounds on the gain. We conduct another set of experiments to study the gain under two realistic (online) replacement policies: cLRU (cooperative Least Recently Used) and cLFU (cooperative Least Frequently Used). cLRU (and similarly cLFU) works as follows. When a cache observes a miss and the object is found in another cache in the group, the object is downloaded from that cache but it is not stored locally. In this case we have only one copy of any object within the cache group. The remote cache updates its data structure as if it had a hit from a local client. In this manner, all caches in the group cooperate to implement a group-wide LRU policy. The setup in this set of experiments is the same as in the previous one, except we compare cLFU versus iLFU and cLRU versus iLRU, instead of comparing cOPT versus iOPT. A sample of the results is shown in Fig. 4.2(b). The figure indicates that substantial improvements in byte hit rates can also be achieved using the realistic replacement policies.

In another experiment, we fix the cache size at 1 TB and compute the potential improvement in byte hit rate due to cooperation among caches in the same group. We compute the percentage of improvement in byte hit rate, which is the difference between the byte hit
CHAPTER 4. THE POTENTIAL OF COOPERATIVE CACHING FOR P2P TRAFFIC

Figure 4.4: Comparison between independent and cooperative caching for caches deployed in AS 11715 using: (a) offline optimal replacement policy, and (b) an online replacement policy.

Figure 4.5: Cooperation among several caches deployed within AS 11715.

The experiments in this subsection show that the byte hit rate could be doubled or tripled in some cases because of cooperation. Considering the huge volume of the P2P traffic, even 1% improvement in byte hit rate accounts to saving in the order of tera bytes of traffic on the expensive WAN links. Therefore, the large savings from cooperation will serve as an incentive for ISPs to deploy caches and enable cooperation among them.
CHAPTER 4. THE POTENTIAL OF COOPERATIVE CACHING FOR P2P TRAFFIC

4.3 Gain from Cooperation among Caches within the same AS

Next, we study the potential gain from cooperation between caches deployed within the same AS. We choose several large ASes with clients distributed over many locations in North America. For each AS, we use the GeoIP database to map the IP addresses of clients in that AS to their geographical locations. The GeoIP database returns the latitude and longitude of a given IP address. We use Matlab to plot these values on the map of North America. Fig. 4.1 shows the distribution of clients in two sample ASes: AS 11715 (Cox Communications) and AS 1859 (AT&T-Comcast), which are two large ISPs in the US. As mentioned in Sec. 2.3, the points of presence (POPs) of an AS are the usual locations for deploying caches. The exact locations of ISP's POPs, however, are not public information. Therefore, we had to approximate the locations of major POPs. Intuitively, a POP will be near the clustering of many clients. In Fig. 4.1, we draw rectangles around apparent clustering of clients in our traces. We assume that the AS deploys a cache somewhere in each of these rectangles. Then, we create traffic traces for each cache by considering only requests from clients falling inside the rectangle in which the cache exists. Notice that the approximate locations of caches do not affect the analysis of cooperative caching, because they could only change the delay between a cache and its clients by a few milliseconds, which is a negligible effect in P2P traffic that runs in the background for much longer periods (minutes).

Similar to the previous case, we study the gain from cooperation under cOPT, cLFU, and cLRU policies versus iOPT, iLFU, and iLRU policies, respectively. Some of the results are shown for AS 11715 in Fig. 4.5; other results are similar. The results confirm that the byte hit rate could be significantly increased with cooperation among caches. Therefore, we can conclude that cooperation improves byte hit rates in both cooperation models considered in this thesis: when multiple caches are deployed within the same AS, and when caches are deployed in different ASes in the same geographical area.

\[\text{We are aware of the Rocketfuel project [21,33], which infers ISP topologies. We did not, however, find the topologies of the ISPs considered in our analysis in the data available from [21].}\]
CHAPTER 4. THE POTENTIAL OF COOPERATIVE CACHING FOR P2P TRAFFIC

4.4 Relative Gain from Cooperation by Different ASes

We noticed in Sec. 4.2 that some ASes may benefit more than others from cooperation. The experiments in that section were limited to only five ASes and they were all fairly large in terms of the amount of traffic seen in each AS. In this subsection, we expand the level of cooperation to include 64 ASes with different sizes and we study the relative gain of each AS. For each of these 64 ASes, we create a trace file that contains requests observed in that AS. Then, we determine the number of objects seen in each trace file. We rank the ASes based on the number of objects that were requested in their corresponding trace files.

We simulate a cooperative cache group that contains all of the 64 ASes. The object replacement policies used are iOPT and cOPT. We measure the byte hit rate achieved by each AS when it cooperates with the others and uses cOPT. We also measure the byte hit rates when ASes do not cooperate with each other, but each of them deploys a cache and uses iOPT. We compute the gain in byte hit rate due to cooperation (cOPT - iOPT) and normalize it by the byte hit rate of iOPT. We summarize the results in the scatter diagram in Fig. 4.6, where the x-axis represents the number of objects seen in an AS and the y-axis represents the percentage of improvement in byte hit rate observed by that AS because of cooperation. The figure indicates that while some ASes improve their byte hit rates by up to 600%, others achieve much less gain from the cooperation, and several ASes actually have negative gain. The figure also shows that the small ASes (in terms of amount of traffic) are mostly the victims. This is because small ASes may replace their own popular objects

![Figure 4.6: Relative gain in byte hit rates when 64 ASes of different sizes cooperate with each other.](image-url)
(based on requests from their own clients) by globally more popular objects. Recall that cOPT maximizes the byte hit rate across all caches, not individual ones. This unequal (and sometimes negative) gain may discourage some ASes to cooperate. We show in Chapter 5 that careful design of replacement policies eliminates the possibility of negative gains and shrinks the gap in the gain achieved by different ASes without sacrificing the total byte hit rate.
Chapter 5

Replacement Policies and Cooperation Overhead

A replacement policy is used when a cache needs to evict an object (or a few objects) to make room for a newly requested one. The replacement policy is a critical component of any caching system, and it is specially so for a P2P cache, because of the large size of objects and therefore the limited number of them that a cache can store. In addition, replacement policies not only affect the total byte hit rate of the caching system, but they also impact the relative gain of individual caches in the cache group, as we will demonstrate in this section. A balanced relative gain by all participating ASes is critical to the success of the cooperative caching for P2P traffic. Furthermore, different replacement policies impose different amounts of overheads, which are important to analyze in order to assess the net benefits of cooperative caching. In this section, we first describe and analyze various replacement policies for cooperative caching. Then, we evaluate the overhead imposed due to cooperation among caches.

5.1 Replacement Policies for Cooperative Caching

We have already described two online replacement policies for cooperative caching in Sec. 4.2: cLFU and cLRU. cLFU and cLRU implement group-wide LFU and LRU policies, respectively. The authors of [38] have proposed a few replacement policies designed for caching P2P traffic. The Least-Sent Byte (LSB) was shown to outperform others in [38].
LSB works for individual caches and it evicts the object that has the minimum number of bytes transmitted from the cache. We consider the cooperative version of LSB, which is denoted by cLSB, as a candidate policy for cooperative caching. cLSB implements a group-wide LSB. All of the cLFU, cLRU, and cLSB replacement policies try to increase the total byte hit rate across all caches. In that sense they are global in nature. Therefore, they may evict locally popular objects from their caches if the global popularity of these objects is not high compared to other objects in the cache group. That is, the byte hit rate of some caches might be sacrificed for enhancing the total byte hit rate of the whole cache group. This uncertainty in the gain from cooperation might discourage ASes from enabling cooperation among caches.

To mitigate this problem, we propose a simple model for object replacement in cooperative caching. We call this model cooperative caching with selfish replacement. Under this model, a cache cooperates by serving requests issued from other caches in the cache group if it has them. The object replacement policy, however, bases its decision to evict objects only on local information of individual caches. We apply the selfish model on the three object replacement policies described above. This results in three new policies: sLFU, sLRU, and sLSB, where the prefix 's' means that the policy is 'selfish'. For example, a cache running sLRU replaces objects that have not been requested for the longest period of time from its clients (i.e., clients from the AS in which the cache is deployed).

We use our traces to analyze the performance of different replacement policies. We implemented the six policies described above: sLRU, sLFU, sLSB, cLRU, cLFU, and cLSB in our cooperative caching simulator. For each policy, we run four simulation experiments:
(i) two for cooperation among caches in different ASes (the five caches in LA and the five caches in NYC), and (ii) two for cooperation among caches within the same AS (AS 11715 and AS 1859). Therefore, we have a total of 24 simulation experiments, and each is run on an 8-month trace of requests. We study the replacement policies along multiple performance metrics. First, we consider the byte hit rate achieved by each cache, which is defined as the number of bytes served from any cache in the group (including the local one) over the total number of bytes requested by the clients behind that cache. Then, we differentiate between bytes served from the local cache and bytes served from other caches in the cache group. We make this distinction because bytes served from other caches typically cost more (in terms of bandwidth and latency) than bytes served locally. We use L-BHR to refer to the byte hit rate achieved by serving objects only from the local cache, and G-BHR refers to the byte hit rate from the whole cache group excluding the local one. Clearly, the total byte hit rate is the summation of L-BHR and G-BHR.

A sample of our results for the cache group in LA is shown in the bar charts in Fig. 5.1. The figure shows the achieved byte hit rates (L-BHR and G-GBR) by two individual ASes as well as the average over all five ASes in LA. Similar results were obtained for the other cache groups in NYC, AS 11715, and AS 1859. The results shown in Fig. 5.1 indicate that cLRU consistently outperforms all other policies in terms of the total byte hit rate (L-BHR + G-BHR). However, the simpler (selfish) sLRU is not too far from it. In fact, sLRU is better in terms of local byte hit rates, which are more valuable. The reason that both LRU versions perform well is that the P2P traffic observes a good degree of temporal locality [27], as popular objects tend to stay popular for some time, then they gradually lose popularity.

Replacement Policies and Relative Gain from Cooperation. We study the effect of the replacement policies on the gain achieved by different ASes. As mentioned in Sec. 4.4, our goal is to find the policies that eliminate negative gains incurred by some ASes, and that reduce the gap between percentage of improvements in byte hit rate between ASes with various sizes. To study this effect, we repeat the experiment described in Sec. 4.4, which simulates the cooperation among 64 ASes with different sizes. The experiment is repeated several times and each time a different replacement policy is used and the gain in byte hit rate of each AS is computed. To facilitate visual comparisons, we represent the results of each experiment as a scatter diagram with the same scales for the x-axis and y-axis. Our results, two samples of them are shown in Fig. 5.2, imply that: (i) the selfish replacement policies sLRU, sLFU, and sLSB eliminate the possibility of negative gains, and
(ii) among the selfish policies, sLRU produces the smallest improvement gap between ASes. For example, comparing Fig. 5.2(a) versus Fig. 5.2(b) indicates that there are a fewer number of ASes that achieve gain less than 10% under sLRU than under sLFU. Moreover, most of the dots representing gains of different ASes for sLFU in Fig. 5.2(b) are spread over a larger range of the y-axis than they are for sLRU in Fig. 5.2(a).

In summary, the experiments in this subsection show that sLRU achieves high byte hit rates and produces the smallest differences in byte hit rate improvement among ASes with different sizes. In addition, as will be shown in the next subsection, sLRU imposes the least amount of overhead. Therefore, we believe that the simple sLRU replacement policy is a good candidate for realizing the potential benefits of cooperative caching for P2P traffic.

### 5.2 Overhead Analysis in Cooperative Caching

Analyzing the overhead imposed by cooperative caching schemes is critical in understanding the net benefit of employing these schemes [4]. It is also important in assessing the potential of deploying cooperative caching in practice. In cooperative web caching, the overhead is one of the factors that plagued its wide deployment. We show below that this is not the case in cooperative caching for P2P traffic.

By overhead we mean the additional number of bytes transmitted beyond the transfer of the requested objects themselves. As mentioned in Sec. 2.2, we use the Internet Cache...
Protocol (ICP) [36, 37] to facilitate communication and object look up among caches. We have implemented the ICP protocol in our cooperative caching simulator. We compute the overhead imposed by different replacement policies. As before, we consider the two caching groups in NYC and LA and the two caching groups within AS 11715 and AS 1859. We count the number of bytes that are exchanged by the ICP protocol, and divide that number by the total number of transferred bytes. The results for the cache group in NYC and LA are shown in Fig. 5.3. The figure implies that the maximum overhead imposed by cooperative caching is less than 0.003% for all policies, which is indeed negligible. The figure also shows that sLRU has the smallest overhead. This is because sLRU has higher local byte hit rate (L-BHR), as discussed in the previous subsection. Local hits do not impose overhead, because they do not require sending ICP queries to other caches.

In contrast, the overhead in cooperative web caching is non-negligible. This is because cacheable web objects are much smaller than P2P objects [9,27]. In addition, sending queries and waiting for replies add noticeable delays, which may negatively impact web clients and reduce the benefits from cooperative web caching. Download sessions in P2P systems, on the other hand, usually run in the background and take long periods. Furthermore, unlike web clients, P2P clients expect some delay for the system to locate their requested objects at some other peers. Thus, the look up delay is not a critical issue in P2P systems, making a stronger case for the cooperative caching of P2P traffic.
Chapter 6

Analytical Study

In this chapter, we analyze the benefits of cooperative caching for P2P traffic using a simple analytical model. Although simple, the model captures the most important parameters in the system, including object popularity, number of caches, and the relative cost of internal and external links.

The model assumes that there are $m$ caches deployed in different ASes and cooperating with each other. We refer to these $m$ caches as a cache group. We denote the cost of retrieving an object by a client from: (i) its local cache as $\tau_l$; (ii) another cache in the group as $\tau_g$; and (iii) an external source (i.e., other peer(s) in the P2P network) as $\tau_e$. We use abstract costs to make the model more general. For example, if the costs are set as delays, the model can be used to analyze the average latency perceived by clients (as it is usually done in web caching). On the other hand, if we set the costs as dollars by bytes, the model would allow us to analyze the average saving in the operational costs observed by an AS because of cooperative caching. Analyzing the saving is more relevant to ASes who are interested in deploying caches to reduce the burden imposed by the sheer volume of the P2P traffic.

For the feasibility of the analysis, we make the following assumptions. In Section 6.4, we relax most of these assumptions and validate the results of our analysis using simulations. First, all caches have the same storage size. Second, since objects in P2P systems are typically divided into equal-size segments, we carry out our analysis in terms of segments. All segments of the same object are assumed to have the same popularity. This is not unrealistic in P2P file-sharing systems, which unlike video streaming systems, download segments in random order and all segments are needed for the object to be useful. Therefore, for clarity
of the presentation, we treat segments as small objects. Given the first assumption, each cache can store up to $k$ segments. Our third assumption in the analysis is that caches observe similar relative popularity of objects. This is also not unrealistic in P2P systems which have no sense of network locality and in which popular objects typically attract global client populations. The goal of our analysis is to determine the saving achieved due to cooperation. To do this, we first compute the cost in the independent caching case. Then, we compute the cost in the cooperative case and compare them.

6.1 Cost in Independent Caching

For the independent caching case, no cooperation among caches is performed. We assume that each cache uses a popularity-based local replacement policy such as the least-frequently used (LFU) policy. Given the above assumptions, the cost of serving $N$ objects from clients in an AS is given by:

$$C_{ind} = \tau_l \sum_{i=1}^{N} f(i) + \tau_e \sum_{i=k+1}^{N} f(i),$$

(6.1)

where $f(i)$ is the probability of accessing the object at rank $i$, and $\tau_l$ and $\tau_e$ are the costs of downloading an object from the local cache and from an external source, respectively. Notice that objects are ranked based on their relative popularity such that $f(i) \geq f(j)$ for all $i < j$. The above equation represents the average cost, because, for large $N$, in the steady state the cache stores the $k$ most popular objects. The second term in (6.1) is the cost of retrieving objects $k+1, k+2, \ldots, N$ from external sources because the first $k$ objects are stored in the cache. The first term is the cost of all objects because they all have to impose local $\tau_l$ cost regardless they are stored in the cache or not. Notice also that because of the lack of cooperation and by the similar relative popularity of objects assumption, all caches will end up storing the same top $k$ popular objects.

6.2 Cost in Cooperative Caching

In the cooperative caching case, when a cache receives a request for an object that it does not store, it first forwards this request to other caches in the cache group. If any of them has the requested object, the object is served to the client (with a cost $\tau_g + \tau_l$). If otherwise, the object is downloaded from an external source (with a cost $\tau_e + \tau_l$). In addition, the caches
coordinate the replacement of objects. In particular, when an object eviction needs to be made by a cache, that cache chooses the least-popular object in the cache group. Given these assumptions, the cost of serving \( N \) objects from clients in an AS in the cooperative caching case is given by:

\[
C_{coop} = \tau_l \sum_{i=1}^{N} f(i) + \tau_g \frac{m-1}{m} \sum_{i=1}^{mk} f(i) + \tau_e \sum_{i=mk+1}^{N} f(i). 
\] (6.2)

There are two differences between the cost in the cooperative case, Eq. (6.2), and the cost in the independent case, Eq. (6.1). The first difference is that the middle term in (6.2), which represents the additional objects stored in the cache group. Since \( \tau_g \leq \tau_e \), the cooperative caching model will keep at most one copy of any object in the cache group. Thus, the total number of objects stored in the cache group is \( mk \). Consider any cache in the cache group. If this cache receives a request for any of the \( mk \) objects stored in the cache group, it will serve it locally with probability \( \frac{1}{m} \), and from another cache in the group with probability \( \frac{(m-1)}{m} \). In the latter case, there is an additional cost of \( \tau_g \) to serve the object. The second difference is that the summation in the third term in (6.2) starts from \( mk+1 \), not from \( k+1 \) as in (6.1).

We define the relative saving in cost due to cooperation as \( \Psi = (C_{ind} - C_{coop})/C_{ind} \). To study the saving, we need an object popularity model. As we mentioned in Chapter 3, the Mandelbrot-Zipf distribution captures the popularity of objects in P2P systems fairly accurately. Thus substituting the Mandelbrot-Zipf distribution in (6.1) and (6.2), we get:

\[
\Psi = \frac{\tau_e \sum_{i=1}^{mk} \frac{1}{(i+q)^\alpha} - \tau_g \frac{m-1}{m} \sum_{i=1}^{mk} \frac{1}{(i+q)^\alpha}}{\tau_l \sum_{i=1}^{n} \frac{1}{(i+q)^\alpha} + \tau_e \sum_{i=mk+1}^{n} \frac{1}{(i+q)^\alpha}}. 
\] (6.3)

To simplify the above equation, we set the local cost \( \tau_l \) to 0, because it is typically much smaller than the external cost \( \tau_e \) and the inter-cache cost \( \tau_g \). In addition, \( \tau_l \) is incurred in both the independent and the cooperative caching cases. Thus, it is not a differentiating factor in determining the potential gain of cooperation. We define \( \tau = \tau_g/\tau_e \), which is the cost of serving an object from the cache group to the cost of serving it from an external source. We carry out our analysis in terms of \( \tau \). We also denote the expression \( \sum_{i=x}^{y} \frac{1}{(i+q)^\alpha} \)
by $S(x, y)$ for clarity. Thus, the saving in the cost due to the cooperation among $m$ caches is given by:

$$
\Psi = \frac{1}{S(k + 1, N)} \left[ S(k + 1, mk) - \tau \frac{m - 1}{m} S(1, mk) \right].
$$

(6.4)

The gain from cooperative caching in the above equation models the three most important aspects of the system: (i) traffic characteristics in terms of $\alpha$ and $q$ of the Mandelbrot-Zipf popularity distribution; (ii) network characteristics captured by $\tau$; and (iii) cache group characteristics captured by the number of caches $m$ and the capacity of each cache $k$ in the cache group. In the following section, we conduct numerical analysis of the impact of these parameters on the gain from cooperation.

### 6.3 Numerical Analysis

We numerically analyze the gain (or saving in the cost) due to cooperative caching given by (6.4). We consider one parameter at a time, while fixing all others at reasonable values. We start by studying the impact of traffic characteristics on the gain from cooperative caching. In our analysis, the traffic characteristics are modeled by the skewness parameter $\alpha$ and the plateau parameter $q$ of the Mandelbrot-Zipf popularity distribution. We fix the number of caches $m$ at 8, and $\tau$ at 0.1. We also fix the storage capacity of individual caches. The storage capacity is represented as the ratio of objects that can be stored in the cache $k$ to...
the total number of objects \( N \). We call \( k/N \) the relative cache size, and it is set to 5%.

In Fig. 6.1(a), we vary \( \alpha \) between 0.4 and 1.2. Larger \( \alpha \) values mean that the popularity curve is more skewed, which implies that the top-ranked objects receive higher fractions of the requests. We plot the gain from cooperation \( \Psi \) versus \( \alpha \) for three different \( q \) values: 0, 50, 100. As shown in the figure, a significant gain of up to 50% can be achieved by cooperative caching.

Fig. 6.1(a) demonstrates an interesting feature in cooperative caching: the gain is more robust against larger values of the skewness parameter \( \alpha \) for P2P traffic than for web traffic. Web traffic follows a Zipf-like distribution [2], which is a Mandelbrot-Zipf with \( q = 0 \). Thus, as shown in Fig. 6.1(a), when \( \alpha \) increases the gain from cooperation diminishes quickly for web traffic. On the other hand, the plateau factor \( q \) somewhat mitigates the effect of large \( \alpha \) values for P2P traffic. This is because the Mandelbrot-Zipf distribution has a flattened head (see Fig. 3.2), which indicates that, even with large \( \alpha \) values, the requests are spread over more objects in the head of the popularity distribution. Since more objects in the head of the popularity distribution require larger storage capacities, cache cooperation will be more beneficial.

Next, we analyze the impact of changing the plateau parameter \( q \). In Fig. 6.1(b), we vary \( q \) between 0 and 100, and plot the gain from cooperation \( \Psi \) for three values of \( \alpha \). The results in Fig. 6.1(b) indicate that small values of \( q \) reduces the gain from cooperation. This is because when \( q \) is small the head of the popularity distribution is less flattened and the impact of the skewness parameter on the gain is higher. Nonetheless, for typical \( q \) values (more than 10 as shown in [11]), the gain from cooperation is at least 35%. Considering the huge volume of the P2P traffic, even 1% gain would amount to significant savings in bandwidth and operational costs for ISPs.

Next, we analyze the impact of the network and cache group characteristics on the gain from cooperative caching. Fig. 6.2 summarizes the results, where in each subfigure we fix all parameters except one and plot the gain for three representative object popularity distributions. Fig. 6.2(a) demonstrates large gains from cooperation for wide ranges of the relative cache size. Gains from cooperation is also demonstrated in Fig. 6.2(b) and Fig. 6.2(c) for different number of caches and various \( \tau \) values. Up to 80% gain could be achieved from cooperative caching. In addition, Fig. 6.2(c) shows that the gain from cooperation is still achievable even if the relative cost of serving an object from the cache group to the cost of serving it from external sources (\( \tau \)) is fairly large. This is more apparent when the
6.4 Analysis Validation using Simulation

In this section, we relax the assumptions made in the analysis, and we verify that our results still hold. We generate synthetic traces with different characteristics. The replacement policies are the same as in the analysis: iLFU for independent caches, and cLFU for cooperating caches.

In the synthetic traces, objects do not have the same size, rather the object size follows
a distribution. We choose a distribution for object sizes based on our real traces. That is, we create $10^5$ objects and randomly choose their sizes from objects in the real traces. We randomly assign popularity for these objects based on the Mandelbrot-Zipf distribution with different values for $\alpha$ and $q$. We use 8 caches in the simulation, and a total of $10^6$ requests are generated from all of them. We also randomly assign $\tau$ values for the caches, i.e., the cost of downloading an object from another cache is no longer a constant. As in the analytical analysis, we define the gain from cooperation, $\Psi$, as the total cost of $iL$FU minus cLFU normalized by $iL$FU.

We study the impact of all parameters on the gain $\Psi$, including $\alpha$, $q$, cache size, and $\tau$. In Fig. 6.3, we vary the parameters of the popularity distribution and measure the achieved gain. The figure shows very similar pattern as Fig. 6.1: the gain in cooperative caching is robust against wide ranges of $\alpha$ and $q$ values. The figure also shows that for $q = 0$ (i.e., web traffic), the gain drops quickly as the skewness parameter $\alpha$ increases. In Fig. 6.4 we vary the network parameters, cache sizes and relative link costs. Again, simulation results are very similar to the analytical results in Figs. 6.2(a) and 6.2(c). The results confirm that in P2P systems where $q$ is greater than 0, gain of cooperation is significant even with very large cache size. Also, the gain is robust against the different network link costs.

In summary, the results from our simulations with relaxed assumptions are similar to those derived from the analytical analysis. Both results confirm the viability of cooperative caching for P2P traffic.
Chapter 7

Implementation in a Cache Prototype

In the previous chapters, we described the potential of cooperative caching, replacement policies, and analytical studies for proxy caching of P2P traffic. In this chapter, we briefly describe a prototype cache for P2P traffic (called pCache) developed by members of the Network Systems Laboratory at Simon Fraser University. As part of this thesis work, we implemented several object replacement policies in pCache.

7.1 Overview of pCache

pCache is designed to improve the traffic locality by storing the requested files on its local storage and serve the files to the clients on the next requests. Our cache is programmed to be deployed at an ISP’s gateway to observe all the traffic between local clients and the external network, and intervene these connections if required. Fig. 7.1 shows the connections between local clients and the external network, say Internet. All the traffic from the local clients to the external peers, and vice versa, should pass through the gateway server. In turn, the gateway can forward the traffic to the pCache server. The specification and portion of forwarded traffic should be configured in the gateway server with respect to the gateway policies and objectives. For example, a Web proxy cache only needs the traffic on port 80. However, the pCache needs more number of ports to be forwarded, because most of the P2P protocols do not use a fixed port number for their traffic. Rather, they usually change their
ports randomly to bypass the blocking firewalls.

pCache design has the following merits which make it outstanding and different from other proxy caches:

- **Partial caching:** Many huge objects, such as video streams, are distributed over peer-to-peer (P2P) file-sharing systems. Caching all of these objects in their entirety is impractical, because a few of them can easily occupy the whole storage space, thus preventing other objects from entering. In general this leads to low hit-ratio and poor performance. pCache system adopts partial caching scheme to cope with these huge objects. In other words pCache determines to cache zero or several segments for an object.

- **Transparency:** pCache is transparent to users as it intercepts and inspects inter-AS traffic, then processes and serves any P2P requests in it. This allows users to benefit from pCache without configuring their P2P software.
• Scalability: pCache offloads data transmission as much as possible in order to scale for more concurrent download sessions. It achieves so by two means. First, a requested segment that is not in the proxy cache and is not admitted by the proxy cache can neither be served by us nor be useful to us. To ease the burden of relaying such a request, pCache instructs the gateway server to bypass that transfer. For example, in Fig. 7.1, the data transfer between workstation E2 and I1 is bypassed from pCache. Second, segments that are available on peers in our AS are more preferable than segments on the Internet and in our proxy cache. This is because redirecting requests to local peers reduces the load on pCache. For example, in Fig. 7.1, the data transfer between workstation J2 and I1 incurs no load on pCache, as the transfer is redirected from some workstations on the Internet.

These advantages enable pCache to serve P2P requests in a flexible way.

pCache has four main modules: Connection Manager, Traffic Identifier, Cache Manager, and Disk Manager. Connection Manager is responsible for accepting the incoming connection and pass them to the traffic identifier. It also receives the data from the Cache Manager. Part of incoming packets as well as the data received from the Cache Manager should be sent back to the gateway. The gateway needs to send these packets to the appropriate destination in turn. Traffic Identifier investigates the incoming traffic in order to identify a known P2P protocol. This traffic can be recognized as a non-P2P traffic or a non-cachable P2P traffic. In both cases, Traffic Identifier should inform the Connection Manager to forward the traffic to the appropriate destination without passing it to the Cache Manager. On the other hand if the traffic is a known P2P protocol, the useful data will be extracted from the packets. Traffic Identifier encapsulates this meaningful data inside a Segment request message. A segment message contains various fields which are of interest to the Cache Manager. The most basic fields are the object ID and requested range. Each object ID points to only one object and it is used to identify an object uniquely. Also, since a request in most of the P2P protocols comes for a specified range within the requested file, the Segment message should contain the requested range as well as the object ID. Then, the Connection Manager would query the Cache Manager for the requested Segment message. Now the Cache Manager should search its contents for the requested Segment.

Upon receiving a query for a requested segment the Cache Manager does the following. First, it compares the requested segment against its content. Three different situations may occur: (i) no part of the requested segment is available in the cache, (ii) the whole
CHAPTER 7. IMPLEMENTATION IN A CACHE PROTOTYPE

Figure 7.2: Each received request might be truncated into different parts: Cached, Admitted, and Relayed.

requested range is available in the cache, and (iii) one or more parts of the requested range are available but not all of it. In the first case, the requested segment should be downloaded from external peer and based on the object replacement policy the whole requested range or a part of that will be stored in the cache. In the second case, the available data will be used by Connection Manager to encapsulate it within a response message and send it back to the requester. In this case no message would be sent to the external network. Finally, the third case is a combination of both previous cases, as it is depicted in Fig. 7.2. Considering that the requester expects to receive the requested segment in only one response message, all the available and requested ranges should be assembled inside one response message. Basically, the Cache Manager sends the Connection Manager two lists of segments, a list of available segments and a list of requested segments. These requested segments can be all the files excluding the available parts, or they can be just a portion of unavailable parts. Note that this decision is made by the object replacement policy.

At last, the Disk Manager provides high-speed access to the Cache Manager and Connection Manager to read and write the data. The storage utilizes a user-space file system to accelerate random segment access of large objects. Disk is filled by Page Clusters. Page clusters such as segments contain object ID and byte range. They also contain other fields like read and write method which find the actual place of the file on the disk, and a reference counter which shows that how many processes are currently using this page cluster.

In the rest of this chapter we continue with more detailed description on the Cache Manager. The most important aspect of the Cache Manager is the object replacement policy; which is responsible for admitting the new objects as well as removing the stored objects to make room for new objects. As we discussed in Chapter 5, replacement policies
have a significant rule in caching of P2P traffic because the P2P objects are significantly large and we definitely cannot keep a reasonable portion of them in a cache all the time. Thus, the replacement policies should be chosen wisely to make the best of the limited cache storage. In the next section we describe the replacement policies in more detail, and for each of them we discuss its implementation ideas and issues.

7.2 Implementation of Replacement Policies

When the Cache Manager receives a request, if the requested file is available in the cache, the cache data structure should be updated. Otherwise, the Cache Manager may request the file for admission. When a new file needs to be stored in the cache and it does not have enough available room for it, one or more file need to be evicted from the cache. All these are the basic responsibilities for the replacement policy. Thus, replacement policy makes the decisions for every file admission and file eviction.

pCache currently contains four popular replacement policies and any other policies can be easily added to it as a new class which implements the defined interface. The replacement policies already included in pCache are LRU, LFU, LSB, and P2P. LRU (Least Recently Used) removes the object with farthest access time from the cache to make room for the newly admitted object. LFU (Least Frequently Used) removes the object with the least number of hits. LSB (Least-Sent Byte) [38] evicts the object which has transmitted the least number of bytes. Finally, P2P [27] is the algorithm designed for partial caching of P2P traffic. In [27], P2P algorithm is shown to outperform other replacement policies. However, P2P algorithm is compared only against full object caching policies and it is not compared with the modified partial object caching policies. pCache takes advantage of the partial caching implementation of all these four algorithms.

Here we describe the detailed implementation of the Cache Manager. Cache Manager is an abstract class which all replacement policies need to extend. The essence of these replacement policies is the data structure they use to store the Item objects. This data structure depends on the replacement policy. For example, LRU uses a FIFO (First-In First-Out) queue to find the least recently used object easily. Items are the objects stored in the cache data structure. An Item object inside the cache corresponds to an actual file. Each Item contains at least an object ID and a list of PageClusters. This list makes the pCache different from other Web proxy caches, since it is used to store an object partially,
CHAPTER 7. IMPLEMENTATION IN A CACHE Prototype

and not entirely. All the PageClusters which are allocated in the cache for the same file, are included in this list. Therefore, having an Item, Cache Manager can easily find the available parts and also add newly admitted parts to it. The basic structure of the Item is shown in the following.

```
std::string objId; // 32 char
long long size; // Cached size in byte
std::set<PageCluster*, DerefereceLess> pcList; // list of PageClusters
bool operator<(const Item &other) const; // less operator defined for sorting
```

The basic functionalities for the Cache Manager are: Query, Free, and Abort. Query method receives a Segment object as an input and responds by a CacheResponse. Segment is already defined in previous sections. A CacheResponse is created based on the requested Segment and contains two lists: a list of cached PageClusters and a list of admitted PageClusters. Each byte on the cache storage belongs to one and only one PageCluster. We can read the data on the disk through the PageCluster’s Read method and data can be written on the disk using PageCluster’s Write method. Disk manager would create new PageCluster for the admitted segments and send these PageClusters through the Cache Manager to the Connection Manager. Connection Manager is responsible for filling these PageClusters by downloading the data from external peers. In case of any connection problems, if the admitted object has not been downloaded completely, Connection Manager should call the Abort method for the aborted PageCluster. Abort method will terminate downloading of the PageCluster and eventually the PageCluster will be erased completely. Followings are the interface for basic functionalities of the pCache.

```
CacheResponse* query(Segment*); // receive a request for a file range
void free(); // evacuate the cache to the low watermark
void abort(PageCluster*); // abort a PageCluster download session
```

When the cache is full and a new PageCluster should be admitted into the cache, Cache Manager will call the Free method to evict a few PageClusters from the cache and make room available for the newly arrived one. pCache uses high-low watermark approach to free the required space. In high-low watermark approach, a high threshold, say 90%, and a low
threshold, say 80%, is defined. Whenever the cache size cross over the high threshold, the cache evacuation method should be called to offload one or more objects. Then the evacuation method, which is the Free method in pCache, starts to evict objects from the cache one by one until the cache size reaches the low threshold. Since pCache uses partial replacement policies, the evacuation method works based on the PageClusters rather than files. Thus, it removes the PageClusters one by one and if an Item remains with no PageCluster, the Item will be removed entirely from the cache. The high-low watermark approach has two main advantages compared to the case which for each admitted item, one item has to be removed. First, a high watermark prevents the cache from the high utilization. Studies show that utilizing the disk to a high extent would dramatically degrade the disk performance, mainly because of the high disk fragmentations. Second, evacuating the disk to a low watermark would reduce the overhead in finding the object which is the candidate to be removed, since finding a bunch of objects at one shot has lower cost than finding each of them one by one. The high-low watermark approach can also decrease the cost of locking and unlocking the mutex. Since many threads may need to access the Cache data structure at the same time, any thread that performs a change on the data structure must first lock the data structure to prevent it from parallel access. Therefore, high-low watermark approach minimizes the overhead by locking the data structure once for several object replacements. In the following, each of the implemented replacement policies will be discussed in detail.

### 7.2.1 Implementation of LRU Policy

LRU selects the least recently used object as the remove candidate. In order to implement the first-in first-out nature of the LRU policy, we use the Queue class in C++ Standard Library. Admitted objects would be pushed on the top of the queue and the remove candidate would be the object at the bottom of the queue. Objects are stored inside the queue as Items. In the previous sections, we described the Item as a class which contains an object ID as a list of available PageCluster(s). All the Items inside the queue should contain at least one PageCluster in their list or they would be removed automatically from the queue. Upon a cache hit for any segments within an object, the object will be removed from its current place in the queue and will be pushed on the top of the queue. LRU replacement policy inherits its other functionalities from the Cache class.
7.2.2 Implementation of LFU Policy

LFU keeps an access frequency parameter for each object and removes the object with the least frequency of access. In order to optimize the frequency lookup, we take advantage of a priority queue. Priority queue, which is usually implemented by a heap, has the best performance in finding the minimum values. Using the priority queue, the minimum valued object can be found in $O(1)$. This instant lookup property would justify the use of priority queue in implementing the LFU policy, where we continuously need to search for the object with the minimum number of access in the past. Each Item contains a frequency variable which counts the number of access to it since the creation. Item class defines a comparator operator to sort the Items based on their access frequencies. Priority queue uses this comparator operator to keep the least frequently accessed Item at the top of the queue. The access frequency is initialized to 1 when an Item is created for the first time. Then the frequency will be incremented by one, upon receiving a request for the object or a part of the object. Original LFU policy would not keep a history of access frequency for every objects even if they had been removed from the cache. Thus, if any previously removed object is admitted again, its access frequency will be initialized to 1, not to its value before removal. Again, LFU replacement policy inherits most of its other methods from the Cache class.

7.2.3 Implementation of LSB Policy

LSB is the outperforming algorithm in terms of byte hit rate on the study by [38]. LSB uses the transmitted number of bytes as the measurement for ranking the objects. We use a priority queue to find the least sent bytes object in instant time. Like the LFU policy, Items in LSB policy keep track of the number of bytes they have been sent since they admitted into the cache. Whenever an Item is used to serve a hit, its sent bytes value will be increased by the size of the requested segment. The Item with minimum number of sent bytes will be removed from the priority queue and the disk to make room available for the object admitted recently.

7.2.4 Implementation of P2P Policy

P2P algorithm introduced in [27] is shown to outperform all the non-partial caching replacement policies designed for Web caching. P2P algorithm has a major difference with
all the replacement policies we discussed so far. All other policies store the requested range entirely. However, P2P algorithm starts to add data to a stored object gradually. Upon receiving the first request for an object, P2P algorithm store only a small portion of the request. Then, after receiving further requests, it start to add more of that object to the disk. P2P algorithm uses a function, named $\gamma$, to sort the objects. It always remove the object with the lowest $\gamma$ value. Hence, we use a priority queue to keep the object with the lowest $\gamma$ at the top. We implement the partial caching mechanism in the following manner. Whenever a request comes for a part of an Item $i$, with the probability of $\gamma_i/\gamma_1$ the new part will be admitted to the cache. Assuming that objects are sorted by their $\gamma$ values incrementally, for $i < j$, $\gamma_i \geq \gamma_j$. Thus, in the steady state, cache will contain $\gamma_i/\gamma_1$ portion of the object $i$. $\gamma$ would be updated based on the P2P algorithm description.

7.3 Evaluation

We use trace-based measurement studies to evaluate the replacement policies implemented for the pCache. We choose a 3-month trace belonging to AS 9406, Verizon, to measure the performance of our cache. Following comes the detailed description of our experiments.

7.3.1 Setup

First we describe the experimental setup for evaluating the pCache replacement policies. To faithfully recreate the actual P2P traffic, we adopt 8-month trace files described in
Chapter 3. In particular, we present the sample results based on the Verizon trace in North America, while other trace files lead to same conclusions. We implement a micro-level request generator in C++, which takes the trace file and produces the smaller segments in the following way. We assume all objects are either entirely or partially downloaded in the format of small segments. We define a maximal segment size as an input parameter of our experiments. We then start from the beginning of the requested object and generate segment requests no larger than the maximal segment size. Effectively, we divide the whole object into many small segments. Our request generator is quite close to the segmentation scheme used in real P2P applications, e.g., BitTorrent clients often generate 16 kilo-bytes request.

We should mention that, object requests in the original trace files contain a single timestamp. A simple, but inaccurate, request generator may assign the same timestamp to all segments belonging to that object. This is inaccurate because downloading an object often takes hours, if not days, to finish. A better way for generating the requests is to randomly choose a download speed for each object, then calculate timestamps for individual segments. Therefore, segments for different object requests may interleave with each other. To preserve the ordering among segment requests, our request generator employs a request priority queue, that called the event queue. Requests for segments are sorted based on their calculated timestamps, which enables us to process segment requests in-order. We use a high-low watermark for the event queue to push the requests to and pop them out of the queue. Thus, when the event queue size reaches the high watermark, requests would be popped from the queue one by one based on their timestamps until the queue size reaches the low watermark. At any moment, the request with the minimum timestamp would be at the top of the event queue.

We conduct our experiments on a commodity PC running Linux 2.6.20. This machine has two Quad Core Intel(R) Xeon(R) CPU X5355 at 2.66 GHz each with 4096 KB L1 cache. To measure the replacement policies performance and compare them with each other, no disk access is required. Therefore, we disable the disk access methods in order to both speed up the measurement process and also make the disk size as large as possible. Considering that our hardware limits the disk size to 250 GB, we configure our pCache to work as if it has up to 1 TB storage by disabling the disk access methods. Disabling these features will not change the byte hit rate of the pCache. Requested segments come at the maximum size of 1 MB. We use the first 500,000 object requests in the Verizon trace to generate segment
Figure 7.4: Byte hit rate under four common replacement policies for different cache sizes in Verizon during 3-month measurement.
requests. We also use a download aborting mechanism to model the real P2P application behavior. According to [9] 66% of the P2P download sessions would be aborted in the middle of downloading. Thus in our measurements, 66% of the object requests were terminated at a uniformly random point in the middle of downloading. We present our results in the next section.

7.3.2 Results

In our experiments, we encounter more than 14.5 TB of traffic imposed by 500,000 requests. Requests come for a various range of object sizes, from a few hundred kilo-bytes up to a few giga-bytes. Distribution of these object sizes is depicted in Fig. 7.3.

In order to evaluate the differences between the replacement policies and to observe the effect of the storage size, we ran the pCache with various disk capacities and different replacement policies. Result of these experiments can be seen in Fig. 7.4. In this figure we plot the total byte hit rate as well as the pCache’s memory byte hit rate. pCache takes advantage of a 512 MB memory to facilitate the access to the several hundreds of the PageClusters. This memory is able to use different policies for storing the PageClusters. In these experiments, we always use the LRU policy to remove the least recently used PageCluster from the memory. we define the memory byte hit rate as the number of bytes served from the cache memory divided by the total requested traffic. Comparing Fig. 7.4(a) with other figures shows that LRU outperform the other policies in terms of byte hit rate. This comparison also shows that using a very small cache size (in order of few GB), the
LRU policy can achieve a reasonable byte hit rate. The reason behind this behavior can be the time locality between the requests for an object in our traces. On the other hand, frequency-based policies like LFU and LSB have an aggressive behavior in using the cache capacity. As the cache size increases, the byte hit rates of LFU and LSB grow rapidly. Fig. 7.5 compares these four replacement policies when the cache size is fixed at 1 TB. Fig. 7.6 demonstrates the results of these experiments during the trace time. Comparing Fig. 7.6(a) and Fig. 7.6(b) again shows that the increase of cache capacity has more impact on frequency-based policies. With a 1 TB cache size, LSB outperform LRU during close to half of the simulation time.

Fig. 7.7(a) compares the three large ASes during a short measurement time. Results show very different request patterns for different ASes. Fig. 7.7 shows the behavior of the cache in the transition phase for LRU. Byte hit rate catches up to its stable value a short time after the cache is full. The figure also shows the impact of using high-low watermark approach for cache evacuation on the cache size.

Fig. 7.8 shows the average life time of the evicted objects inside the cache. For P2P algorithm on average it takes more than 25 days for an object to be removed from the cache (if it ever removed). The reason behind this behavior is the partial storing nature of the P2P algorithm. It is unlikely for an object to be evicted from the cache entirely. In P2P policy we store a few segments from every object in a large pool of objects.
CHAPTER 7. IMPLEMENTATION IN A CACHE PROTOTYPE

Figure 7.7: \textit{pCache} trace-based measurement using 100GB storage, (a) BHR of 3 large ASes using LRU policy (b) BHR of AS 9406 in cache warm up state.

Figure 7.8: Average lifetime of the evicted objects in the cache.
Chapter 8

Conclusions and Future Work

In this chapter, first we summarize this thesis. Then, we briefly describe the ways which we can extend this work in the future work section.

8.1 Conclusions

In this thesis, we considered the potential gain of cooperative caching for P2P traffic. We proposed two models for cooperation: (i) among caches deployed in different ASes located inside the same geographical region such as a city, and (ii) among caches deployed within a large AS such as an ISP network. In both models, caches cooperate to save bandwidth on expensive WAN links. We collected traces from an eight-month measurement study on a popular P2P system. The traces describe object requests that would have been seen by many caches if they were deployed in ASes operating in different geographical regions and have different number of clients. We designed several trace-based simulation experiments to rigorously analyze various aspects of cooperative caching. Our results show that cooperative caching is viable for P2P traffic, because it could improve the byte hit rate by up to 330% in some cases. Considering the huge volume of the P2P traffic, even 1% improvement in byte hit rate accounts to saving in the order of tera bytes of traffic on the expensive WAN links. The large savings from cooperation could serve as an incentive for ISPs to deploy caches and enable cooperation among them. Our results also show the overhead imposed because of cooperation among caches is negligible, less than 0.003% of the total traffic.

In addition, we proposed simple models for object replacement policies in cooperative caching systems. These models allow an individual cache to cooperate with other caches to
serve requests from each other, but without harming its own performance. This is done by basing the decision to replace an object from the cache only on local information from that cache. We showed that the proposed replacement policies not only eliminate the possibility that some ASes suffer negative gains from cooperation, but they also shrink the gap in the gain achieved by ASes with different sizes. This is achieved without sacrificing the total byte hit rate.

In addition, we used a simple analytic model to assess the gain from cooperative caching under different traffic and network characteristics. Our model confirms that substantial gains from cooperation are possible under wide ranges of traffic and network characteristics. We also validated the results from our analysis using simulations, where most of the assumptions made in the analysis are relaxed. Furthermore, we implement several object replacement policies in a real prototype cache, and we compare their performance using P2P traffic traces collected from a widely-deployed P2P system.

8.2 Future Work

The research presented in this thesis can be extended in several directions. We summarize some of these directions below.

- In order to understand the behavior of P2P traffic and extend our knowledge about the characteristics of the P2P traffic, we need to collect more traces from real request messages. This goal cannot be achieved without close collaboration between our research group and a large ISP. For this purpose, we need to capture the request messages at the ISP's gateway. These new traces can be more accurate and informative in compare with our current traces which are consist of the QUERYHIT messages rather than the request messages.

- In this thesis research study, we compare several replacement policies in terms of byte hit rate and other characteristics. Though some of them outperform the others, non of these replacement policies are designed for the case of cooperative caching in P2P networks. Considering our knowledge of cooperative caching, we can come up with a list of requirements for the appropriate policy. This policy can be derived with a few modification on current replacement policies to make it more suitable for the cooperative case.
• We successfully test our proxy cache server in our research lab using different subnets and real P2P application clients. We also need to deploy the pCache on a gateway server to measure its performance and verify its scalability. Further improvements in the performance, security, scalability, and validity of pCache can be achieved after its establishment in a real high-speed gateway.

• Traffic in P2P networks is generated by several P2P protocols like BitTorrent, eMule, Gnutella, KaZaa, and etc.. These protocols have their own overlay networks and object repositories. However, a high percentage of the popular files are the same in all these networks. Employing a cross platform caching mechanism, can significantly reduce the disk utilization by removing the duplicated copies of the objects in different P2P networks. This improvement in disk performance will results in a higher byte hit rate.

• In this research study, we consider two models for cooperation among caches, cooperation among caches in different ISPs in a small geographical region, and cooperation among caches within a large ISP. We can extend the benefits of cooperation by combining these two models. For example we can have a hierarchical caching structure in which some high bandwidth links within an ISP connect a bunch of caching clusters together.
Bibliography


